```
#-------------------------------------------------------------------------------
-----------------------
#NEURONRAIN ASFER - Software for Mining Large Datasets
#This program is free software: you can redistribute it and/or modify
#it under the terms of the GNU General Public License as published by
#the Free Software Foundation, either version 3 of the License, or
#(at your option) any later version.
#This program is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#GNU General Public License for more details.
#You should have received a copy of the GNU General Public License
#along with this program.  If not, see <http://www.gnu.org/licenses/>.
#-------------------------------------------------------------------------------
-----------------------
#K.Srinivasan
#NeuronRain Documentation and Licensing: http://neuronrain-
documentation.readthedocs.io/en/latest/
#Personal website(research): https://sites.google.com/site/kuja27/
#-------------------------------------------------------------------------------
-----------------------

Document Licensed by:
###############################################################################
#############################################################
<a rel="license" href="http://creativecommons.org/licenses/by-nc-nd/4.0/"><img
alt="Creative Commons Licence" style="border-width:0"
src="https://i.creativecommons.org/l/by-nc-nd/4.0/88x31.png" /></a><br />This
work is licensed under a <a rel="license"
href="http://creativecommons.org/licenses/by-nc-nd/4.0/">Creative Commons
Attribution-NonCommercial-NoDerivatives 4.0 International License</a>.
###############################################################################
#############################################################

-------------------------------------------------------------------------------
--------------------------
NeuronRain - Design Objectives
-------------------------------------------------------------------------------
--------------------------
NeuronRain OS is machine learning, cloud primitives enriched linux-kernel fork-
off with applications in Internet-of-Things. AstroInfer is
the machine learning side of it to learn analytics variables later read as
config by linux kernel and any device specific driver.
NeuronRain has been partitioned into components so that effective decoupling is
achieved e.g Linux kernel can be realtime OS for realtime less-latency
applications, Message Queuing can be KingCobra or any other standard MQ product.
There are IOT operating systems like https://github.com/RIOT-OS/. Linux and IoT
- linux on cameras,automobiles etc., - https://lwn.net/Articles/596754/.
NeuronRain in itself is not just a product but a framework to build products.
Applications can invoke VIRGO cloud memory,clone,filesystem primitives to create
new products on NeuronRain Smart-IoT framework.


Copyright attributions for other open source products dependencies:
-------------------------------------------------------------------
1.Maitreya's Dreams - http://www.saravali.de (some bugs were fixed locally in
degree computation of textual display mode)
2.SVMLight -  http://svmlight.joachims.org/
3.BioPython and ClustalOmega Multiple Sequence Alignment BioInformatics Tools
(www.biopython.org, www.ebi.ac.uk/Tools/msa/clustalo/ )

--------------------------
Open Source Design and Academic Research Notes have been uploaded to
http://sourceforge.net/projects/acadpdrafts/files/MiscellaneousOpenSourceDesignA
```

ndAcademicResearchNotes_2013-08-11.pdf/download

Presently a complete string sequence mining subsystem and classification algorithms with indexing have been implemented for mining patterns in rules and encoded strings and executing those rules (in special case,an encoded horoscope).
****
AstroInfer Classifiers - Documentation on the dataset files used
****

decisiontree-attrvalues.txt - Attributes based on which decision is done and the values they can take (comma separated list)
decisiontree-test.txt - Test dataset with set of values for attributes in each line
decisiontree-training.txt - Training dataset with set of values for attributes and class it belongs in each line
test-set.txt - List of article id(s) to be classified - NaiveBayes
topics.txt - List of classes the article id(s) in training set belong to - NaiveBayes
training-set.txt - List of articles id(s) already classified - training dataset for NaiveBayes
word-frequency.txt - Words and their frequencies of occurrence in all articles - NaiveBayes
words.txt - words, the article id(s) having those words and number of occurrences of those words within specific article id (~ separated)

Above XXX.txt files need to be populated after doing some preprocessing of the articles to be classified. Preprocessing might include finding the word frequencies in the articles, finding classes of the articles, finding attributes and possible values of those attributes. At present the asfer.enchoros file contains encoded horoscopes for single class of events. Thus classification is redundant. But if it has encoded horo strings for all events then to filter out strings of a particular class of events, classification is needed.

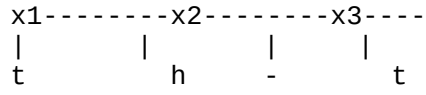
Python script for autogenerating above txt files has been added under python-src/autogen_classifier_dataset.This script needs to be changed forgenerating training and test set files.

****
DESIGN NOTES, THEORETICAL INTERLUDES(might have errors), TODO AND NICE TO HAVE FEATURES (list is quite dynamic and might be rewritten depending on feasibility - long-term design goals with no deadline)
****

(FEATURE - DONE-MDL,Entropy,Edit Distance,Compressed Sensing) 1. Test with Massive Data Sets of encoded strings for pattern mining. Algorithms for Approximate Kolmogorov Complexity or Minimum Description Length (MDL), Shannon Entropy, Levenshtein Edit Distance have been implemented in Python and C++. Compressed Sensing (used in Image and Signal Processing) which "sense" from "compressed" large data - This involves computing a matrix product AX=B where X is an image or bitmap and A is a chosen matrix which together give a sketch B. Sketch of an image bitmap has been implemented.

(FEATURE - THEORY-POC Implementation-DONE) 2. An experimental text compression algorithm that deletes vowels and stores only consonants as far as meaning is not altered. For example "follow" could be stored as "fllw". Since on an average every third letter in an english word is a vowel, approximate compression is 33%. Message is reconstructed using most probable meaningful word for "fllw" (For example "follow" could be more probable than "fellow" or vice versa). This

is similar to Texting in phones and to some extent encoding in Match Rating (http://en.wikipedia.org/wiki/Match_rating_approach). One more example could be "Decision" which can be compressed as "Dcsn". An interesting phonetic aspect of this is that "Decision" is spelt as "Dee-C-shan", or each vowel following a consonant is subsumed or coalesced into the preceding consonant while spelling. Thus "Dcsn" gives 50% compression ratio. PyEnchant python package SpellChecker has suggest() function that returns a tuple of closely related words to a compressed (or "misspelt") word. Conventional WSD algorithms might have to be used on this tuple to get the maximum match. Initial testing reveals that the accuracy with spellcheckers is less and this problem requires a non-trivial algorithm which might require error-correcting codes like Reed-Solomon and Berelekamp-Massey assuming the english text as a finite field of alphabets as elements. Predominantly used dictionary-based Lesk's disambiguation Algorithm can find the intersection between "follow" and the rest of the context and "fellow" and rest of the context and choose the word with maximum intersection with context - this is quite commonsensical too. But the drawback of this disambiguation is that keywords are disambiguated well while other connectives (is, are, thus, this, who, why etc.,) in the sentences are not. Another limitation of applying WSD while decompressing is lack of meaningful context words at runtime - only compressed words are available and they cannot be used as disambiguating context creating a circular dependency - disambiguation requires decompression and decompression needs disambiguation. For computing maximum likelihood, most probable vowel between 2 consonants can be zeroed in on by a 26*26 table of consonant ordered pairs and having vowels between them as probability distribution priors. This is feasible only if priors are available. For example th-t has (h,t) as consonant pairs and a is most probable vowel than e,i,o and u and th-t is disambiguated as "that". The vowels missing in compressed text can be represented by a single extra bit. This is an alternative to PyEnchant spellcheck suggest() function. This can be Hidden Markov Model also - with missing vowels as hidden states to measure and compressed letters as observations. This requires forward-backward probabilities computation or Viterbi path which gives most likely word for a compressed word. For "th-t" the Markov Model looks like:



and Bayesian for the above is:

Pr(x3/[t,h,-,t]) is directly proportional to Pr(x3/[t,h]) * Pr([t]/x3)

and to be precise it is:

Pr(x3/[t,h,-,t] = Pr(x3/[t,h]) * Pr([t]/x3) / Pr(t)/Pr(t,h) with denominator being a pre-computable constant from substring hashtable as below.

Viterbi path would give the most likely path or most likely word for above.

(2.1) Pr(x3/[t,h]) is computed from dictionary - number of words having the sequence / total number of words
which is the argmax(Pr(a/t,h), Pr(e/t,h), Pr(i/t,h), Pr(o/t,h), Pr(u/t,h)) and priors are computed for the substrings tha, the, thi, tho, thu.

(2.2) Pr(t/x3) is computed from dictionary similar to the previous for argmax of probabilities for substrings at, et, it, ot, ut.

(2.3) If total number words in dictionary is E (could be 200000 to 300000) and yi is the number of words of length i, then y1 + y2 + ... + yn = E .

(2.4) Number of substrings of an n bit word is (n-1)n/2.

(2.5) Thus number of substrings for all words in dictionary is y2 + 3*y3 + 6*y4 + 10*y5 + ... + (n-1)n*yn/2. Thus number of substrings (could be significantly more than 500000) is independent of the text to be decompressed and the substring prior probabilties can be precomputed and stored in a hash table (size of the table is = number of substrings * 5 for each vowel). This table has to be huge but does not depend on text to be decompressed.

(2.6) Above is theoretical basis only implementation of which needs precomputing the above hashtable.

(2.7) Above experimental compression scheme that ignores vowels gives a string complexity measure (a minimum description length) similar to Kolmogorov complexity.

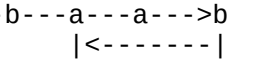
(2.8) Algebraically, above can be imagined as a Group Action where group G is set of consonants (with the assumption that inverses exist for each consonant with a special "backspace" element added to alphabet set and identity is "space" element) and set X to act on is the set of vowels. Orbit is the set of X elements moved (g.x) and if concatenation (with phonetic coalition) is a group operator then consonants act on vowel sets. For example, the consonant "t" acts on "o" from vowel set to give the word "to". The concatenation is non-abelian. Interestingly the above phonetic coalition of vowels to an adjoining consonant is the Stabilizer set or set of Fixed points(g.x=g). For example, the consonant "d" acting on the vowel "e" gives a phonetically coalesced compression "d" ("d" ,"de" and "dee" are phonetically same as "d") as a fixedpoint. Thus Orbit-Stabilizer Theorem (set of cosets of Fixed points - Quotient group G/G(x) - isomorphic to Orbit) should apply to the above. There may be scenarios where a maximum likely vowel from above argmax() computation does not equal the actual vowel expected and for arbitrary non-dictionary strings prior computation is difficult. Thus this algorithm is quite experimental.

(2.9) Linguistic words can be construed as a polynomial graph plotted on an x-y axis: x-axis represents position of a letter in a text and y-axis represents an alphabet (vowels and consonants). From the previous string complexity measure missing vowels are missing points on the polynomial. Disambiguating the word is then equivalent to Polynomial Reconstruction Problem/List Decoding/Error correcting codes (Reference: Point 345) that reconstructs a list of polynomials (i.e set of disambiguated words) from a codeword(i.e compressed vowelless string) one of which disambiguates and decompresses the vowelless codeword - polynomial reconstructed should include (almost all) missing vowels. Text can be represented as a polynomial function defined as f:N->Alphabets where N is set of natural numbers. Factoring over this polynomial ring enables text to be represented as products of texts. An implementation of polynomial encoding of texts has been done in 364. Berlekamp-Welch algorithm creates a system of equations which themselves are evaluations of two polynomials over finite fields for all xi (P(xi) = Q(xi)/E(xi)) where each xi is from input set of ordered pair of points (xi,yi). This system of equations is solved by Gaussian Elimination to compute Q(x)/E(x) = P(x). Berlekamp-Welch encodes a text in a different way than plain polynomial curve fitting - each letter ordinal in text is a coefficient of a polynomial of degree equal to size of text whereas polynomial encoding of text fits a curve passing through the ordinal values of letters in successive positions. This polynomial with ordinal coefficients over a finite field is then evaluated in n points and transmitted. Recipient reconstructs the original polynomial from transmission errors in received points by Gaussian Elimination. If transmission error is simulated as removing vowels in original message polynomials, reconstruction by solving system of equations should almost correctly identify missing vowels.

(2.10) From point 345 and 346, high level language texts like English,French etc., can be thought of as 255-coloring of letter positions where 255 is the size of alphanumeric alphabet (ASCII,Unicode). Each letter in a text "colors" corresponding letter position in text with an alphabet. From Van Der Waerden and other coloring theorems, there are monochromatic arithmetic progressions in letter positions in high level language texts i.e alphabets in APj positions are all same for some APj. This implies there is an order in high level context sensitive natural language grammars and answers in affirmative the conjecture posited in 346.

(2.11) Monochromatic arithmetic progressions in letter positions of natural language texts implies that finite state automata can be learnt to accept such arithmetic progressions in text. For example , text "abaabaabaab..." over

alphabets {a,b} has alphabet b in AP positions where AP is 2+3n. Regular language Finite State Automata for this 2-colored infinite string is:



(2.12) In terms of vowelless string complexity measure previously, if a 255-colored string is simplified to 2-colored string as "Consonant-Vowel-Consonant-Consonant-Vowel-Consonant-Consonant-Vowel..." replacing consonant and vowels with "Consonant" and "Vowel" notations, english language (and most others) on an average has an arithmetic progression k+3n of vowel positions for some constant k.

(2.13) Previous vowelless string compression can be linguistically explained in terms of sequence of syllables. Related syllable based name clustering of proper nouns occurring in BigData sets has been described in 708 - Strings of Names in english suffer from lack of compound alphabets which most other classical languages are bestowed with. Every syllable in an English name string (when it is spelt out verbally) could contain more than one alphabet which is concatenation of vowels and consonants while in most other natural languages every syllable is mostly associated with just one compound alphabet(vowel + consonant). This makes clustering similar names easier in other languages by comparing syllables (blocks of substrings) instead of alphabets. In previous example word "decision" has blocks of substrings or syllables - "de-ci-sion". Thus distance between two strings could be defined as distance between audio of the words (by recording the syllables of the two words by human speech and comparing the two audios by speech merit measures like MFCC). Text Compression then amounts to succinctly storing the sequences of syllables - an example succinct storage is to transcribe the string from English to another language having compound alphabets and storing the concise syllables in another language. Example string "de-ci-sion" is compressed syllable-wise as "d-c-sn" forfeiting vowels in English while a transcription could benefit from compound alphabets. From 2.8, algebraically each syllable is an Orbit and could be a Stabilizer set too (g.x=g) - e.g d.e=d, c.i=c in previous example string "decision".

3. (FEATURE - THEORY - Pairwise LCS implementation DONE) KMeans, KNN and other clustering and classification algorithms implemented thus far, group similar encoded strings for astronomical (or any other) data. Clustered strings can further be mined for patterns with ordering by applying a Longest Common Substring algorithm within each cluster. This gives a fine grained pattern hidden in the encoded input strings. These longest common substrings can then be translated to a rule similar to class association rule (Frequent item set and GSP algorithms are quite expensive and do not fit well for mining patterns in strings).

4. (FEATURE - DONE) Implementation of String Distance measure using different distance measures (python-src/StringMatch.py) for comparing Encoded Strings in addition to pairwise and Multiple Sequence Alignment.

5. (FEATURE - DONE) Construct a Hidden Markov Model with State transitions and Observations(events). If the number of states are exponential then this will be infeasible. A HMM has been implemented for text decompression with vowel-removed strings. Also a HMM has been implemented for Part-of-Speech tagging, Named Entity Recognition and Conditional Random Fields.

6. (FEATURE - DONE - continuation of point 3) Correlate with Rules from Classics (BPHS, BJ etc.,) with the mined datasets for particular class of events (Special Case of Mundane Predictive Model is described in items 47 to 51 and basic framework is already implemented). The Longest Common Substring implementation already mines for most common pattern with in a clustered or classified dataset of encoded strings which suffices for astronomical datasets. Script for parsing the clustered data from classifiers and decoding the longest common pattern have been added.

7. (FEATURE - DONE) Integrate Classifiers in above String Pattern mining

(classify strings and then mine).

8. (FEATURE - DONE) At present fundamental algorithms like - Perceptrons with Gradient Descent, Linear and Logistic Regression, Hidden Markov Model for Text Decompression, KMeans Clustering, Naive Bayesian, Decision Tree, SVM (uses thirdparty OSS) and kNN Classifiers, String Alignment and Distance based algorithms, Knuth-Morris-Pratt String Match Algorithm, Sequence Mining, Deep Learning(Convolution and Backpropagation), Social Network Analysis, Sentiment Analysis - have been implemented specialized for the encoded strings astronomical datasets.

9. (FEATURE - DONE) InterviewAlgorithm code in https://sourceforge.net/p/asfer/code/146/tree/python-src/InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit.py and Classification based on indegrees of wordnet subgraph node in action item 6 above (as mentioned in http://arxiv.org/abs/1006.4458 , https://sites.google.com/site/kuja27/TAC2010papersubmission.pdf?attredirects=0 and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf). Related Publication Drafts are: https://sites.google.com/site/kuja27/LowerBoundsForMajorityVotingPseudorandomChoice.pdf?attredirects=0, https://sites.google.com/site/kuja27/CircuitForComputingErrorProbabilityOfMajorityVoting.pdf?attredirects=0, https://sites.google.com/site/kuja27/IndepthAnalysisOfVariantOfMajorityVotingwithZFAOC.pdf?attredirects=0. Test C++ code written in 2006 for computing Majority voting error probability is at: http://sourceforge.net/p/asfer/code/HEAD/tree/cpp-src/miscellaneous/pgood.cpp and python script written in 2010 is at: http://sourceforge.net/p/asfer/code/HEAD/tree/python-src/pgood.py.

[Disclaimer and Caveat: Due to radical conclusions that are derivable from the P(good) binomial coefficients summation infinite series when applied to majority voting with SAT setting in perfect zero-error case, all the related points elsewhere in this document on majority voting + SAT versus psuedorandom choice (e.g 100% noise stability and circuit lowerbounds for it) are work-in-progress , unreviewed (excluding those done during 2009-2011) , not-quite rigorous drafts only with possible theoretical errors, which might have commissions-omissions and more additions to make as mentioned earlier and this analyzes a very special case of voting scenario. It is not an attempt to prove or disprove P=NP,but rather an analysis of very generic unnatural proof framework based on boolean social choice complexity for proving lowerbounds which might include the question of P=NP. Infact it shrouds entire complexity classes underneath it. It is non-conventional and questionable as to whether LHS pseudorandomness or dictator boolean function can be equated to RHS Majority Voting. This just analyzes the special case when there is zero-error or same-error on both sides and the outcomes of processeses related to pseudorandom/dictator choice and Majority Voting with Voter SAT oracles are "equally good" where "Goodness" is defined as for all scenarios which could be infinite, a PRG/Dictator choice or Majority Voted Choice "make zero-error decisions". It is not a statistical mean on both sides in which all probabilities are averaged. More specifically, (in)tractability of perfect voting is pivotal to this - assuming perfection which does away with BP* complexity classes this poses itself as a strong non-trivial counterexample. In theoretical interludes in this document probability of perfection is defined as a function of NoiseSensitivity or NoiseStability which is based on commonsense notion that if an entity (human or software) has "correct thinking in decision making" outcome "decision" is "correct". For example if noisy inputs do not perturb a voter's boolean function, voter is 100% stable. This is equivalent to real life scenarios where conflicting inputs from people make a person to decide incorrectly. A perfect voter is never swayed. Even without equating the 2 sides(e.g whether there exists a P algorithm for RHS PH or EXP circuit) above are non-trivial problems. Lot of assumptions have been made in arriving at conclusions in this document, most important being equal stability implies circuit lowerbounds which is based on a matrix of scenarios

where error and noise intersect and 100% noise stability regime in percolation crossing events. In general setting, distribution is Poisson binomial with unequal voter error and Error is not just limited to noise sensitivity but also includes flawed decision tree evaluation which adds one more scenario to matrix. It is more apt to say there are no conclusions arrived at from equating LHS and RHS of Pr(Good) circuits. Rather it raises lot of more open questions and contradictions. A major requirement for proving a separation or containment is that RHS majority voting circuit has to be a complete problem for class C whether exact or probabilistic(BP*), implying whole L(RHS) is in L(LHS) because LHS is an algorithm for a complete problem. For many probabilistic BP* and PH classes no complete problems are known. Conclusions can be drawn subject to knowledge of complete problems for RHS majority voting circuit.

There are zero error voting systems:
E.g Paxos protocol without byzantine failures -
http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf
]

Psuedorandom Choice and Majority Voting (Majority circuit with SAT inputs)

---

10. (THEORY)
https://sites.google.com/site/kuja27/ImplicationRandomGraphConvexHullsAndPerfectVoterProblem_2014-01-11.pdf?attredirects=0&d=1 on decidability of existence of perfect voter and the probability series for a good choice of https://sites.google.com/site/kuja27/CircuitForComputingErrorProbabilityOfMajorityVoting_2014.pdf?attredirects=0&d=1 are related to already well studied problems in social choice theory but problem definition is completely different. Arrow's theorem of social choice for an irrational outcome in condorcet election of more than 2 candidates and its complexity theory fourier analysis proof [GilKalai] are described in www.cs.cmu.edu/~odonnell/papers/analysis-survey.pdf. Irrational outcome is a paradox where the society is "confused" or "ranks circularly" in choice of a candidate in multipartisan condorcet voting. Rational outcome converges to 91.2% (Guilbaud number) with a possibility of 8.8% irrational outcome.

Additional References:

10.1. Social Choice Theory, Arrow's Theorem and Boolean functions -
http://www.ma.huji.ac.il/~kalai/CHAOS.pdf
10.2. http://www.project-syndicate.org/commentary/kenneth-arrow-impossibility-theorem-social-welfare-by-amartya-sen-2014-11
10.3. Real life illustration of Arrow's Theorem -
http://www.nytimes.com/2016/05/09/upshot/unusual-flavor-of-gop-primary-illustrates-a-famous-paradox.html - Condorcet Circular choice paradox, Change in Voter decision when a new Candidate is added and Conflict between individual decision and group decision - Incompleteness of democratic process.
10.4. How hard is it to control elections [BartholdiToveyTrick] -
http://www.sciencedirect.com/science/article/pii/089571779290085Y - manipulating outcomes of variety of voting schemes is NP-hard.
10.5. How hard is it to control elections with tiebreaker [MatteiNarodytskaWalsh] - https://arxiv.org/pdf/1304.6174.pdf

11.(THEORY) What is perplexing is the fact that this seems to contravene guarantee of unique restricted partitions described based on money changing problem and lattices in https://sites.google.com/site/kuja27/SchurTheoremMCPAndDistinctPartitions_2014-04-17.pdf?attredirects=0 and https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions_2014.pdf?attredirects=0&d=1 which are also for elections in multipartisan setting (if condorcet election is done). Probably a "rational outcome" is different from "good choice" where rationality implies without any paradoxes in ranking alone without giving too much weightage to the "goodness" of a choice by the elector. Actual real-life elections are not condorcet elections where NAE tuples are generated. It is not a conflict between Arrow's theorem as finding atleast 1

denumerant in multipartisan voting partition is NP-complete (as it looks) - which can be proved by ILP as in point 20 of https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions_2014.pdf?attredirects=0&d=1 - the assumption is that candidates are not ranked; they are voted independently in a secret ballot by electors and they can be more than 3. The elector just chooses a candidate and votes without giving any ordered ranking for the candidates which makes it non-condorcet.

12.(THEORY) Moreover Arrow's theorem for 3 candidate condorcet election implies a non-zero probability of error in voting which by itself prohibits a perfect voting system. If generalized to any election and any number of candidates it could be an evidence in favour of P != NP by proving that perfect voter does not exist and using democracy Maj+SAT circuits and P(Good) probability series convergence (As described in handwritten notes and drafts http://sourceforge.net/projects/acadpdrafts/files/ImplicationGraphsPGoodEquationAndPNotEqualToNPQuestion_excerpts.pdf/download, https://sites.google.com/site/kuja27/ImplicationRandomGraphConvexHullsAndPerfectVoterProblem_2014-01-11.pdf?attredirects=0&d=1, https://sites.google.com/site/kuja27/LowerBoundsForMajorityVotingPseudorandomChoice.pdf?attredirects=0, https://sites.google.com/site/kuja27/CircuitForComputingErrorProbabilityOfMajorityVoting_2014.pdf?attredirects=0&d=1 and https://sites.google.com/site/kuja27/PhilosophicalAnalysisOfDemocracyCircuitAndPRGChoice_2014-03-26.pdf?attredirects=0).

13.(THEORY) But it is not known that if Arrow's theorem can be generalized for infinite number of candidates as above and whether such an electoral system is decidable. The possibility of a circular ranking in 3-condorcet election implies that there are some scenarios where voter can err though not exactly an error in making a "good choice" (or Perfect Voter Problem is decidable in case of 3 candidates condorcet election).

14.(THEORY) Each Voter has a k-SAT circuit which is input to Majority circuit. k-SAT can be reduced to 3-SAT (not 2-SAT) and thus is NP-complete. Error by a Voter SAT circuit implies that voter votes 0 instead of 1 and 1 instead of 0. This is nothing but the sensitivity of the voter boolean function i.e number of erroneous variable assignments by the voter that change the per-voter decision input to the Majority circuit. Thus more the sensitivity or number of bits to be flipped to change the voter decision, less the probability of error by the voter. If sensitivity is denoted by s, 1/q is probability that a single bit is flipped and probability of error by the voter is p then p = 1/q^s which is derived by the conditional probability that Pr[m bits are flipped] = Pr[m-th bit is flipped/(m-1) bits already flipped]*Pr[(m-1) bits are flipped] = 1/q*1/q^(m-1) = 1/q^m (and m=s) . This expression for p can be substituted in the Probability series defined in https://sites.google.com/site/kuja27/CircuitForComputingErrorProbabilityOfMajorityVoting_2014.pdf?attredirects=0&d=1. Probability of single bit flip is 1/q if the number of variables across all clauses is q and each variable is flipped independent of the other. Error by voter can also be simulated with probabilisitc truth tables(or) probabilistic CNFs.

For example, a term in the binomial summation becomes: mC(n+1) * (1/q^s)^(n+1) * (1-1/q^s)^(m-n-1) where m is total number of voter SATs and (1/q) is probability of single variable flip in Voter SAT and s is sensitivity of Voter SAT boolean function - s can change for each voter SAT in which case it has to be s1,s2,s3,s4,s5,... and the summation requires hypergeometric algorithms.

The Voter SAT circuit has an interesting application of Hastad's Switching Lemma - random probabilistic setting of variables known as "restrictions" decreases the decision tree depth significantly , or, number of variables that determine the Voter SAT outcome reduces significantly - in other words, random "restricted convincing" of a voter SAT CNF has huge impact on number of variables yet to be "convinced".

As done in the Switching Lemma proof, parity circuit can be an AND of ORs (or OR of ANDs) which is randomly restricted to switch the connectives in lowest 2 levels to arrive at super-polynomial lowerbound for parity and hence for each voter SAT. Thus the Parity CNF can be the special case of Voter SAT also (i.e voter wants odd number of variables to be satisfied - odd voter indeed). This kind of extends switching lemma to an infinite majority with parity oracle case.

Fourier expansion of the unbounded fan-in Majority+SAT circuit is obtained from indicator polynomial (http://www.contrib.andrew.cmu.edu/~ryanod/?p=207) by substituting fourier polynomial for each voter SAT in the majority circuit fourier polynomial. Conjecturally, this expansion might give an infinite summation of multilinear monomials - has some similarities to permanent computation if represented as an infinite matrix i.e Permanent converges to 1 for this infinite matrix for perfect voting (?) and Permanent is #P complete. This is obvious corollary of Toda's theorem in a special case when RHS of P(Good) is a PH=DC circuit because Toda's theorem implies that any PH problem is contained in P^#P(P with access to #P number_of_sats oracle). Permanent computes bipartite matchings - non-overlapping edges between two disjoint vertex sets - probably pointing to the bipartisan majority voting. This makes sense if the entire electorate is viewed as a complete social network graph and voting is performed on this network electorate. Majority voting divides the social network into two disjoint sets (for and against) with edges among them - division is by maximum matching - "for" voter vertex matched with "against" voter vertex. When this matching is perfect, election is tied. Imperfect matchings result in a clear winner vertex set - which has atleast one unmatched vertex is the winner.

Demarcation of linear and exponential sized ciruits: P(good) RHS has Circuit SAT for each voter with unbounded fanin.
1) If the variables for each voter SAT are same, then the circuit size is exponential in number of variables - DC-uniform
2) else if the variables are different for each SAT, then the circuit is linear in number of variables - polynomial size

1) is more likely than 2) if lot of variables among voters are common.

There are three possibilities
Circuit lies between 1) and 2) - some variables are common across voters - kind of super-polynomial and sub-exponential
Circuit is 1) - all variables are common across voters - the worst case exponential (or even ackermann function?)
Circuit is 2) - there are no common variables

As an alternative formulation, LHS of the P(Good) series can be a dictator boolean function (property testing algorithms like BLR, NAE tuples - http://www.contrib.andrew.cmu.edu/~ryanod/?p=1153) that always depends on only one variable and not on a Pseudorandom generator. Thus how error-free is the assignment to LHS dictator boolean function determines P(Good) LHS(sensitivity and probabilistic truth tables are the measures applicable as described previously). Thus both LHS and RHS can be boolean functions with corresponding circuits constructible for them.

Influence of a variable i in a boolean function is Probability[f(x) != f(x with i-th bit flipped)]. For Majority function influence is ~ 1/sqrt(n) from Berry-Esseen Central Limit Theorem - sum of probability distributions of values assigned to boolean random variables converge to Gaussian. Thus in infinite majority a voter is insignificant. Error in majority voting can be better defined with Stability and Noise Sensitivity measures of a Boolean Function (Influence, Stability, Noise, Majority is Stablest theorem etc., - www.cs.cmu.edu/~odonnell/papers/analysis-survey.pdf) where correlated pairs of boolean variable tuples form an inner product to get the expression : Noise(f) = 1/2 - 1/2*Stability(f) where Noise is the measure of corruption in votes polled while Stability is the resilience of votes polled to corruption so that outcome is not altered. [But for infinite majority inner product could become infinite

and the measures may lose relevance]. Also related is the Kahn-Kalai-Linial Theorem for corrupted voting - http://www.contrib.andrew.cmu.edu/~ryanod/?p=1484 - that derives a lower bound for the maximum influence (log(n)/n). Applications of Noise Sensitivity in real-world elections is described in - https://gilkalai.wordpress.com/2009/03/06/noise-sensitivity-lecture-and-tales/ , http://www.cs.cmu.edu/~odonnell/slides/mist.pps and percolation in http://research.microsoft.com/en-us/um/people/schramm/memorial/spectrumTalk.pdf?tduid=(332400bcf8f14700b3a2167cd09a7aa9)(256380)(2459594)(TnL5HPStwNw-SliT5K_PKao3NElOXgvFug)()

As done for sensitivity previously, a term in the P(Good) binomial summation becomes:
mC(n+1) * product_of_n+1_(NoiseSensitivityOfVoterSAT(i)) * product_of_m-n-1_(1-NoiseSensitivityOfVoterSAT(i)) where m is total number of voter SATs and the summation requires hypergeometric algorithms. Infact it makes sense to have Stability as a zero-error measure i.e the binomial summation term can be written as mC(n+1) * product_of_n+1_(NoiseSensitivityOfVoterSAT(i)) * product_of_m-n-1_(1/2+1/2*StabilityOfVoterSAT(i)) since 1-NoiseSensitivity = 1/2 + 1/2*Stability. Difference between sensitivity and NoiseSensitivity here is that NoiseSensitivity and Stability are probabilities and not number of bits and are directly substitutable in P(Good) series.

Summation of binomial coefficients in RHS is the collective stability or noise in the infinite majority circuit. It converges to 0.5 if NoiseStability is uniform 0.5 for all Voter SATs and in best case converges to 1. This summation connects analysis, non-uniform infinite majority voting and complexity theory - Hypergeometric functions, Noise and Stability of Voter Boolean functions, Majority Circuit.

P(Good) summation is Complementary Cumulative Binomial Distribution Function (or) Survival Function

---

Goodness Probability of an elected choice is derived in https://sites.google.com/site/kuja27/CircuitForComputingErrorProbabilityOfMajorityVoting_2014.pdf . For an above average good choice (i.e a choice which is more than 50% good) atleast majority populace must have made a good decision (this is axiomatically assumed without proof where all voters have equal weightages). In probability notation this is Pr(X > n/2) where X is the binomial random variable denoting the number of electorate who have made a good choice which has to be atleast halfway mark. For each Pr(X=k) the binomial distribution mass function is nCk(p)^k(1-p)^k [Reference: https://en.wikipedia.org/wiki/Binomial_distribution]. Cumulative distributive function is defined as Pr(X <= k) = summation_0_k(nCk(p)^k(1-p)^(n-k)) and its complement is Pr(X > k) = 1 - Pr(X <= k) mentioned as Complementary Cumulative Distributive Function [Reference: https://en.wikipedia.org/wiki/Cumulative_distribution_function] which is also known as Survival Function[Reference: https://en.wikipedia.org/wiki/Survival_function].

A plausible proof for the assumption that for an above average good choice atleast half of the popluation must have made a good decision:
If there are n inputs to the Majority circuits, the voter inputs with good decision can be termed as "Noiseless" and voter inputs with bad decisions can be termed as "Noisy" in Noise Sensitivity parlance. For infinite majority it is known that(from http://analysisofbooleanfunctions.org/):

lim NS(delta, Majority(n)) = 1/pi * arccos(1-2*delta) (due to Central Limit Theorem and Sheppard's Formula)
n->infinity
n odd

where delta is probability of noise and hence bad decision. If delta is assumed to be 0.5 for each voter (each voter is likely to err with probability 0.5) then above limit becomes:

1/pi * arccos(1-1) = 1/pi * pi/2 = 1/2 which is the Noise (Goodness or Badness) of the majority circuit as a whole.

From this expected number of bad decision voters can be deduced as E(x) = x*p(x) = 0.5 * n = n/2 (halfway mark)

Above is infact a simpler proof for P(Good) without any binomial summation series for delta=0.5. In other words the number of bad decisions are reverse engineered. But P(Good) binomial summation (complementary cumulative distributive function) is very much required in the hardest problem where each voter has a judging function to produce an input to Majority function and each such judging function is of varied complexities and noise stabilities.

For delta=0.5, NoiseSensitivity of infinite majority=0.5 and both are equal. For other values of delta this is not the case. For example interesting anomalies below surface:

delta=0.3333...:

NS = 1/pi * arccos(1 - 2*0.333...) = 0.3918

delta=0.25:

NS = 1/pi * arccos(1-2*0.25) = 0.333...

Chernoff upper tail bound

Upper bound for P(Good) binomial distribution can be derived from Chernoff bound (assuming Central Limit Theorem applies to sum of Binomial random variables)- http://www.cs.berkeley.edu/~jfc/cs174/lecs/lec10/lec10.pdf - For P(Good) series the upper tail bound is needed i.e. Pr(Number of voters making good decision > n/2) which is derivable to 0.5 and 1 exactly wherein the upper bound is not necessary. But for other cases , upper tail Chernoff bound is:

P[X > (1+delta)*mean] < {e^delta/(1+delta)^(1+delta)}^mean

Total population is n, mean=n*p and (1+delta)*mean = n/2

Thus delta = (1 - 2*p)/2*p

P[X > n/2] < {e^[(1-2p)/p]/(1/2p)^(1/2p)} = {e^(1-2p)n * (2p)^(1/2p)}

When p=0.5 , P[X > n/2] < 1 which is an upperbound for 0.5.

But intriguingly for p=1 the upper tail bound is:

(e^(-1)*sqrt(2))^n

which is increasingly less than 1 for high values of n whereas P(good) converges to 1 in exact case (why are these different?). This contradiction arises since in exact derivation of P[X > n]:

mC(n+1)(p)^(n+1)(1-p)^(m-n-1) + mC(n+2)(p)^(n+2)(1-p)^(m-n-2) + ... + mCm (p)^m (1-p)^(m-m)

all the terms vanish by zero multiplication except mCm (p)^m (1-p)^m which becomes:

mCm*1*0^0

and by convention, 0^0 = 1, thereby converging to 1 while the Chernoff upper tail bound decreases with n.

Hoeffding upper tail bound

For Bernoulli trials involving binomial distribution summation, Hoeffding Inequality can be applied for upper tail bound - https://en.wikipedia.org/wiki/Hoeffding%27s_inequality - defined by:

P(X > (p+epsilon)n) <= exp(-2*epsilon^2 * n)

If p+epsilon = 0.5 for halfway mark, epsilon = 1/2 -p.

P(X > n/2) <= exp(-2(0.5-p)^2*n)
For p=1, P(X > n/2) <= exp(-n/2) which for infinite n converges to 1.

Thus tailbounds above are only less tight estimates and exact summation is better for p=0.5 and 1.

There are two independent aspects to the P(Good) series - Goodness and Hardness:

---

(1) Goodness of Voting:
The convergence of the binomial coefficient summation series in RHS is the "goodness" side of the voting expressed in terms of Noise sensitivity and Stability of each Voter's Boolean Function - whether the voters have exercised their franchise prudently to elect a "good" choice. If the Voter Boolean Functions are all balanced then by Kahn-Kalai-Linial lowerbound there are n/log(n) influential variables that control the decision of each Voter.
(2) Hardness of Voting:
The circuit for RHS(e.g Boolean Function Composition) and its Fourier polynomial expansion.

Mix of Hardness and Goodness result in variety of BP* classes or P* classes (if error is unbounded). Goodness is related to Noise Sensitivity and Stability. Probabilistic Hardness is due to pseudorandomness. Are Goodness and Pseudorandomness related? Yes, because Noise Sensitivity occurs when there is a correlation between two strings with flips. Correlation inturn occurs when flips are pseudorandom. Connection between Stability, Fourier coefficients and pseudorandom flip probability are expressed in (16) and (20) below (Plancherel's theorem).

(*IMPORTANT*) Noise Stability, Noise Sensitivity and Complexity Class BPP (error in voting)

In all points in this document, an assumption is made that Noise Stability and BPP are equivalent notions. Is this valid? A language L is in BPP if for x in L there exists a Turing Machine that accepts x and outputs 1 with bounded probability (e.g 2/3) and if x not in L, rejects x and outputs 0 with bounded probability (e.g 1/3). Following table illustrates all possible scenarios and where BPP and Noise Stability fill-in (x/e is correlated, flipped version of x):

---

x	f(x) = f(x/e)	f(x) != f(x/e) Noise
x in L, x/e in L	No error	Error
x in L, x/e not in L	Error	No error if f(x)=1,f(x/e)=0
		else Error
x not in L, x/e in L	Error	No error if f(x)=0,f(x/e)=1
		else Error
x not in L, x/e not in L	No error	Error

Third column of the table relates Noise Sensitivity and BPP - it subdivides the Noise Sensitivity into 4 probable scenarios. Hence Noise Sensitivity overlaps BPP - explains one half of the error. Even after a circuit is denoisified i.e third column is removed, it can still be in BPP due to the second column possibilities above. Noise sensitivity handles only half the possibilities in the above table for output noise. The second and third rows in second column represent noise in input where the circuit doesn't distinguish inputs correctly (false positives and false negatives). This requires derandomization. Thus error-free decision making depends on both denoisification and derandomization. A Turing machine that computes all 8 possibilities in the table above without error is 100% perfect decision maker. In terms of probability:

Probability of Good Decision = Input noise stability (second column) + Output noise stability (third column)

which can also be stated as:

Probability of Good Decision = 1-(Probability of False positives + Probability of False negatives)

Above is a better ,rather ideal, error probability measure that can be substituted in P(Good) RHS binomial distribution summation because it accounts for all possible scenarios of errors. Presently, P(Good) binomial summation depends only on output noise sensitivity. Complexity literature doesn't yet appear to have the theoretical notion of input noise counterpart of output noise sensitivity for boolean functions(e.g what is the fourier theoretic estimation of false positivity similar to the Plancherel version of noise stability?). Second column input noise can be written as TotalError-[f(x) != f(x/e)] where TotalError<=1. Probability of Good Decision is sum of conditional probabilities of these 8 possibilities in the table:

Probability of Good Decision = Pr(f(x)=f(x/e) / x in L, x/e in L)*Pr(x in L, x/e in L) +
Pr(f(x)!=f(x/e) / x in L, x/e in L)*Pr(x in L, x/e in L) +
...
Pr(f(x)!=f(x/e) / x not in L, x/e not in L)*Pr(x not in L, x/e not in L)

Computation of the above conditional probability summation further confounds the estimation of voter decision error in Judge Boolean Functions.

In other words, a precise estimate of voter or voting error from table above which related BP* complexity class and Boolean Noise Sensitivity:

Probability of Bad Decision = NoiseSensitivity - {Pr(f(x)=1, f(x/e)=0)/x in L, x/e not in L}
- {Pr(f(x)=0, f(x/e)=1)/x not in L, x/e in L}
+ {Pr(f(x)=f(x/e))/x in L, x/e not in L}
+ {Pr(f(x)=f(x/e))/x not in L, x/e in L}

which may be more or less than NoiseSensitivity depending on cancellation of other conditional probability terms. This is the error term that has to be substituted for each voter decision making error. Because of this delta, NoiseSensitivity (and NoiseStability) mentioned everywhere in this document in P(Good) binomial summation and majority voting circuit is indeed NoiseSensitivity (and NoiseStability) (+ or -) delta. Previous probability coalesces Noise and Error into one expressible quantity - i.e denoisification + derandomization.

When Probability of Bad Decision is 0 across the board for all voters, the Pr(Good) series converges to 100% and previous identity can be equated to zero and NoiseSensity is derived as:

NoiseSensitivity = {Pr(f(x)=1, f(x/e)=0)/x in L, x/e not in L}
{Pr(f(x)=0, f(x/e)=1)/x not in L, x/e in L}
- {Pr(f(x)=f(x/e))/x in L, x/e not in L}
- {Pr(f(x)=f(x/e))/x not in L, x/e in L}

Above matrix of 8 scenarios and identity above give somewhat a counterintuitive and startling possibility that NoiseSensitivity of a voter's decision boolean function can be non-zero despite the convergence of Pr(Good) Majority Voting binomial series when some error terms do not cancel out in previous expression. This implies that there are fringe, rarest of rare cases where voting can be perfect despite flawed decision boolean functions.

Hastad Switching Lemma, P(Good) Majority Voting circuit and Election Approximation or Forecasting

---

Hastad Switching Lemma for circuit of width w and size s with probability of random restriction p for each variable states the inequality:

$$Pr[DecisionTree(f/restricted) > k] <= (constant*p*w)^k$$

where p is a function of width w and p is proportional to 1/w. In forecasting, a sample of voters and their boolean functions are necessary. If the Majority circuit is PH=DC(variables are common across voters) or AC(variables are not so common across voters), the sampling algorithm pseudorandomly chooses a subset from first layer of the circuit's fanin and applies random restriction to those Voters' Boolean Circuits which abides by above inequality. The rationale is that probability of obtaining required decision tree depth decreases exponentially (RHS mantissa < 1). Thus the forecast is a 3-phase process:
(1) Pseudorandomly choose subset of Voter Boolean Functions in the first layer of the circuit - This is also a random restriction
(2) Randomly Restrict variables in each boolean circuit to get decision trees of required depth if voters reveal their blackbox boolean functions
(3) Simulated Voting (and a property testing) on the restricted boolean functions subset accuracy of which is a conditional probability function of (1) and (2). This could be repetitive random restrictions of (2) also.

(1) and (2) are random restrictions happening in 2 different layers of the circuit - amongst voters and amongst variables for each voter .

Alternatively if subset of voters reveal their votes (1 or 0) in opinion polls then the above becomes a Learning Theory problem (e.g. PAC learning, Learning Juntas or oligarchy of variables that dominate) - how to learn the boolean function of the voters who reveal their votes in a secret ballot and extrapolate them to all voters. This is opposite of (1),(2),(3) and the most likely scenario as voters hold back their decision making algorithms usually. From Linial-Mansour-Nisan theorem , class of boolean functions of n variables with depth-d can be learnt in O(n^O(logn^d)) with 1/poly(n) error. From Hastad Switching Lemma, Parseval spectrum theorem (sum of squares of fourier coefficients) and LMN theorem , for a random restriction r, the fourier spectrum is e-concentrated upto degree 3k/r where k is the decision tree depth of the restriction. For a function g that approximates the above majority+SAT voting circuit f, the distance function measures the accuracy of approximation.

An interesting parallel to election approximation is to sample fourier spectrum of boolean functions of a complexity class C for each voter.(This seems to be an open problem - http://cstheory.stackexchange.com/questions/17431/the-complexity-of-sampling-approximately-the-fourier-transform-of-a-boolean-fu) . Essentially this implies complexity of approximating a voter's boolean function in complexity class C by sampling the fourier coefficients of its spectrum and hence election forecasting is C-Fourier-Sampling-Hard.

Points (53.1) and (53.2) define a Judge Boolean Function with TQBF example that is PSPACE-complete. Each voter in RHS of P(Good) has such a TQBF that is input to Majority circuit. TQBF is also AP-hard where AP is the set of all languages recognized by an alternating turing machine (ATM) with alternating forall and thereexists states, and it is known that PSPACE=AP

(http://people.seas.harvard.edu/~salil/cs221/spring10/lec6.pdf).

(*IMPORTANT*) Boolean Function Composition (BFC) and Majority+SAT voting circuit
---
Boolean Function Composition is described in detail at [AvishayTal] Section 2.3 - http://eccc.hpi-web.de/report/2012/163/ - which is defined as f*g = f(g(x)) and the sensitivity and complexity measures of BFC are upperbounded by products of corresponding measures. Majority Voting circuit for P(Good) Binomial summation can be formulated as composition of two functions - Majority and SAT i.e Voters' SAT outputs are input to Majority (or) Majority*SAT = Majority(SAT1, SAT2,...,SATn). Hence sensitivity of this composition is upperbounded by sensitivity(MAJn)*max(sensitivity(SATi)) which becomes the "sensitivity of complete electorate" . This has a direct bearing on the error probability of voter decision in P(Good) binomial summation - the psuedorandom choice error probability or dictator boolean function sensitivity in LHS and electorate sensitivity obtained from previous composition in RHS should be equal in which scenario LHS achieves what RHS does (or) there is a PH or EXP collapse to P. Circumeventing this collapse implies that these two sensitivity measures cannot be equal and either Dictator/PRG choice or Majority Voting is better over the other - a crucial deduction. Dictator boolean function has a sensitivity measure n (all variables need to be flipped in worst case as function depends on only one variable) while the Majority*SAT composition is a maximum of products of 2 sensitivities. Counterintuitively, RHS electorate sensitivity can be larger than LHS as there is no SAT composition in Dictator boolean function. But the convergence of P(Good) binomial coefficient series to 1 when p=1 does not rule out the possibility of these two sensitivity measures being equal and when it happens the Stability is 1 and NoiseSensitivity is 0 on both sides (assumption: NoiseSensitivity for LHS and most importantly RHS are computable and have same product closure properties under composition like other sensitivity measures).

[
Quoting Corollary 6.2 in http://eccc.hpi-web.de/report/2012/163/ for inclusions of complexity measures:
"... C(f) ≤ D(F) ≤ bs(f) · deg(f) ≤ deg(f)^3 ..."
]

Noise Sensitivity and Probability of Goodness in P(Good) LHS and RHS - Elaboration on previous
---
LHS of P(Good):
---
(1) Depends either on a natural (psuedo)random process or a Dictator Boolean Function that unilaterally decides outcome.

(2) When error is zero, probability of good decision is 1 (or) Pr(f(x) != f(y)) is zero where x and y are correlated bit strings obtained by a bit flip. In other words Noise in input does not alter outcome. If NS is Noise Sensitivity, probability of good outcome is 1-NS which is 1/2 + 1/2*Stability.

RHS of P(Good):
---
(1) Depends on the Noise Sensitivity of Boolean Function Composition - Maj(n)*SAT - Majn(SAT1, SAT2,....,SATn).

(2) When error is zero, probability of good decision is 1 because nCn*(1)^n*(1-1)^(0) becomes 1 in summation and all other terms vanish. If each individual voter has different probability of good decision (which is most likely in real

world elections) then it becomes Poisson Probability Distribution trial. Throughout this document, only Binomial Distribution is assumed and therefore NoiseStability of voters are assumed to be equal though the individual voter boolean functions might differ. In a generic setting each voter can have arbitrary decision function which need not be restricted to boolean functions. Throughout this document, only boolean voter judging functions are assumed.

(3) NoiseSensitivity is a fitting measure for voter error or voting error as it estimates the probability of how a randomly changed input distorts outcome of an election. Randomly changed input could be wrong assignment by voter, wrong recording by the voting process etc.,

(4) Perfect Voter has a SAT or Boolean Function that is completely resilient to correlation i.e Pr(f(x) != f(y)) = 0. The open question is: is it possible to find or learn such a Voter Boolean Function.

(5) Brute Force exponential algorithm to find perfect voter is by property testing:
- For 2^n bit strings find correlation pairs with random flips - 2^n * (2^n - 1) such pairs are possible.
- Test Pr(f(x) != f(y))
- This is not a learning algorithm.

(6) For perfect voter, probability p of good decision is 1-NS or 1/2+1/2*Stability and has to be 1 which makes NS=0 and Stability=1.

(7) Open Question: Is there a learning algorithm that outputs a Perfect Voter Decision Boolean Function f such that NS=Pr(f(x) != f(y)) = 0 for all correlated pairs x,y? (Or) Can LHS and RHS of P(Good) be learnt with NS=0 and Stability=1.Such a function is not influenced by any voting or voter errors. Majority is Stablest theorem implies that Majority function is the most stable of all boolean functions with least noise, but still it does not make it zero noise for perfect stability.

Is there some other function stabler than stablest - answer has to be no - But there is an open "Majority is the Least Stable" conjecture which asserts "yes" by [Benjamini-Kalai-Schramm]:
For Linear Threshold Functions (majority, weighted majority et al) f, Stability(f) >= Stability(Majn).
Majority is Stablest theorem is for small-influence LTFs (maximum influence of f < epsilon). If existence of perfect voter boolean function is proved it would be a generalization of this conjecture.

An obvious corollary is that for stablest voting boolean function, majority is apt choice (or) in the Majority voting circuit each voter has a majority as voter boolean function i.e it is a boolean function composition of Majority*Majority = Maj(Maj1,Maj2,Maj3,...,Majn). This composition of Maj*Maj is the stablest with least noise. This is also depth-2 recursive majority function mentioned in [RyanODonnell] - definition 2.6 in http://analysisofbooleanfunctions.org/ and Majority Voting with constituencies in https://sites.google.com/site/kuja27/IndepthAnalysisOfVariantOfMajorityVotingwithZFAOC_2014.pdf. Also the 2-phase document merit algorithm in http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf is a variant of this. A voter with Majority as boolean function trivially decides i.e if more than half of his/her variables are satisfied, outputs 1 without any fancy decision making or judging boolean function circuit (e.g an EXP-complete adversarial simulation). By far this is the stablest possible voting scheme unless [Benjamini-Kalai-Schramm] conjecture is true or there exists a 100% noise stable percolation boolean function - derived in (17) and (18) below.

(8) Theorem 2.45 in http://analysisofbooleanfunctions.org/ is important which proves Stability and NoiseSensitivity for infinite majority:

lt (n->infinity, n odd) Stability[Majn] = 2/pi*arcsin(rho)
(or)
lt (n->infinity, n odd) NS[delta,Majn] = 2/pi*delta + O(delta^1.5) where delta is the probability of correlation - flipping x to y.

(9) There are two random variables f(x)=f(y) and f(x) != f(y) with respective probabilities.

(10) This implies for infinite electorate, error probability is (an error that causes x to become y with probability delta):
lt (n->infinity, n odd) NS[delta,Majn] = 2/pi*delta + O(delta^1.5) where delta is the probability of correlation - flipping x to y.

(11) probability of good outcome = p = 1-NS:
p = 1 - lt (n->infinity, n odd) NS[delta,Majn] = 1 - 2/pi*delta - O(delta^1.5).

(12) If delta = 1:
lt (n->infinity, n odd) NS[delta,Majn] = 2/pi + O(1).
(and)
probability of good outcome = 1-NS = 1/2+1/2*Stability = 1-2/pi-O(1).

(13) In P(Good) RHS:

If the error probability of voter is 0 and probability of good decision is 1 then binomial summation becomes nCn*(p)^n*(1-p)^n = 1 and all other terms are zero. From (11) p can be 1 only if delta is 0 or no correlation flips exist. This proves the obvious because commonsensically votes are perfect iff there is no wrongdoing.

(14) In P(Good) LHS:

If error probability of dictator boolean function is 0 and p=1 then:
p = 1 - NS(DictatorFunction)
= 1 - delta^n
Similar to RHS p=1 iff delta=0
Thus if there is no Noise on either side LHS=RHS (or) LHS is a P algorithm to RHS EXP or PH=DC. This depends on existence of perfect voter function defined in (7). This just gives a counterexample when LHS can be as efficient as RHS and viceversa.

(15) [Benjamini-Kalai-Schramm] theorem states the condition for a boolean function to be noise sensitive:
A boolean function is noise sensitive if and only if L2 norm (sum of squares) of influences of variables in f tends to zero.
Therefore, for stability any voter boolean function in P(Good) RHS should not have the L2 norm of influences to tend to zero.

(16) "How much increasing sets are positively correlated" by [Talagrand] - http://boolean-analysis.blogspot.in/2007/03/how-much-are-increasing-sets-positively.html - defines the inequality (Chang's Level-1 inequality):
Weight(f) = sum_1_to_n(fouriercoeff(i)) <= O(mean^2*1/log(mean))
where Weight of boolean function f in n variables is the sum of squares of n fourier coefficients and mean is Pr(f(x)=1)-(Pr(f(x)=-1). This is alternatively defined as derivative of stability of f wrt rho at rho=0. This measure intuitively quantifies how Pr(f(x)=f(y)) changes as x becomes increasingly correlated with y (due to random flips). Related to this, point (20) is derived from Plancherel's theorem which equates E(f(x)g(x)) to summation(f_fourier(x)g_fourier(y)). But stability is E(f(x)f(y/x-correlated)). The noise operator E(f(y)) is introduced for each monomial in fourier series by which E(monomial(S)) = (rho^|S|*fourier_f(S)). E(f(y)) is thus equivalent to a new boolean function g(x) for which Plancherel formula can be applied - E(f(x)E(f(y))) = summation(rho^|S|*fourier_f(S)^2) = Noise stability. First derivative of Stability wrt rho at rho=0 is Weight(f) - in other words this

connects pseudorandomness with stability of a boolean function in terms of fourier coefficients.

(17) By Majority is Stablest Theorem if P(Good) RHS is a boolean function composition of Maj*Maj=Majn(Maj1,Maj2,...,Majn) then its stability is derived as:
   (1-2/pi * arccos(rho))*(1-2/pi * arccos(rho)) assuming that stability of the composition is the product like other measures
If the LHS of P(Good) is the Dictator Boolean Function its stability is (rho)^n (because all bits have to be flipped). By equating the two stability measures both sides following expression results:
   [cos(pi/2*(1-(rho)^n))]^2 = rho

For rho=0:

 [cos (pi/2*(1-0^n))]^2 = 0 hence LHS=RHS

For rho=1:

 [cos (pi/2*(1-1^n))]^2 = 1 hence LHS=RHS

But for rho=0.5:
----
 arccos 1/sqrt(2) = pi/2*(1-0.5^n)
 n = log(1-2/pi*0.7071)/log(0.5)
 n = 0.863499276 which is quite meaningless probably hinting that for uniform distribution the parity size n is < 1.

(18) In RHS the P(Good) binomial coefficient summation in terms of Noise Sensitivities of each Voter SAT:
nC0(1-NS)^0(NS)^n + nC1(1-NS)^1(NS)^(n-1) + nC2(1-NS)^2(NS)^(n-2) + ...
Above summation should ideally converge to holistic Noise Sensitivity of the Boolean Function Composition of Maj*Maj described previously for stablest voting:
       1/2 + 1/2*(1-2/pi*arccos(rho))^2 = nC0(1-NS)^0(NS)^n + nC1(1-NS)^1(NS)^(n-1) + nC2(1-NS)^2(NS)^(n-2) + ...

If each voter has boolean function with varied Noise Sensitivities, above becomes a Poisson Trial instead of Bernoulli. Above is a closed form for the goodness of voting expressed as the binomial coefficient summation which otherwise requires hypergeometric algorithms because for NS=a/b (!=0.5) summation becomes (nC0*(b-a)^0a^n + nC1(b-a)^1(a)^(n-1) + nC2(b-a)^2(a)^(n-2) + ...)/b^n and closed form is non-trivial as summation is asymmetric whereas the case with NS=1/2 is symmetric with all coefficients being plain vanilla binomial coefficients. Previous identity equates Noise Sensitivity of Majority Voting Circuit in two versions: Boolean Function Composition and Majority with Oracle Voter SAT inputs.

(19) There are classes of boolean functions based on percolation crossings (http://arxiv.org/pdf/1102.5761.pdf - Noise Sensitivity of Boolean Functions and Percolation) which decide if the paths (left-to-right) in a percolation random graph in which edges are created with a probability p, have a left-right crossing. Perturbed paths are analogous to correlated bit strings. [Russo-Seymour-Welsh] theorem states that the percolation crossing boolean functions are non-degenerate i.e function depends on all variables (quite intuitive as to find out a left-right crossing entire path has to be analyzed). Noise Stability Regime of the crossing events [page 68] answers the following question on noise stability of crossing events:

For epsilon=o(1/n^2*alpha4), Pr[fn(sequence)=fn(sequence with epsilon perturbation)] tends to zero at infinity where alpha4 is the four-arm crossing event probability (paths cross at a point on the percolation random graph creating four arms to the boundary from that point). In terms of fourier expansion coefficients this is: summation(fourier_coeff(S)^2) tending to zero, |

S| > n^2*alpha4. [Open question: Is such a percolation crossing the perfect voting boolean function required in (7) and does it prove BKS Majority is least stable conjecture? But this happens only when there are infinite number of Z^2 grid cells - n * n is infinite]. If this is the only available perfect boolean function with 100% stability, [Benjamini-Kalai-Schramm] conjecture is true and both LHS and RHS use only percolation as judge boolean functions(in compositions for each voter - Majority * Percolation), then LHS=RHS by P(Good) binomial summation convergence and LHS is class C algorithm for RHS PH-complete or EXP-complete algorithm where C is the hardness of LHS pseudorandom choice judge boolean function. Also noise probability epsilon=o(1/n^2*alpha4) can have arbitrarily any value <= 1 depending on n and alpha4.

Open question: If Percolation Boolean Functions are 100% noise stable (as in infinite grid previously), what is the lowerbound of such boolean functions (or) how hard and in what complexity class C should they be. This infinite version of percolation is a non-uniform polynomial circuit class (grid has n^2 coordinates).

(*IMPORTANT* - this requires lot of reviewing - if correct places a theoretical limit on when stability occurs) From Plancherel's theorem and stability relation mentioned in (16), summation(rho^|S|*fourier_f(S)^2) = Noise stability. When stability is 1, this summation is equal to 1. In other words rho^|S1| *fourier_f(S1)^2 + rho^|S2|*fourier_f(S2)^2 + ...+ - 1 = 0 which is a univariate polynomial in rho as variable with n roots and degree n. Polynomial Identity Testing algorithms are required to ensure that this polynomial is non-zero (coefficients are non zero). The real, positive, roots of this polynomial are the probabilities where a boolean function is resilient to noise and 100% stable. It non-constructively shows existence of such stable boolean functions. [Talagrand] proved existence of a random CNF with noise sensitivity > Omega(1) - described in http://www.cs.cmu.edu/~odonnell/papers/thesis.pdf. Noise stability depends on huge [S] as evidenced by stability = summation(rho^|S| *fourier_coeff(f(S))^2). What this implies is that for any boolean function, 100% stability is attainable at only n points in [0,1]. In other words there is no boolean function which is 100% noise stable for all points of rho in [0,1]. It also implies that BKS majority is least stable conjecture is true at these points of rho. Since coefficients of this polynomial are Fourier coefficients of the boolean function, roots of this polynomial i.e stability points are functions of Fourier coefficients.

(20) Noise stability in terms of Fourier coefficient weights is defined as:
summation(rho^k * W^k) for all degree-k weights where W = summation(fourier_coeff(S)^2) for all |S|, S in [n]

(21) Variant 1 of Percolation as a Judge boolean function is defined in 53.4 below which is in non-uniform NC1. If this is 100% noise stable judge boolean function for LHS and RHS noise stability also converges to 100%, then there is a non-uniform NC1 circuit for RHS PH-complete or EXP-complete DC uniform circuit. This gives a non-uniform algorithm for an RHS uniform circuit. Variant 2 of Percolation in 53.5 as Judge boolean function is also in non-uniform NC1 subject to 100% noise stability condition. In 53.6 a sorting network based circuit is described for Percolation boolean function.

(22) Also from Theorem 5.17 - [RyanODonnell] - http://analysisofbooleanfunctions.org/ - for any rho in [0,1], Stability of Infinite Majority is bounded as:
2/pi*arcsin ρ ≤ Stabρ[Majn] ≤ 2/pi*arcsin ρ + O(1/(sqrt(1-rho^2)*sqrt(n))).

(23) Peres' Theorem rephrases above bound for NoiseStability in terms of NoiseSensitivity for class of uniform noise stable linear threshold functions f (which includes Majority):
NoiseSensitivityδ[f] ≤ O(sqrt(δ))

(24) From http://arxiv.org/pdf/1504.03398.pdf: [Boppana-Linial-Mansour-Nisan]

theorem states that influence of a boolean function f of size S and depth d is:
Inf(f) = O((logS)^d-1)

(25) [Benjamini-Kalai-Schramm] Conjecture (different from Majority is least stable BKS conjecture which is still open) is the converse of above that states: Every monotone boolean function f can be epsilon (e) approximated by a circuit of depth d and of size exp((K(e)*Inf(f))^1/(d−1)) for some constant K(e). This conjecture was disproved by [ODonnell-Wimmer] and strengthened in http://arxiv.org/pdf/1504.03398.pdf.

(26) Class of uniform noise stable LTFs in (23) are quite apt for Voter Boolean Functions in the absence of 100% noise stable functions, which place an upperlimit on decision error. Depth hierarchy theorem in http://arxiv.org/pdf/1504.03398.pdf separates on what fraction, 2 circuits of varying depths agree in outputs.

Additional References:

14.1 k-SAT and 3-SAT - http://cstheory.stackexchange.com/questions/7213/direct-sat-to-3-sat-reduction
14.2 k-SAT and 3-SAT - http://www-verimag.imag.fr/~duclos/teaching/inf242/SAT-3SAT-and-other-red.pdf
14.3 Switching Lemma
14.4 Donald Knuth - Satisfiability textbook chapter - http://www-cs-faculty.stanford.edu/~uno/fasc6a.ps.gz - Quoted excerpts:
" ... Section 7.2.2.2 Satisfiability
Satisfiability is a natural progenitor of every NP-complete problem’
Footnote: At the present time very few people believe that P = NP. In other words, almost everybody who has studied the subject thinks that satisfiability cannot be decided in polynomial time. The author of this book, however, suspects that N^O(1)–step algorithms do exist, yet that they’re unknowable. Almost all polynomial time algorithms are so complicated that they are beyond human comprehension, and could never be programmed for an actual computer in the real world. Existence is different from embodiment. ..."
14.5 Majority and Parity not in AC0 - www.cs.tau.ac.il/~safra/ACT/CCandSpaceB.ppt, http://www.math.rutgers.edu/~sk1233/courses/topics-S13/lec3.pdf
14.6 Sampling Fourier Polynomials - http://cstheory.stackexchange.com/questions/17431/the-complexity-of-sampling-approximately-the-fourier-transform-of-a-boolean-fu

14.7 Kummer's theorem on binomial coefficient summation - quoted from Wikipedia: "In mathematics, Kummer's theorem on binomial coeffients gives the highest power of a prime number p dividing a binomial coefficient. In particular, it asserts that givenintegers n ≥ m ≥ 0 and a prime number p, the maximum integer k such that p^k divides the binomial coefficient \tbinom n m is equal to the number of carries when m is added to n − m in base p. The theorem is named after Ernst Kummer, who proved it in the paper Kummer (1852). It can be proved by writing \tbinom{n}{m} as \tfrac{n!}{m! (n-m)!} and using Legendre's formula."

14.8 Summation of first k binomial coefficients for fixed n:
- http://mathoverflow.net/questions/17202/sum-of-the-first-k-binomial-coefficients-for-fixed-n
- https://books.google.co.in/books?id=Tn0pBAAAQBAJ&pg=PA61&lpg=PA61&dq=summation+of+first+k+binomial+coefficients&source=bl&ots=sqB_iQweyS&sig=ye5TxmCkjP-Ud5JnWilMzqOkDjM&hl=en&sa=X&ved=0CFgQ6AEwCWoVChMI6rDEo9nVxwIVR0yOCh2AXgpz#v=onepage&q=summation%20of%20first%20k%20binomial%20coefficients&f=false
which imply bounds for the summation:
- sum_k_2n(2nCi) = 2^2n - sum_0_k-1(2nCi)
- 2^2n - sum_0_k-1(2nCi) > 2^2n - (2nCk) * 2^(2n) / 2*(2nCn)

14.9 Probabilistic boolean formulae - truth table with probabilities - simulates

the Voter Circuit SAT with errors
http://www.cs.rice.edu/~kvp1/probabilisticboolean.pdf

14.10 BPAC circuits (= Probabilistic CNF?)
http://www.cs.jhu.edu/~lixints/class/nw.pdf

14.11 Mark Fey's proof of infinite version of May's theorem for 2 candidate majority voting - Cantor set arguments for levels of infinities - https://www.rochester.edu/college/faculty/markfey/papers/MayRevised2.pdf

14.12 Upperbounds for Binomial Coefficients Summation - http://mathoverflow.net/questions/17202/sum-of-the-first-k-binomial-coefficients-for-fixed-n - Gallier, Lovasz bounds - where Lovasz bound is:

$$f(n,k) \leq 2^{(n-1)} * \exp((n-2k-2)^2/4(1+k-n))$$

for f(n,k) = sum_i_0_to_k(nCi). Proof by Lovasz at:
http://yaroslavvb.com/upload/lovasz-proof2.pdf

14.13 (*IMPORTANT*) Connections between Hypergeometric series and Riemann Zeta Function :

- http://matwbn.icm.edu.pl/ksiazki/aa/aa82/aa8221.pdf - This relation implies that:
  - P(good) binomial coefficient infinite series summation (and hence the Psuedorandom Vs Majority Voting circuit) which require Hypergeometric functions
  - and Complement Function circuit special case for RZF (and hence the Euler-Fourier polynomial for Riemann Zeta Function circuit)

are deeply intertwined. Kummer theorem gives the p-adic number (power of a prime that divides an integer) for dividing a binomial coefficient which relates prime powers in Euler product RZF notation and Binomial coefficients in Majority voting summation.

Above implies that binomial coefficients can be written in terms of prime powers and vice-versa: P(good) series can be written as function of sum of prime powers and Euler Product for RZF can be written in terms of binomial coefficients. Also Euler-Fourier polynomial obtained for complement function circuit (Expansion of http://arxiv.org/pdf/1106.4102v1.pdf described in 10) can be equated to binomial coefficient summation. If these two apparently unrelated problems of Complexity-Theoretic Majority voting and Analytic-Number-Theoretic Riemann Zeta Function are related then complexity lowerbound for some computational problems might hinge on Riemann Zeta Function - for example PH=DC circuit problems .

14.15 ACC circuits and P(Good) RHS circuit

ACC circuits are AC circuits augmented with mod(m) gates. i.e output 1 iff sum(xi) is divisible by m. It is known that NEXP is not computable in ACC0 - http://www.cs.cmu.edu/~ryanw/acc-lbs.pdf [RyanWilliams]. From (129) above the P(good) RHS can be an EXP circuit if the boolean functions of the voters are of unrestricted depth. Thus open question is: can RHS of P(Good) which is deterministic EXP, be computed in ACC0. [RyanWilliams] has two theorems. Second theorem looks applicable to P(Good) RHS circuit i.e "E^NP, the class of languages recognized in 2^O(n) time with an NP oracle, doesn't have non-uniform ACC circuits of 2^n^o(1) size". Boolean functions are the NP oracles for this P(Good) Majority Voting EXP circuit if the variables are common across voters. Applying this theorem places strong super-exponential lowerbound on the size of the P(Good) majority voting circuit.

14.16. Infinite Majority as Erdos Discrepancy Problem

Related to (132) - May's Theorem proves conditions for infinite majority with

sign inversions {+1,-1} and abstention (zero). Erdos discrepancy problem which questions existence of integers k,d, and C in sequence of {+1,-1} such that sum_0_to_k(x(i*d)) > C and has been proved for C=infinite - [TerenceTao] - http://arxiv.org/abs/1509.05363. For spacing d=1, Erdos Discrepancy Problem reduces to Infinite Majority and as C is proved to be infinite, it looks like an alternative proof of non-decrementality condition in Infinite version of May's Theorem.

14.17 Noise Sensitivity of Boolean Functions and Percolation - http://arxiv.org/pdf/1102.5761.pdf, www.ma.huji.ac.il/~kalai/sens.ps

14.18 Binomial distributions, Bernoulli trials (which is the basis for modelling voter decision making - good or bad - in P(Good) series) - http://www.stat.yale.edu/Courses/1997-98/101/binom.htm

14.19 Fourier Coefficients of Majority Function - http://www.contrib.andrew.cmu.edu/~ryanod/?p=877, Majority and Central Limit Theorem - http://www.contrib.andrew.cmu.edu/~ryanod/?p=866

14.20 Hastad's Switching Lemma - http://windowsontheory.org/2012/07/10/the-switching-lemma/, http://www.contrib.andrew.cmu.edu/~ryanod/?p=811#lemrand-restr-dt

14.21 Degree of Fourier Polynomial and Decision tree depth - http://www.contrib.andrew.cmu.edu/~ryanod/?p=547#propdt-spectrum (degree of fourier polynomial < decision tree depth, intuitive to some extent as each multilinear monomial can be thought of as a path from root to leaf)

14.22 Judgement Aggregation (a generalization of majority voting) - Arriving at a consensus on divided judgements - http://arxiv.org/pdf/1008.3829.pdf.

14.23 Noise and Influence - [Gil Kalai, ICS2011] - https://gilkalai.files.wordpress.com/2011/01/ics11.ppt

14.24 Doctrinal Paradox (or) Discursive dilemma - Paradoxes in Majority voting where both Yes and No are possible in judgement aggregations - https://en.wikipedia.org/wiki/Discursive_dilemma

---

15. (FEATURE - DONE-Minimum Implementation using NetworkX algorithms) Add Graph Search (for example subgraph isomorphism, frequent subgraph mining etc.,) for "inferring" a graph from dataset with edges as relations and nodes as entities(astronomical or otherwise).Reference survey of FSM algorithms: http://cgi.csc.liv.ac.uk/~frans/PostScriptFiles/ker-jct-6-May-11.pdf. The context of "Graph Search" includes deducing relationships hidden in a text data by use of WordNet(construction of wordnet subgraph by Definition Graph Recursive Gloss Overlap etc.,), mining graph patterns in Social Networks etc., At present a WordNet Graph Search and Visualizer python script that renders the definition graph and computes core numbers for unsupervised classification (subgraph of WordNet projected to a text data) has been added to repository.

16. (FEATURE - DONE) Use of already implemented Bayesian and Decision Tree Classifiers on astronomical data set for prediction (this would complement the use of classifiers for string mining) - autogen_classifier_dataset/ directory contains Automated Classified Dataset and Encoded Horoscopes Generation code.

17. (FEATURE - DONE) Added Support for datasets other than USGS EQ dataset(parseUSGSdata.py). NOAA HURDAT2 dataset has been parsed and encoded pygen horoscope string dataset has been autogenerated by parseNOAA_HURDAT2_data.py

18. (FEATURE - Minimum Functionality DONE-TwitterFollowersGraphRendering, Centrality) (Astro)PsychoSocialAnalysis - Social Network Graph Analysis and

Sentiment Analysis of Social Media and drawing inferences on Psychology and Human Opinion Mining in broader sense - for example, how a social network forms, how opinions, edges and vertices appear over time , Rich-get-Richer in Random Network Erdos-Renyi Model - which by majority opinion seems to be a bad model choice for Social Networking (Do people flock to popular social groups?), Bonacich Power Centrality (a social prestige measure predating PageRank) etc.,. Related to point 15.

References:

18.1 Networks,Crowds,Markets - http://www.cs.cornell.edu/home/kleinber/networks-book/networks-book.pdf
18.2 Introduction to Social Network Methods - http://faculty.ucr.edu/~hanneman/nettext/
18.3 Bonacich Power Centrality - http://www.leonidzhukov.net/hse/2014/socialnetworks/papers/Bonacich-Centrality.pdf
18.4 Modelling Human Emotions - http://www.jmlr.org/proceedings/papers/v31/kim13a.pdf
18.5 Depression Detection - http://www.ubiwot.cn/download/A%20Depression%20Detection%20Model%20Based%20on%20Sentiment%20Analysis%20in%20Micro-blog%20Social%20Network.pdf
18.6 Market Sentiments - http://www.forexfraternity.com/chapter-4-fundamentals/sentiment-analysis/the-psychology-of-sentiment-analysis
18.7 Sentiment Analysis - http://www.lct-master.org/files/MullenSentimentCourseSlides.pdf
18.8 Dream Analysis - http://cogprints.org/5030/1/NRC-48725.pdf
18.9 Opinion Mining - http://www.cs.uic.edu/~liub/FBS/SentimentAnalysis-and-OpinionMining.pdf
18.10 Linked - [Albert Laszlo Barabazi] - [Ginestra Bianconi - http://www.maths.qmul.ac.uk/~gbianconi/Condensation.html] Bose gas theory of networks - Bose-Einstein Condensate is equivalent to evolution of complex networks - Page 101 on Einstein's Legacy - "Nodes in Network are energy levels of subatomic paritcles and Links across nodes are subatomic particles" - "Winner takes it all" - particles occupying lowest energy correspond to people flocking to a central node in social networks - This finding is from behaviour of large scale networks viz., WWW, Social networks etc., This Condensate theory has striking applications to Recursive Gloss Overlap Graph construction and using it for unsupervised classification based on core numbers - each network and for that matter a graph constructed from text document is a macrocosmic counterpart of quantum mechanics.
18.11 Small World Experiment and Six Degrees of Separation - [Stanley Milgram] - https://en.wikipedia.org/wiki/Small-world_experiment
18.12 LogLog estimation of Degrees in Facebook graph - 3.57 Degrees of Separation - https://research.facebook.com/blog/three-and-a-half-degrees-of-separation/

19. (FEATURE - DONE-PAC Learnt Boolean Conjunction) Implement prototypical PAC learning of Boolean Functions from dataset. Complement Boolean Function construction described in http://arxiv.org/abs/1106.4102 is in a way an example of PAC learnt Boolean Function - it is rather C Learnt (Correct Learning) because there is no probability or approximation. Complement Boolean Function can be PAC learnt (with upperbounded error) as follows:

19.1 There are two datasets - dataset1 of size 2^n of consecutive integers and dataset2 of first n-bit prime numbers of size 2^n

19.2 Each element of dataset1 is mapped to i-th bit of the corresponding prime number element in the dataset2. Boolean Conjunction is learnt for each of the i mappings.

19.3 Above step gives i probabilistic, approximate, correct learnt boolean conjunctions for each bit of all the prime numbers.

19.4 Reference: PAC Learning - http://www.cis.temple.edu/~giorgio/cis587/readings/pac.html

19.5 PAC Learnt Boolean Conjunction is an approximated Decision Tree.

19.6 PAC learning is based on Occam's Razor (no unnecessary variables) PAC Learning implementation for learning patterns in Binary encoded first 10000 Prime Numbers have been committed to AsFer repository (Commit Notes 324)

20. (FEATURE - DONE) Integrate AstroInfer to VIRGO-KingCobra-USBmd Platform which makes it an Approximately Intelligent Cloud Operating System Framework - a Cloud OS that "learns" ,"decides" from datasets and "drives" using AstroInfer code. AsFer+USBmd+VIRGO+KingCobra integrated cloud platform with machine learning features together have been christened as "Krishna iResearch Intelligent Cloud Platform" or "NeuronRain". It has to be mentioned here that NeuronRain cloud differs from traditional cloud in following ways:

20.1 Usual cloud platforms support only application-application layer RPCs over an existing operating system.

20.2 NeuronRain differs in that aspect by adding new cloud features (with advanced features in development) like RPC, distributed kernel memory and network filesystem related system call clients and kernel modules kernelsocket-server-listeners into linux kernel (a fork-off in that respect)

20.3 Cloud support within kernelspace has lot of advantages - low network latency because of kernelspace-kernelspace cloud communication by kernel sockets, userspace to kernelspace communication (a userspace application like telnet can connect to a kernel socket listener of a VIRGO driver in kernelspace), userspace-userspace communication by kernel upcalls to userspace from kernelspace.

20.4 Kernelspace-Kernelspace communication is a privileged mode privy to kernel alone and userspace applications shouldn't be able to access kernelspace socket data.

20.5 At present the data sent over kernel sockets is not encrypted which requires OpenSSL in kernel. Kernel doesn't have OpenSSL support presently and it is not straightforward to implement SSL handshake in NeuronRain cloud. But there are MD5 and other hashing libraries (e.g crypto) available for encrypting data before before they are sent over cloud.

20.6 There are ongoing efforts to provide Transport Layer Security (TLS) in kernel sockets which are not yet mainline. Hopefully kernel sockets in future kernel mainline versions would support AF_KTLS sockets and thus NeuronRain cloud implicitly is TLSed :

20.6.1 KTLS - [DaveWatson - Facebook] - https://lwn.net/Articles/666509/ and https://github.com/ktls/af_ktls

20.6.2 IPSec for kernel sockets - [SowminiVaradhan - Oracle] - http://www.netdevconf.org/1.1/proceedings/slides/varadhan-securing-tunneled-kenel-tcp-udp-sockets.pdf

20.7 Usual userspace-userspace network communication has following routing - user1----kernel1----kernel2----user2 for two cloud nodes node1 and node2. NeuronRain cloud communication has following routings - user1----kernel1---kernel2, kernel1---kernel2, user1---kernel1--kernel2--user2. With VFS filesystem API in kernel, both userspace and kernelspace mode communication can be persisted in filesystem and better than an equivalent application layer persistence mechanisms which require lot of internal kernel calls for disk writes. Thus kernel-kernel communication saves lot of user layer to kernel disk write invocations.

21. (FEATURE - DONE-Minimum Implementation) Unsupervised Named Entity Recognition and Part-of-Speech tagging using Conditional Random Fields(CRF) and Viterbi path computation in CRF. Bayesian Network and Belief propagation might be implemented if necessary - as they are graphical models where graph edges are labelled with conditional probabilities and belief potentials respectively and can be part of other larger modules.

##
############

A. Design Notes on Mining Numerical Datasets

##
############

22. (FEATURE - DONE-Minimum Implementation) Period Three theorem

(www.its.caltech.edu/~matilde/LiYorke.pdf) which is one of the earliest results in Chaotic NonLinear Systems, implies any data that has a periodicity of 3 can have larger periods. Fractals, Mandelbrot-Julia Sets have modelled natural phenomena. Thus finding if a data has periodicity or Chaos and if it has sensitive dependence on initial conditions (for example logistic equations similar to x(n+1) = kx(n)(1-x(n))) is a way to mine numerical data. Computation of Hausdorff-Besicovitch Dimension can be implemented to get Fractal Dimension of an uncountable set (this has to be visualized as a fractal curve than set of points). In addition to these, an Implementation of Chaotic PRG algorithms as described in https://sites.google.com/site/kuja27/ChaoticPRG.pdf?attredirects=0 and https://sites.google.com/site/kuja27/Analysis%20of%20a%20Randomized%20Space%20Filling%20Algorithm%20and%20its%20Linear%20Program%20Formulation.pdf?attredirects=0 can be done.

23. (FEATURE - DONE) Entropy of numerical data - Sum of -Pr(x)LogPr(x) for all possible outcome probabilities for a random variable. This is already computed in Decision Tree Classifier as impurity measure of a dataset. Also a python Entropy implementation for texts has been added to repository - computes weighted average of number of bits required to minimum-describe the text.

##
##

==
==
(THEORY) DECIDABILITY OF EXISTENCE AND CONSTRUCTION OF COMPLEMENT OF A FUNCTION
(important draft additions to http://arxiv.org/pdf/1106.4102v1.pdf - quite experimental, non-conventional and not necessarily correct)
==
==

(*)24. (DONE) COMPLEMENT OF A FUNCTION - Approximating or Interpolating with a Polynomial: This is quite expensive and is undecidable for infinite sets. Better alternative is to approximate with Spline interpolant polynomials, for example, cubic splines which have less approximation error. (On a related note, algorithms described by the author (that is, myself) in http://arxiv.org/pdf/1106.4102v1.pdf for construction of a complement of a function are also in a way eliciting pattern in numerical data. The polynomial interpolation algorithm for complement finding can be improved with Spline interpolants which reduce the error and Discrete Fourier Transform in addition to Fourier series for the boolean expression. Adding this as a note here since TeX file for this arXiv submission got mysteriously deleted and the PDF in arXiv has to be updated somehow later.). A test python script written in 2011 while at CMI for implementing the above is at: http://sourceforge.net/p/asfer/code/HEAD/tree/python-src/complement.py. Also Trigonometric Polynomial Interpolation which is a special case of Polynomial interpolation, has following relation to DFT X(i)s:

p(t) = 1/N [X0 + X1*e^(2*pi*i*t) + ... upto XN] and
p(n/N) = xn

Thus DFT amd interpolation coincide in the above.

24a. Theorems on prime number generating polynomials - special case is to generate all primes or integral complement of xy=z
----

Legendre - There is no rational algebraic function which always gives primes.
Goldbach - No polynomial with integer coefficients can give primes for all integer values
Jones-Sato-Wada-Wiens - Polynomial of degree 25 in 26 variables whose values are exactly primes exists

Polynomial interpolation and Fourier expansion of the boolean function in http://arxiv.org/pdf/1106.4102v1.pdf probably would have non-integral coefficients due to the above.

24b. Circuit construction for the complement function boolean DNF constructed in http://arxiv.org/pdf/1106.4102v1.pdf

---

The DNF constructed for complement function has to be minimized (through some lowerbound techniques) before it is represented as a circuit because the number of clauses could be exponential. The circuit minimization problem has been shown to be NP-complete(unpublished proof by Masek, [GareyJohnson]). The above complement function boolean DNF would yield a constant depth 2 circuit (probably after minimization also) with unbounded fanin. If the size of the circuit is polynomial it is in AC (=NC). The size of the circuit depends on the boolean DNF constructed above which inturn depends on input complement set. Thus circuit depends on input making it a Non-Uniform AC circuit. Though complement function is undecidable as described in http://arxiv.org/pdf/1106.4102v1.pdf, Non-uniform circuits "decide" undecidable languages by definition.

Old draft of the complement function - https://sites.google.com/site/kuja27/ComplementOfAFunction_earlier_draft.pdf?attredirects=0 describes this and proposes a name of co-TC0 as integer multiplication is in TC0 (threshold circuits) which may not be true if the circuit size is super-polynomial. Super-polynomial size circuits are DC circuits allowing exponential size. Assuming polynomial size the above may be named as "Non-uniform co-TC" for lack of better naming.

Due to equivalence of Riemann Zeta Function and Euler's theorem - inverse(infiniteproduct(1-1/p(i)^z)) for all primes p(i) - gives a pattern in distribution of primes which is the Riemann Hypothesis of Re(nontrivial zeroes of RZF)=0.5. Complement function for xy=z obtained by http://arxiv.org/pdf/1106.4102v1.pdf by polynomial interpolation or Fourier approximation polynomial of the DNF boolean formula can thus be conjectured to have a strong relation to RZF or even generalize RZF. In circuit parlance, the DNF boolean formula and its minimized Non-uniform AC(if polynomial sized) circuit that outputs prime number bits, thus are related to non-trivial zeroes of Riemann Zeta Function. Another conjecture that can be made is that if the real part of the non-trivial zeroes is 0.5 in RZF, then the Non-uniform circuit family constructed for the above complement function DNF should also have a pattern in the circuit graph drawn - subgraphs of the circuit family of graphs have some common structure property.

Fourier polynomial of a boolean formula is of the form:

$$f(x) = \text{Sigma\_S}(\text{fouriercoeff}(S)\text{parityfn}(S))$$

and is multilinear with variables for each input. S is the restriction or powerset of the bit positions.

Thus if a prime number is b-bit, there are b fourier polynomials for each bit of the prime. Thus for x-th prime number fourier expression is,

$$P(x) = 1+2*fx1(x)+2^2*fx2(x).....+2^b*fxb(x)$$

where each fxi() is one of the b fourier expansion polynomials for prime bits.

24c. Riemann Zeta Function written as Euler-Fourier Polynomial :

Fourier polynomials for each prime can be substituted in Euler formula and equated to Riemann Zeta Function:

RZF = Inverse((1-1/P(x1)^s)(1-1/P(x2)^s))...ad infinitum----------

(1).

Non-trivial complex zero s is obtained by,
P(xi)^s = 1
P(xi) = (1)^(1/s)
1+2*fx1(xi)+2^2*fx2(xi).....+2^b*fxb(xi) = (1)^(e^(-i*theta)/r) after rewriting in phasor notation s=r*e^(i*theta).-----------(2)

Above links Fourier polynomials for each prime to roots of unity involving non-trivial zeroes of Riemann Zeta Function in RHS. LHS is the xi-th prime number. Since LHS is real without imaginary part, RHS should also be real though exponent involves imaginary part. There are as many roots of unity in RHS as there are prime numbers. The radian is taninverse which is semi-determined assuming RH with Re(s) = 0.5.

LHS is multilinear with at most b variables for b bits of the prime number. A striking aspect of the above is that Fourier parity function is +1 or -1 and a lot of them might cancel out in the above summed up polynomial in LHS after substitution and rewriting.

If s is written as a+ic, then (1)^(1/s) = (1)^(1/a+ic) = (1)^((a-ic)/(a^2+c^2))

If RH is true a=0.5 and above becomes, (1)^((0.5-ic)/(0.25+c^2)).

1+2*fx1(xi)+2^2*fx2(xi).....+2^b*fxb(xi) = (1)^((0.5-ic)/(0.25+c^2))
----------(3)

Thus imaginary part c has one-to-one correspondence to each prime. Non-trivial zeroes are usually found using functional notation of RZF instead of the above - applying Analytic Continuation that extends the domain in steps to be exact.

24d. Delving deeper into (1) above which equates Riemann Zeta Function with Complement Function Fourier Polynomials substituted in Euler's infinite product (Hereinafter referred to as Euler-Fourier polynomial) - Pattern in primes from above Euler-Fourier polynomial:

Finding pattern in distribution of primes is equivalent to finding pattern in above set of prime bit circuits or fourier polynomials for each prime - The family of circuits for primes above has to be mined for circuit DAG subgraph patterns which is more of machine learning than complexity (there is almost no complexity theoretic reference to this aspect). The set of fourier polynomials can be mined for linear independence for example. (1),(2) and (3) are multiple ways of expressing same relation between complement function boolean circuit and RZF.

Similar to zeros of RZF, Fourier poynomial obtained from Euler's formula by substituting complement function prime bits fourier polynomials gives a very complex polynomial whose degree could be d*s in b variables, where d is the maximum degree in prime bit fourier polynomials. In randomized setting, Schwartz-Zippel Lemma can be applied for finding an upperbound for number of roots of this Fourier-Euler polynomial which is the traditional tool in Polynomial Identity Testing. Using the lemma, number of roots of the above polynomial is <= d*s/ |K| where K is the random finite subset of b variables in the boolean complement function. If |K| = b, then number of roots is upperbounded by d*s/b.[This is assuming rewriting as (2) or (3)]. But degree is a complex number d*s (It is not known if there is a Schwartz-Zippel Lemma for complex degree). The roots of this Fourier-Euler polynomial should intuitively correspond to zeros of Riemann Zeta Function which probably gives a complexity theoretic expression of Riemann Zeta Function (than Analytic Number Theoretic). This implies that there should be complex roots also to the Euler-Fourier polynomial which itself is puzzling on what it means to have a complex number in

boolean circuits.

Important thing to note is that what it means to be a zero of the Euler-Fourier polynomial [(1),(2) and (3)] which has complex degree variable and also boolean variables within each clause. RZF just has the s in exponent. Thus Euler-Fourier polynomial is more generic and fine-grained. But zeroes of this polynomial are both within the multilinear components and the exponents instead of just in exponent in RZF. This might reveal more pattern than what RH predicts conjecturally. Moreover due to non-uniformity each Fourier polynomial component substituted in Euler formula for each prime, is different.

Sensitivity and Block sensitity of the above complement function circuit should also have a strong relation to prime distribution as sensitivity is the number of input bits or block of bits that are to be flipped for change in value of the circuit(prime bit is the value) . For each Fourier polynomial of a prime bit above the sensitivity measure would differ and the prime distribution is proportional to collective sensitivity of all the prime bit circuit Fourier polynomials . Also the degree of approximating Fourier polynomial is lower bounded by sensitivity (www.cs.cmu.edu/~odonnell/papers/analysis-survey.pdf).

24e. Arithmetic Circuit for Euler Product and RZF - Alternative formulation to Euler-Fourier polynomials

---

Instead of a boolean circuit above can be an arithmetic circuit (with * and + gates with elements of some field as inputs to these) alternatively which is useful in representing polynomials. The division in the above Euler product can be replaced by the transform f/g = f/(1-(1-g)) = f*(1+(1-g)+(1-g)^2+...) by geometric series expansion. This is how division is implemented using powering and hence multiplication in NC circuits. Above Euler product can be represented as a depth 3 Pi-Sigma-Pi ---- * for product of each prime clause, + for subtraction within each clause, * for raising (1/p) to power s ---- circuit assuming 1/p. Powering with complex exponent s requires representing s as a 2*2 matrix M(s) where a and b are real and imaginary parts of s:

$$M(s) = \begin{vmatrix} a & -b \\ b & a \end{vmatrix}$$

Thus p^s can be written as p^M(s) with matrix exponent. Rewriting this as e^(M(s) ln p) and expanding as series,

$$e^{(M(s) \ln p)} = 1 + M \ln p + M*M \ln p^2/2! + ...$$

and $1 - 1/p^s = 1 - e^{(-M(s) \ln p)} = 1-(1-M*\ln p+ M*M \ln p^2/2! - ...) = M*\ln p - M*M \ln p^2/2! + ...$ (alternating +,- terms ad infinitum)

Thus each prime clause in the Euler product is written as an infinite summation of 2*2 matrix products. Using the transform f/g above division 1/(1-1/p^s) for each prime clause in the Euler product can be reduced to powering. Product gate at root multiplies all such prime clauses. Obviously above circuit is exponential and probably is the lowerbound for this circuit. If the product is made finite then a non-uniform arithmetic circuit family is created depending on input advice and degree of the above polynomial varies. This is an alternative to Euler-Fourier polynomial obtained based on complement function boolean circuit. No Fourier polynomial for prime bits is used here but prime power as such is input to arithmetic gates.

Instead of Euler formula, circuit for Riemann Zeta Function can be constructed with + gate at the root and circuits for n^s (or e^(s ln p)) computation at the leaves using the above series expansion. For first n terms of RZF, the circuit represents the polynomial (with s replaced by the 2*2 matrix for the complex s):

$$RZF(n) = n - M(s) * (\ln 2 + \ln 3 + ...+ \ln n)/1! + M(s)^2 * (\ln 2^2 +$$

ln3^2 +...+ lnn^2)/2! + ... + M(s)^n * (ln2^n + ...+lnn^n)/n!

The reason for drawing above arithmetic circuit is to represent the Riemann Zeta Function in complex plane as a circuit that depends on Matrix representation of complex number field(determinant of the matrix is the norm) and relate the roots of it to non-trivial zeroes of Riemann Zeta Function. The geometric intuition of Schwartz Zippel lemma is to find the probability of a point being on this circuit represented polynomial's surface. In the above circuit the variable is the Matrix M (matrix representation for the zero s). The imaginary part is hidden within the 2*2 matrices. The degree of above polynomial is n and is univariate in M. Non-uniform Sigma-Pi-Sigma Arithmetic circuit for above restricted version of Riemann Zeta Function can be constructed similar to the above for arbitrary n.

Using Taylor series expansion and Jordan Normal Form above can be written as a huge square matrix.(http://en.wikipedia.org/wiki/Matrix_function). Taylor series for a real function f(x) = f(0) + f'(0)/1! + f''(0)/2! + ... and 2*2 matrix for complex zero can be written in Jordan Normal Form as XYX^(-1) with Y being the Jordan block of diagonal fixed entries and superdiagonal 1s. Evaluating f(Y) with Taylor expansion gives a square matrix. Thus Riemann Zeta Function has a matrix in Jordan Normal Form. Finding zeros of RZF is equivalent to solving system of equations represented as Jordan Nomal Form using Gauss-Jordan Elimination. The matrix is already upper triangular and can be equated to zero matrix.

Eigen values of Chaotic systems Wave modelling Random matrices (matrix as random variable) have been shown to have a striking relation to zeros of RZF - Spacing of Random matrix eigenphases and RZF zeros have been conjectured to be identical (Montgomery). From complexity standpoint, the characteristic polynomial or the determinant of such Random Matrices can be computed by determinant circuits which have polynomial size and polylog depth.

24f. Can be ignored - quite Experimental - Different way to reduce the complex exponent in (1)

From (1), P(xi)^s = 1.
=> P(xi) ^ (k+il) = 1 where s=k+il
=> P(xi)^k * P(xi)^(il) - 1 = 0
 P(xi)^k * [cos(l*ln(P(xi))) + i*sin(l*ln(P(xi)))] - 1 = 0 -----------------(4)

Thus real and imaginary parts can be equated to zero independently as,
      P(xi)^k * cos(l*ln(P(xi))) = 1 --
(5)

      P(xi)^k * sin(l*ln(P(xi))) = 0 --
(6)

From (5) and (6), it can be deduced that:

      tan(l*ln(P(xi))) = 0 --
(7)

From (7), it can be inferred that Re(s)=k is not involved and PIT has to be done only on the b+1 variables (b bits in primes and the Im(s)=l). This probably points to the fact that for all primes, the prime distribution is independent of the Re(s). Series expansion of (7) gives,

      tan(T) = T + T^3/3 + 2*T^5/15 + ... = 0 where T = l*ln(P(xi)) ---------
(8)

l*ln(P(xi)) = 0 =>
      l = 0 or ln(P(xi)) = 0
      P(xi) = 1
both of which can not hold.

From (5),
      P(xi)^k = sec(l*ln(P(xi))) ----------
(9)

if T = l*lnP(xi), e^(T/l) = P(xi)

      P(xi)^k = e^(kT/l) = 1+T^2/2 + 5T^4/24 + 61*T^6/720 + 277T^8/8064 + ...
(expansion of sec)   ---------- (10)

      1+kT/l + (kT/l)^2/2! + (kT/l)^3/3! + ... =   1 + T^2/2 + 5T^4/24 + ...
(expansion of e^(KT/l)) --------(11)

(5) can also be written as,

      ln P(xi)^k =   ln(sec(l*ln(P(xi)))

      k   =   ln[sec(l*ln(P(xi)))] / lnP(xi)
      ----------(12)

By RH k has to be 0.5 irrespective of RHS.

Approximation of l:

By assuming k=0.5 and Using series expansion of cos(x) - choosing only first two terms, l is approximately,

      P(xi) = 1/[cos(l*lnP(xi))]^2
      ----------(13)
      l = sqrt(2)*sqrt(sqrt(P(xi)) - 1) / ln (P(xi)
      ----------(14)

More on (7):
---
      tan(l*ln(P(xi))) = 0 implies that l*ln(P(xi)) = n*pi for some integer n.
      ln(P(xi)) = n*pi/l
      P(xi) = e^(n*pi/l)

--------(15)

Above equates the Fourier polynomial for xi-th prime in terms of exponent of e with Imaginary part l of some RZF zero. It is not necessary that primes have one-one correspondence with zeros in the same order. All above just imply that it is true for some prime(and its Fourier polynomial) and some RZF zero that satisfy these identities.

24g. Ramanujan Graphs, Ihara Zeta Function and Riemann Zeta Function and Special case of Complement Function
----

A graph is Ramanujan graph if it is d-regular and eigen values of its adjacency matrix are sqrt(d-1)*2 or d. Ihara zeta function similar to RZF is a Dirichlet series that is based on prime cycle lengths of a graph defined as ProductOf(1/1-q^(-s*p)) where s is a zero and p prime for a (q+1)-regular graph. Thus a reduction is already available from RZF to a graph. The Ihara Zeta Function satisfies Riemann Hypothesis iff graph is Ramanujan - this follows from Ihara

identity that relates Ihara Zeta Function and the adjacency matrix of a graph. Thus proving above conjecture for Euler-Fourier polynomial of complement function for primes boolean and arithmetic circuits family might need this gadget using prime cycles.

If a set of p-regular graphs for all primes is considered,then Riemann Zeta Function can be derived (using Ihara Zeta Function identity) as a function of product of Ihara Zeta Functions for these graphs and a function of the determinants of adjacency matrices for these graphs divided by an infinite product similar to Euler product with (1+1/p^s) clauses instead of (1-1/p^s). This requires computing product of determinants of a function of adjancency matrices for these graphs. A regular connected graph is Ramanujan if and only if it satisfies RH. Zeroes of the individual Ihara zeta functions are also zeroes of this product for the set of graphs and hence for the RZF. Intuitively the product might imply set of all paths of all possible lengths across these graphs. Determinant of the product of adjacent matrices is product of determinants of the matrices and equating it to zero yields eigenvalues. All these graphs have same number of edges and vertices. The eigen values then are of the form sqrt(q^(2-2a)+q^(2a)+2q) were s=a+ib for each of the regular graphs. The RHS of Ihara Zeta Function can be written as [(1+1/q^s)(1-1/q^s)]^[V-E] and the product gives the RZF and the other series mentioned above. Eigen values can be atmost q.

[If an eigen value is t (<= q), then it can be derived that
$$q^s = q + (or) - \sqrt{q^2 - 4t} / 2$$
where s=a+ib. Setting a=0.5 is creating a contradiction apparently which is above divided by sqrt(q) (while equating real and imaginary parts for q^(ib) or e^(ib*logq)- needs to be verified if this kind of derivation is allowed in meromorphic functions).]

Above and the Informal notes in https://sites.google.com/site/kuja27/RamanujanGraphsRiemannZetaFunctionAndIharaZetaFunction.pdf?attredirects=0&d=1 can further be simplified as follows to get RZF in terms of IZF identity.

Disclaimer: It is not in anyway an attempted proof or disproof of RH as I am not an expert in analytical number theory. Hence arguments might be elementary. Following was found serendipitously as a surprise while working on special case of circuits for complement function for primes and it does not have direct relation to complementation - Fourier polynomial for Complement Function generalizes Riemann Hypothesis in a sense. Moreover, Notion of complementing a function is absent in mathematical literature I have searched so far except in mathematical logic. Following are derived based on Ihara Identity of Ihara Zeta Function.[https://lucatrevisan.wordpress.com/2014/08/18/the-riemann-hypothesis-for-graphs/#more-2824] for a prime+1 regular graph.

(1) For some i-th prime+1 regular graph (i.e q+1 regular graph where qi is prime),

$$\frac{[1+qi^{-s}]}{[zi(s)\det(I-A*qi^{-s} + qi^{(1-2s)}*I]^{(1/|V|-|E|)}} = \frac{1}{[1-qi^{-s}]}$$

(2) Infinite product of the above terms for all prime+1 regular graphs gives the RZF in right in terms of
Ihara Zeta Function Identities product on the left - The infinite set of prime+1 regular graphs relate to
RZF zeros.

(3) For non-trivial zeros s=a+ib,RZF in RHS is zero and thus either numerator product is zero or denominator product tends to infinity

(4) From Handshake Lemma, it can be derived that a q-regular graph with order

n(=|V| vertices) has
q*n/2 edges (=|E| edges)

(5) If numerator is set to zero in LHS,
q^(a+ib) = -1
and
cos(blogq) + isin(blogq) = -1/q^a which seems to give further contradiction

(6) If denominator is set to infinity in LHS,
[z1*z2*.....det()det()....]^1/|V|-|E| = Inf
(or)
[z1*z2*.....det()det()....]^1/|E|-|V| = 0

for some term in the infinite product of LHS which implies that
[z1*z2*.....det()det()....] = 0 which is described earlier above in the notes.

(7) More derivations of the above are in the notes uploaded in handwritten preliminary drafts at:
(7.1) https://sites.google.com/site/kuja27/RZFAndIZF_25October2014.pdf?attredirects=0&d=1
(7.2) http://sourceforge.net/p/asfer/code/HEAD/tree/python-src/ComplFunction_DHF_PVsNP_Misc_Notes.pdf

(8) By equating the determinant in (6) above to zero, written notes in (7) derive values of s=a+ib for non-Ramanujan prime+1 regular graphs for the expressions in points (1) to (6) above. They seem to suggest that RH is true with elementary complex arithmetic without using any complex analysis, with or without any assumption on the eigenvalue surprisingly - assuming q^s + q^(1-s) = v (v can be set to any eigenvalue - can be atmost q+1 for the q+1-regular non-ramanujan graph) and solving for s=a+ib - gives a=1/2 and thus for both ramanujan and non-ramanujan graphs as eigen value becomes irrelevant - needs lot of reviewing - because it implies that graph formulation above of Riemann Hypothesis is true for all prime+1 regular graphs by applying Ihara Identity of Ihara Zeta Function and thus Riemann Hypothesis is true. The choice of prime+1 regularity is just a contrived gadget to equate to Riemann Zeta Function through an infinite product. Crucial fact is the independence of eigen value in the previous derivation whenever the graph regularity is prime+1, thus directly connecting prime numbers and real part of Riemann Zeta Function non-trivial zero.

(9) Above doesn't look circular also because infinite product of characteristic polynomials - determinants - are independent of infinite product of Ihara Zeta Functions preceding them in denominator - this happens only when the product z1*z2*... is not zero i.e when the graphs are non-ramanujan. The infinite product of determinants of the form det(I-A(qi)^(-s)+(qi)^(1-2s)I) when solved for zero do not depend on eigenvalue and s is assumed nowhere. Independence of eigen value implies all possible graphs. Further the last step is independent of regularity q too. Had it been circular, the eigenvalue for Ramanujan graph should have occurred somewhere in the derivation throwing back to square one.

(10) Expression in (1) for a prime+1 regular graph can also be equated to Euler-Fourier polynomial mentioned in (24c) per-prime term for each prime+1 regular graph - LHS is a graph while RHS is a fourier polynomial for boolean circuit for a prime - P(xi). Thus pattern in prime polynomials in RHS are related to patterns in graphs on left:

$$\frac{[1+qi^{-s}]}{[zi(s)\det(I-A*qi^{-s}+qi^{(1-2s)}*I]^{(1/|V|-|E|)}} = \frac{1}{[1-P(xi)^{-s}]}$$

(11) qi can be replaced with Fourier polynomial P(xi),and the above becomes:
[1-P(xi)^-2s] = [zi(s)det(I-A*P(xi)^-s + P(xi)^(1-2s)*I]^(1/|V|-|E|)

(or) [1-P(xi)^-2s]^(|V|-|E|) = [zi(s)det(I-A*P(xi)^-s + P(xi)^(1-2s)*I]

(12) LHS of (11) can be expanded with binomial series. Thus Euler-Fourier polynomial is coalesced into Ihara identity to give Euler-Fourier-Ihara polynomial for a prime(and hence corresponding prime+1 regular graph). Partial Derivatives of the above polynomial look crucial in deciphering pattern in primes - for example doe(s)/doe(P(xi)).

(13) ACC circuits have support for mod(m) gates. Thus a trivial circuit for non-primality is set of mod(i) circuits - 1,2,3,...,sqrt(N) - that output 1 to an OR gate up (factor gates output 1). This non-primality circuit is equivalent to complement of complement function circuit for xy=z described previously (and it can be stated in its dual form also).

(14) The Fourier polynomial of a prime P(xi) is holographic in the sense that it has information of all primes due to the multiplexor construction.

(15) Without any assumption on Ihara and Riemann Zeta Functions, for any (q+1)-regular graph for prime q, just solving for eigenvalue in det[-[A - I*(q^s + q^(1-s))]] to get Real(s) = 0.5 looks like an independent identity in itself where A is adj matrix of graph. It neither requires Riemann Zeta Function nor Ihara Zeta Function to arrive at Real(s)=0.5.

(16) In (15), even the fact that q has to be prime is redundant. Just solving for q^s + q^(1-s) = some_eigen_value gives Real(s)=0.5. In such a scenario what the set {Imaginary(s)} contains is quite non-trivial. It need not be same as {Imaginary(RZF_zero)}.

(17) Assuming Real(s)=0.5 from 7.1 and 7.2, it can be derived with a little more steps that eigen_value = 2*sqrt(q)*cos(b*log(q)) - eigen value depends only on imaginary(s) and regularity.

(18) It is not known if {Imaginary(RZF_zero)} = {Imaginary(s)}. If not equal this presents a totally different problem than RZF and could be a disjoint_set/overlap/superset of RZF zeroes.

(19) Maximum eigen value of (q+1)-regular graph is (q+1) and the infinite set {Imaginary(s)} can be derived from eigen_value=2*sqrt(q)*cos(b*log(q)) in (17) for infinite set of (q+1)-regular graphs.

(20) An experimental python function to iterate through all {Imaginary(s)} mentioned in (19) supra has been included in complement.py. Because of the restriction that cos() is in [-1,1], eigen_value <= 2*sqrt(q) which has a trivial value of q=1 when eigen_value=q+1.

(21) (SOME EXPERIMENTATION ON DISTRIBUTION OF PRIMES) List of primes read by complement.py is downloaded from https://primes.utm.edu/lists/. IharaIdentity() function in complement.py evaluates Imaginary(s)=b=arccos(v/(2*sqrt(q)))/log(q) by incrementing v by small steps in a loop till it equals 2*sqrt(q). This allows v to be in the range [0,2*sqrt(q)] whereas Ramanujan graphs require it to be 2*sqrt(q-1) or q. Hence non-ramanujan graphs are also allowed. Logs in testlogs/ print all Imaginary(s) iterations of this eigenvalues for all 10000 primes. arccos() function returns radians which is a cyclic measure and hence can take x+2*y*pi where y=0,1,2,3,4,5,6,... which could be arbitrarily large infinite set. Logs print only x with y=0 and not all cycles. Prima facie visual comparison shows this set to be different from Imaginary(RZF) for x alone which could be inaccurate. Sifting through all cycles and verifying if these are indeed Im() parts of RZF might be an arduous task and could be undecidable too because two infinite sets have to be compared for equality. But theoretically the Re() part of (2) which is infinite product of (1) equivalent to Riemann Zeta Function is 0.5 for all while Im() part is computationally intensive. Crucial evidence that comes out of it is the possibility of cyclic values of radians in b=arccos(v/(2*sqrt(q))/log(q) which implies a fixed (prime, eigenvalue) ordered tuple correspond to infinitely many b(s) (Im() parts). Thus

2*sqrt(q)*cos(b*log(q)) = eigen_value is a generating function mining pattern in distribution of primes. Rephrasing, b has generating function: b <= (x + 2*y*pi)/logq, where 0 <= x <= pi/2. SequenceMining.py also has been applied to binary representation of first 10000 prime numbers (Commit Notes 273 below) which is learning theory way of finding patterns in distribution in primes. Logs for the most frequent sequences in prime strings in binary have been committed in testlogs/. These binary sequences mined from first 10000 primes have been plotted in decimal with R+rpy2 function plotter. Function plot in decimal shows sinusoidal patterns in mined sequences with periodic peaks and dips as the length of sequence increases i.e the mined sequences in prime binary strings periodically have leftmost bits set to 1 and rightmost bits set to 1 and viceversa which causes decimals to vacillate significantly. Primes are not regular and context-free languages which are known from formal languages theory. The function doing complementation in complement.py can be translated to Turing Machine by programming languages Brainfuck and Laconic which is equivalent to a lambda function for complement.

24h. PAC Learning and Complement Function Construction

Complement Boolean Function construction described in http://arxiv.org/abs/1106.4102 is in a way an example of PAC learnt Boolean Function - it is rather C Learnt (Correct Learning) because there is no probability or approximation. Complement Boolean Function can be PAC learnt (with upperbounded error) as follows:

- There are two datasets - dataset1 of size 2^n of consecutive integers and dataset2 of first n-bit prime numbers of size 2^n
- Each element of dataset1 is mapped to i-th bit of the corresponding prime number element in the dataset2. Boolean Conjunction is learnt for each of the i mappings (PAC Learning Algorithm: http://www.cis.temple.edu/~giorgio/cis587/readings/pac.html).
- Above step gives n probabilistic, approximate, correct learnt boolean conjunctions for each bit of all the 2^n prime numbers.
- An example PAC Boolean Conjunction Learner is at: http://sourceforge.net/p/asfer/code/HEAD/tree/python-src/PACLearning.py

---
24i. Star Complexity and Complement Function Circuit Lowerbound (related to 198)

Star Complexity of a Boolean Circuit is the minimum number of AND and OR gates required in monotone circuit graph. Size lowerbound for Complement Function circuit can thus be lowerbounded by Strong Magnification Lemma (Stasys Jukna - http://www.thi.informatik.uni-frankfurt.de/~jukna/ftp/graph-compl.pdf) - Size(ComplementCircuit) >= Star(ComplementCircuitGraph) - (2 + o(1))*n . Mapping from Boolean Circuit to Graph is done through a bipartite graph gadget wherein a boolean variable x(v1,v2) iff there is an edge from v1 to v2 in the bipartite graph and replacing each boolean literal by an OR of two new variables. Star Complexity views Boolean Circuits as a graph and for most graphs it is Omega(n^2/logn). Obtaining Size Lowerbounds for arbitrary complement functions is non-trivial.

24j. Additional references:

24.1 Google groups thread reference 2003 (OP done by self): https://groups.google.com/forum/#!search/ka_shrinivaasan%7Csort:relevance%7Cspell:false/sci.math/RqsDNc6SBdk/Lgc0wLiFhTMJ
24.2 Math StackExchange thread 2013: http://math.stackexchange.com/questions/293383/complement-of-a-function-f-2n-n-in-mathbbn-0-n-rightarrow-n1
24.3 Theory of Negation - http://mentalmodels.princeton.edu/skhemlani/portfolio/negation-theory/ and a quoted excerpt from it :

"... The principle of negative meaning: negation is a function that takes a single argument, which is a set of fully explicit models of possibilities, and in its core meaning this function returns the complement of the set..."
24.4 Boolean Complementation [0 or 1 as range and domain] is a special case of Complement Function above (DeMorgan theorem - http://www.ctp.bilkent.edu.tr/~yavuz/BOOLEEAN.html)
24.5 Interpolation - http://caig.cs.nctu.edu.tw/course/NM07S/slides/chap3_1.pdf
24.6 Formal Concept Analysis - http://en.wikipedia.org/wiki/Formal_concept_analysis, http://ijcai.org/papers11/Papers/IJCAI11-227.pdf
24.7 Prime generating polynomials - http://en.wikipedia.org/wiki/Formula_for_primes
24.8 Patterns in primes - every even number > 2 is sum of 2 primes - Goldbach conjecture : http://en.wikipedia.org/wiki/Goldbach%27s_conjecture
24.9 Arbitrarily Long Arithmetic progressions - Green-Tao theorem: http://en.wikipedia.org/wiki/Green%E2%80%93Tao_theorem
24.10 Prime generating functions - https://www.sonoma.edu/math/colloq/primes_sonoma_state_9_24_08.pdf
24.11 Hardness of Minimizing and Learning DNF Expressions - https://cs.nyu.edu/~khot/papers/minDNF.pdf
24.12 DNF Minimization - http://users.cms.caltech.edu/~umans/papers/BU07.pdf
24.13 DNF Minimization - http://www.cs.toronto.edu/~toni/Papers/mindnf.pdf
24.14 Riemann Zeta Function Hypothesis - all non-trivial(complex) zeroes of RZF have Re(z) = 0.5 - http://en.wikipedia.org/wiki/Riemann_hypothesis.
24.15 Frequent Subgraph Mining - http://research.microsoft.com/pubs/173864/icde08gsearch.pdf
24.16 Frequent Subgraph Mining - http://glaros.dtc.umn.edu/gkhome/fetch/papers/sigramDMKD05.pdf
24.17 Roots of polynomials - Beauty of Roots - http://www.math.ucr.edu/home/baez/roots/
24.18 Circuit lowerbounds (Gate Elimination, Nechiporuk, Krapchenko, etc) - http://cs.brown.edu/~jes/book/pdfs/ModelsOfComputation_Chapter9.pdf
24.19 Schwartz-Zippel Lemma for Polynomial Identity Testing - http://en.wikipedia.org/wiki/Schwartz%E2%80%93Zippel_lemma
24.20 Schwartz-Zippel Lemma for PIT of multilinear polynomials - http://www.cs.huji.ac.il/~noam/degree.ps
24.21 Analysis of Boolean Functions - http://analysisofbooleanfunctions.org/
24.22 Riemann-Siegel Formula for computation of zeros - http://numbers.computation.free.fr/Constants/Miscellaneous/zetaevaluations.html
24.23 RZF zeros computation - http://math.stackexchange.com/questions/134362/calculating-the-zeroes-of-the-riemann-zeta-function
24.24 Online Encyclopedia of Integer Sequences for Prime numbers - all theorems and articles related to primes - https://oeis.org/search?q=2%2C3%2C5%2C7%2C11%2C13%2C17%2C19%2C23%2C29%2C31%2C37%2C41%2C43%2C47%2C&language=english&go=Search
24.25 Intuitive proof of Schwartz-Zippel lemma - http://rjlipton.wordpress.com/2009/11/30/the-curious-history-of-the-schwartz-zippel-lemma/
24.26 Random Matrices and RZF zeros - http://www.maths.bris.ac.uk/~majpk/papers/67.pdf
24.27 Circuit for determinant - S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. Inf. Prod. Letters 18, pp. 147–150, 1984.
24.28 Mangoldt Function, Ihara Zeta Function, Ramanujan Graphs - http://lucatrevisan.wordpress.com/2014/08/18/the-riemann-hypothesis-for-graphs/#more-2824
24.29 Multiplicity of an eigen value in k-regular graph - http://math.stackexchange.com/questions/255334/the-number-of-connected-components-of-a-k-regular-graph-equals-the-multiplicit
24.30 PAC Learning - http://www.cis.temple.edu/~giorgio/cis587/readings/pac.html
24.31 Pattern in Prime Digits - [Oliver-Kannan Soundarrajan] - http://arxiv.org/abs/1603.03720 - Prime numbers which are juxtaposed avoid ending in same digit. This has direct bearing on Fourier polynomial of prime complement function and

Euler-Fourier polynomial derived above which is binary representation of a prime and a special case of function complementation. If decimal representation of adjacent primes is repulsive in last digit, then last binary bits (LSB) output by the complement circuit should be too.
24.32 Pattern in Prime Digits - http://www.nature.com/news/peculiar-pattern-found-in-random-prime-numbers-1.19550 - above with assumption of Hardy-Littlewood k-tuple conjecture a generalization of twin primes conjecture to all prime constellations.
24.33 Pattern in Primes - Ulam's Spiral - [Stainslaw Ulam] - https://www.alpertron.com.ar/ULAM.HTM - Prime numbers are clustered along diagonals of anticlockwise spiral of integers 1,2,3,4,5,6,... ad infinitum.
24.34 Theory of Negation (Broken URL in 24.3 updated) - http://mentalmodels.princeton.edu/papers/2012negation.pdf
24.35 Prime Number Theorem - n-th prime is ~ O(nlogn)
24.36 Riemann Hypothesis prediction of prime distribution - Integral_2_p(n)_[dt/logt] = n + O(sqrt(n(logn)^3)
24.37 Brainfuck Turing Machine Compiler - https://esolangs.org/wiki/Brainfuck
24.38 Laconic Turing Machine Compiler - https://esolangs.org/wiki/Laconic
24.39 Circuit Complexity Lowerbound for Explicit Boolean Functions - http://logic.pdmi.ras.ru/~kulikov/papers/2011_3n_lower_bound_mfcs.pdf - 3n - o(n)
24.40 Star Complexity of Boolean Function Circuit - [Stasys Jukna] - http://www.thi.informatik.uni-frankfurt.de/~jukna/ftp/graph-compl.pdf
24.41 Random Matrices, RZF zeroes and Quantum Mechanics - https://www.ias.edu/ideas/2013/primes-random-matrices
24.42 Jones-Sato-Wada-Wiens Theorem - Polynomial of 25 degree-26 variables for Prime Diophantine set - http://www.math.ualberta.ca/~wiens/home%20page/pubs/diophantine.pdf
24.43 Energy levels of Erbium Nuclei and zeros of Riemann Zeta Function - http://seedmagazine.com/content/article/prime_numbers_get_hitched/

##
##

25. (FEATURE - DONE) Approximating with some probability distribution - Gaussian, Binomial etc., that model the probability of occurence of a datapoint in the set. Kullback-Leibler Divergence python implementation which computes distance in terms of amount of bits between two probability distribution has been added to python-src/. Minimum of the distance for different standard distributions with a dataset is the closest distribution approximation for the dataset.

(FEATURE - DONE) 26. Streaming algorithms - Finding Frequency moments, Heavy Hitters(most prominent items), Distinct Elements etc., in the numerical dataset. Usually numerical data occur in streams making these best choice for mining numerical data.
(FEATURE - DONE) 26.1 Implementation of LogLog and HyperLogLog Counter(cardinality - distinct elements in streamed multiset), CountMinSketch-CountMeanMinSketch (Frequencies and heavy hitters) and Bloom Filters(membership)
(FEATURE - DONE) 26.2 Parser and non-text file Storage framework for Realtime Streaming Data - Hive, HBase, Cassandra, Spark and Pig Scripts and Clients for Storage Backends have been implemented. The Storage is abstracted by a generator - architecture diagram at: http://sourceforge.net/p/asfer/code/HEAD/tree/asfer-docs/BigDataStorageAbstractionInAsFer.jpg
(FEATURE - DONE) 26.3 Python scripts for Stock Quotes Data and Twitter tweets stream search data for a query.

References:

26.4 https://gist.github.com/debasishg/8172796

27. (FEATURE - DONE) Usual Probabilistic Measures of Mean, Median, Curve fitting on the numeric data. Python+RPy2+R implementation wrapper has been added to

repository (python-src/Norms_and_Basic_Statistics.py)

28. (FEATURE - DONE - using python, R+rpy2) Application of Discrete Fourier Transform(using R), LOESS(using R), Linear Polynomial Approximate Interpolation(using R), Logistic Regression and Gradient Descent

29. (FEATURE - DONE) Least Squares Method on datapoints y(i)s for some x(i)s such that f(x)~y needs to be found. Computation of L0, L1 and L2 norms. Python+RPy2+R implementation wrapper has been added to repository (python-src/Norms_and_Basic_Statistics.py)

(FEATURE - DONE)30. K-Means and kNN Clustering(if necessary and some training data is available) - unsupervised and supervised clustering based on coordinates of the numerical dataset

31. (FEATURE - DONE) Discrete Hyperbolic Factorization - Author has been working on a factorization algorithm with elementary proof based on discretization of a hyperbola since 2000 and there seems to be some headway recently in 2013. To confirm the polylog correctness, a minimal implementation of Discrete Hyperbolic Factorization (and could be even less than polylog due to a weird upperbound obtained using stirling formula) has been added to AstroInfer repository at: http://sourceforge.net/p/asfer/code/HEAD/tree/cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound.cpp with factors output logs.

32. (FEATURE - DONE) Multiple versions of Discrete Hyperbolic Factorization algorithms have been uploaded as drafts in https://sites.google.com/site/kuja27:
32.1) http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicFactorization_UpperboundDerivedWithStirlingFormula_2013-09-10.pdf/download and
32.2) http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_updated_rectangular_interpolation_search_and_StirlingFormula_Upperbound.pdf/download(and multiple versions due to various possible algorithms for search and upperbound technique used)

33. (DONE) NC PRAM version of Discrete Hyperbolic Factorization:
33.1) An updated NC PRAM version of Discrete Hyperbolic Factorization has been uploaded at: http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound.pdf/download that does PRAM k-merge of discrete tiles in logarithmic time before binary search on merged tile.

33.2) Preliminary Design notes for CRCW PRAM implementation of the above is added to repository at: http://sourceforge.net/p/asfer/code/237/tree/ImplementationDesignNotesForDiscreteHyperbolicFactorizationInPRAM.jpg. This adds a tile_id to each tile so that during binary search, the factors are correctly found since coordinate info gets shuffled after k-tile merge.

34. (FEATURE - DONE) An updated draft version of PRAM NC algorithm for Discrete Hyperbolic Factorization has been uploaded - with a new section for Parallel RAM to NC reduction, disambiguation on input size (N and not logN is the input size for ANSV algorithm and yet is in NC - in NC2 to be exact - (logN)^2 time and polynomial in N PRAM processors - NC circuit depth translates to PRAM time and NC circuit size to number of PRAMs):
34.1 LaTeX - http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.tex/download

34.2 PDF -

http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.pdf/download

---
Additional references:

34.3 Above PRAM k-merge algorithm implies Factorization is in NC, a far more audacious claim than Factorization in P. It has been disputed if PRAM is indeed in NC because of input size being N and not logN. But there have been insurmountable evidences so far which all point to PRAM model being equivalent to NC circuits in certain conditions and NC circuit nodes can simulate PRAM with polylog upperbound on number of bits [HooverGreenlawRuzzo]- https://homes.cs.washington.edu/~ruzzo/papers/limits.pdf . References for this are in LaTex and PDF links previously and also mentioned below in "Additional References"

34.4) (IMPORTANT OPTIMIZATION OVER 34.1, 34.2 ABOVE) Instead of [BerkmanSchieberVishkin] an O(logloglogN) algorithm can be used described in (since the numbers in tiles are within a known range - 1 to N - for factorization of N in discretized tesselated hyperbolic arc) - Triply-Logarithmic Parallel Upper and Lower Bounds for Minimum and Range Minima over Small Domains (Omer Berkman, Yossi Matias, Prabhakar Ragde) - 1998 - http://www.sciencedirect.com/science/article/pii/S0196677497909056. This algorithm internally applies [BerkmanSchieberVishkin] but lemmas 2.3 and 3.1 mentioned in [BerkmanMatiasPrabhakar] preprocess the input within a known domain [1..N]. This takes O(logloglogN) time using (logN)^3/logloglogN processors for each merging problem and O(logloglogN) time using N/logloglogN processors overall. This algorithm can therefore supersede 34.1 and 34.2.

34.5) Randomized Algorithms ,Rajeev Motwani and Prabhakar Raghavan, Chapter 12 on Distributed and Parallel Algorithms, also describes input size and randomized algorithms for Parallel sort - Definition 12.1 of NC (Page 336) - "NC consists of languages that have PRAM algorithms with O(logN) time and O(N^k) PRAM processors".

34.6) Also an alternative merge algorithm in constant depth polysize circuit described in Chandra-Stockmeyer-Vishkin [http://cstheory.com/stockmeyer@sbcglobal.net/csv.pdf] can be applied (but it is for merging two lists of m, m-bit numbers where as the above factorization needs merging two lists of O(m), log(m)-bit numbers)

34.7) [RichardKarp-VijayaRamachandran] define the inclusion of PRAM models in NC in page 29 of http://digitalassets.lib.berkeley.edu/techreports/ucb/text/CSD-88-408.pdf - NCk in EREWk in CREWk in CRCWk=ACk in NC(k+1). [KarpRamachandran] cite [HooverKlawePippenger] - Bounding Fanout in Logical Networks - http://dl.acm.org/citation.cfm?id=322412 - for this inclusion. Thus references for PRAM=NC imply All Nearest Smaller Values (ANSV) CRCW PRAM algorithm [BerkmanSchieberVishkin] is also in NC counterintuitively despite the input size being N=n (and not logN).

34.8) Parallel RAM survey - http://www.cs.utoronto.ca/~faith/PRAMsurvey.ps

34.9) Related: Quantum Search using Grover Algorithm over unsorted lists can be done in O(sqrt(N)) - https://en.wikipedia.org/wiki/Grover's_algorithm. This is another counterintuitive fact that a quantum search on unsorted lists is slower than ANSV Parallel RAM algorithm.

34.10) Recent Advances in All Nearest Smaller Values algorithms:
34.10.1) ANSV in hypercube - Kravets and Plaxton - http://www.cs.utexas.edu/~plaxton/pubs/1996/ieee_tpds.ps
34.10.2) ANSV lowerbound - Katajainen - http://www.diku.dk/~jyrki/Paper/CATS96.ps - omega(n) processors with omega(logn)

time

34.10.3) ANSV in BSP machines - Chun Hsi Huang - http://www.cse.buffalo.edu/tech-reports/2001-06.ps

34.11) The crucial fact in the above is not the practicality of having order of n parallel processors with RAM to get the logarithmic time lowerbound (which looks costly in number of PRAMs wise), but the equivalence of PRAM to NC which is theoretically allowed despite input size being n instead of logn (because each PRAM cell mapped to a circuit element can have polylogn bits and polyn such PRAMs are allowed) which is sufficient for Discrete Hyperbolic Factorization to be in NC.

34.12) Rsync - PhD thesis - chapters on external and internal sorting - https://www.samba.org/~tridge/phd_thesis.pdf

34.13) Handbook of Parallel Computing - [SanguthevarRajasekaran-JohnReif] - http://www.engr.uconn.edu/~rajasek/HandbookParallelComp.pdf, https://books.google.co.in/books?id=OF9hk4oC6FIC&pg=PA179&lpg=PA179&dq=PRAM+NC+equivalence&source=bl&ots=LpYceSocLO&sig=GtslRh1I1AveOLo0kTylSzyDd48&hl=en&sa=X&ved=0ahUKEwi_1OfmuKbJAhUCHY4KHTpdAfoQ6AEISTAI#v=onepage&q=PRAM%20NC%20equivalence&f=false

34.14) [Eric Allender] - NC^1 and EREW PRAM - March 1990 - https://groups.google.com/forum/#!topic/comp.theory/0a5Y_DSkOao - "... Since NC^1 is contained in DLOG, and many people suspect that the containment is proper, it seems unlikely that NC^1 corresponds to log time on an EREW PRAM ..." - contradicts 34.7.

34.15) Efficient and Highly Parallel Computation - [JeffreyFinkelstein] - https://cs-people.bu.edu/jeffreyf/static/pdf/parallel.pdf - "... NC represents the class of languages decidable by a CREW PRAM with a polynomial number of processors running in polylogarithmic parallel time. Languages in NC are considered highly parallel ...". [Berkman-Schieber-Vishkin] algorithm - www.umiacs.umd.edu/users/vishkin/PUBLICATIONS/ansv.ps - for ANSV is for both CREW PRAM of O(logn, n/logn) and CRCW PRAM of O(loglogn, n/loglogn) and thus former is in NC.

34.16) There is a commercially available Parallel CRCW RAM chip implementation by NVIDIA CUDA GPUs - http://developer.nvidia.com/cuda and XMT research project - www.umiacs.umd.edu/users/vishkin/XMT/ but not for CREW PRAM which is a limitation of implementing All Nearest Smaller Values PRAM merge for discretized hyperbolic arc and doing a benchmark.

34.17) StackExchange thread on Consequences of Factorization in P - http://cstheory.stackexchange.com/questions/5096/consequences-of-factoring-being-in-p . Factorization in P is unlikely to have any significant effect on existing class containments, though practical ecommerce becomes less secure.

34.18) What happened to PRAM - http://blog.computationalcomplexity.org/2005/04/what-happened-to-pram.html - Quoted excerpts from comments - "... I don't understand why some CS theory people apologize for the PRAM. NC is robust and interesting as a complexity class, and an easy way to show that a problem is in NC is to give a PRAM algorithm. That's all the argument I need for the PRAM's existence. And yes, I've heard all of the arguments about why the PRAM is completely unrealistic ..."

34.19) [Page 29 - HooverGreenlawRuzzo] - "... but there is no a priori upper bound on the fanout of a gate. Hoover, Klawe, and Pippenger show that conversion to bounded fanout entails at most a constant factor increase in either size or depth [160]..."

34.20) (DONE-BITONIC SORT IMPLEMENTATION) In the absence of PRAM implementation, NC Bitonic Sort has been invoked as a suitable alternative in

parallel tile merge sort step of Parallel Discrete Hyperbolic Factorization Implementation. Commit Notes 261-264 and 265-271 below have details on this. Bitonic Sort on SparkCloud requires O((logn)^2) time with O(n^2logn) parallel comparators (which simulate PRAM but comparators required are more than PRAMs). With this NC factorization has been implemented on a Spark cloud.

34.21) An important point to note in above is that an exhaustive search in parallel would always find a factor, but in how many steps is the question answered by NC computational geometric search. This is 1-dimensional geometric factorization counterpart of 2-dimensional Graham's Scan and Jarvis March for Convex Hull of n points. A parallel algorithm in O(n) steps wouldn't have qualified to be in NC. Above NC algorithm scans just an one-dimensional tesselated merged tiles of a hyperbolic arc in O((logn)^2) for merge + O(logn) for search and doesn't scan more than 1-dimension. Parallel tesselation of O(n) long hyperbolic arc to create locally sorted tile segments requires < O(logn) steps only if number of processors is > O(n/logn). Hence it adheres to all definitions of NC. Thus total parallel work time is O((logn)^2) + O(logn) + O(logn) = O((logn)^2). Here again N=n.

34.22) NC Computational Geometric algorithms - [AggarwalChazelleGuibasDunlaingYap] - https://www.cs.princeton.edu/~chazelle/pubs/ParallelCompGeom.pdf

34.23) Parallel Computational Geometry Techniques - [MikhailAtallah] - http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1021&context=cstech - Section 3.1 - Sorting and Merging - "...best hypercube bound for Parallel Sorting is O(logn(loglogn)^2) - [CypherPlaxton] - http://dl.acm.org/citation.cfm?id=100240 ..."

34.24) [DexterKozen-CheeYap] - Cell Decomposition is in NC - http://anothersample.net/order/fe1d53539cff07d3e836ccc499d443b38b2848ba - This is a deep Algebraic Geometry/Topology result for Cell Decomposition by constructing NC circuit for it. Coincidentally, the connection between NC factorization and geometry conjectured in 35 below is already present as evidenced by this. This algorithm is for Cell decomposition of a manifold in arbitrary dimensions - a set of disjoint union of cells created by intersecting polynomials. For NC factorization, the polynomial of interest is hyperbola. Illustration in [Kozen-Yap] - page 517 - is for 0,1,2-dimension cells with 2 polynomials - parabola and circle. This corresponds to tesselation step of factorization where a continuous hyperbola is discretized into disjoint union of cells.

34.25) Cell decomposition for tesselation of hyperbola can be created in two ways. In the first example, set of polynomials are {hyperbola, stepfunction1, stepfunction2, y=k forall integer k}. Geometrically the hyperbola is bounded above and below by 2 step functions i.e hyperbola intersects a grid of x-y axes lines. This creates 2-dimensional cells above and below hyperbola ensconced between 2 step functions which are homeomorphic to R^2 (there is a bijective map between cells and R^2 - Topology - James Munkres}. Each cell has same sign for a polynomial {above=+1, on=0, below=-1}. This cell decomposition is in NC. Cells above and below hyperbola have to be merged to get squared tesselation.

34.26) In another example Cell decomposition is created by intersection of integer y-axis lines and hyperbola which results in set of 0-cells (discrete set of points on hyperbola).

34.27) NVIDIA CUDA Parallel Bitonic Sort Implementation Reference - http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/Data-Parallel_Algorithms.html, Linux code - http://developer.download.nvidia.com/compute/cuda/1.1-Beta/Projects/bitonic.tar.gz

34.28) Tiling of Hyperbolic curve: The discretization step of a

hyperbola to form set of contiguous tiled segments has been mentioned as "Tesselation" throughout this document which could be a misnomer. Precise equivalent of this tiling is pixelation in Computer Graphics where an image bitmap is downsampled to create a low-resolution approximation of image - in this example, an image of hyperbolic curve is pixelated to g et a tiled set of segments (http://demonstrations.wolfram.com/PixelizationOfAFont/ shows how a letter A is pixelated to create stylized pixelated A).

34.29) The tiling of hyperbolic curve is done by deltax = N/[y*(y+1)] which is a process similar to low-pass filter or Box Blur in Graphics. In a parallel setting with PRAMs or Parallel Bitonic Sort, the preprocessing step required is the tiled hyperbolic arc already in place spread out across all nodes of total number O(N/logN). This naive tiling though not as sophisticated as Box Blur Gaussian Filter, needs time O(logN) for each node i.e delta computation per coordinate is O(1) and each node is allotted O(logN) long arc segment and thus O(1*logN) per node.

34.30) Above algorithm ignores Communication Complexity across PRAMs or Comparator nodes on a Cloud.

34.31) Comparison of Parallel Sorting Algorithms (Bitonic, Parallel QuickSort on NVIDIA CUDA GPU etc.,) - http://arxiv.org/pdf/1511.03404.pdf. Bitonic Sort has the best parallel performance in most usecases.

34.32) NC-PRAM Equivalence and Parallel Algorithms - [David Eppstein] - https://www.ics.uci.edu/~eppstein/pubs/EppGal-ICALP-89.pdf

34.33) Parallel Merge Sort - [Richard Cole - https://www.cs.nyu.edu/cole/] - best known theoretical parallel sorting algorithm - requires O(logn) time with n processors and thus in NC - https://www.semanticscholar.org/paper/Parallel-Merge-Sort-Cole/6b67df5d908993eca7c03a564b5dcb1c4c8db999/pdf

34.34) NC-PRAM equivalence - http://courses.csail.mit.edu/6.854/06/notes/n32-parallel.pdf

34.35) Theorem 13 - PRAM(polylog,log) = uniform-NC - www.wisdom.weizmann.ac.il/~oded/PS/CC/l27.ps - This mentions that Communication Complexity in PRAM model is assumed to be O(1) and thus negligible though practically PRAMs are unrealistic.

34.36) PRAM Implementation Techniques - http://www.doc.ic.ac.uk/~nd/surprise_95/journal/vol4/fcw/report.html - "... However to say that an algorithm is in NC does not mean that it can be efficiently implemented on a massively parallel system..."

34.37) Logarithm Time Cost Parallel Sorting - [Lasse Natvig] - Survey on Batcher's Bitonic Sort, AKS Sorting Network, Richard Cole's Parallel Sort - www.idi.ntnu.no/~lasse/publics/SC90.ps - Bitonic Sort is faster in practice though Richard Cole Parallel Sorting is the fastest and most processor-efficient known.

34.38) What is wrong with PRAM - "Too much importance is placed on NC . In particular, algorithms which have “fast” runtimes but use, for example, O(n^2) processors are simply of no use" - Section 3.2 - https://people.eecs.berkeley.edu/~jrs/meshpapers/SuThesis.pdf

34.39) NC-PRAM equivalence - https://www.ida.liu.se/~chrke55/courses/APP/ps/f2pram-2x2.pdf - "set of problems solvable on PRAM in polylogarithmic time O((logn)^k) k>0, using only n^O(1) processors (i. e. a polynomial number) in the size n of the input instance"

34.40) Number of PRAMs in NC definition - http://cs.stackexchange.com/questions/39721/given-a-pram-may-use-arbitrarily-

many-processors-why-is-hamiltonian-cycle-not-i

34.41) EURO-PAR 1995 - Input size in Parallel RAM algorithms - https://books.google.co.in/books?id=pVpjOwUHEigC&pg=PA245&lpg=PA245&dq=size+of+input+in+PRAM&source=bl&ots=uZ9GeONqtg&sig=6YHl4CvNUdFERPe8p192hdB1Kc0&hl=en&sa=X&ved=0ahUKEwiZ3_z20qTOAhVJGJQKHX8bALIQ6AEIOTAG#v=onepage&q=size%20of%20input%20in%20PRAM&f=false - "... Third, inputs are usually measured by their size. We use overall number of array elements ..."

34.42) Batcher's Bitonic Sort is in NC - https://web.cs.dal.ca/~arc/teaching/CS4125/Lectures/03b-ParallelAnalysis.pptx - also other
PRAM algorithms are in NC.

34.43) Brief Overview of Parallel Algorithms, Work-Time Efficiency and NC - [Blelloch] - http://www.cs.cmu.edu/~scandal/html-papers/short/short.html - "... Examples of problems in NC include sorting, finding minimum-cost spanning trees, and finding convex hulls ...". Work-Time Efficiency is more about optimizing number of processors required (with polylog time) and two algorithms with differing work-time are both theoretically in NC.

34.44) Sorting n integers is in NC - [BussCookGuptaRamachandran] - www.math.ucsd.edu/~sbuss/ResearchWeb/Boolean2/finalversion.ps

34.45) PRAM and Circuit Equivalence - [Savage] - http://cs.brown.edu/~jes/book/pdfs/ModelsOfComputation.pdf - Lemma 8.14.1

34.46) Efficient Parallel Computation = NC - [AroraBarak] - http://theory.cs.princeton.edu/complexity/book.pdf - Theorem 6.24 - Simpler version of 34.3

34.47) Parallel sorting and NC - http://courses.csail.mit.edu/6.854/06/notes/n32-parallel.pdf

34.48) Parallel sorting in PRAM model - http://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/MW87/MW87.pdf

34.49) Parallel Sorting is in NC - http://www.toves.org/books/distalg/distalg.pdf - Section 5 (Page 17)

34.50) Parallel K-Merge Algorithm for merging k sorted tiles into a single sorted list - LazyMerge - www.cs.newpaltz.edu/~lik/publications/Ahmad-Salah-IEEE-TPDS-2016.pdf

34.51) Timsort and its parallel implementation in Java, Python and Android for parallel merge sort of arrays - https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#parallelSort-int:A- . Timsort exploits order in unsorted lists. Java implementation is based on ForkJoinPool work-stealing pattern (similar to router-worker-dealer pattern in ZeroMQ) and parallelism is equal to number of processors.Cloud Bitonicsort SparkPython implementation in NeuronRain AsFer could be more time efficient than Arrays.parallelSort() in Java 8 as it implements Batcher Sort on Cloud.

34.52) Comparison of Parallel and Sequential Sorts - https://github.com/darkobozidar/sequential-vs-parallel-sort

34.53) Trigonometric Functions over Finite Galois Fields (of prime power order) - https://arxiv.org/pdf/1501.07502.pdf - Defines trigonometric functions (hyperbola is also a trigonometic function - class of hyperbolic functions) on discrete objects like a finite field yielding a function on a set of complex points(Gaussian integers). This abstracts and reduces to discretization step of hyperbolic factorization but requires prime power sized finite field.

34.54) Adaptive Bitonic Sorting - http://epubs.siam.org/doi/abs/10.1137/0218014 - very old paper (1989) on PRAM implementation of bitonic sort - NeuronRain AsFer implements bitonic sorting on Spark Cloud and parallelizes comparators on cloud (34.20) effectively treating cloud as sorting network with cloud communication complexity assumed as constant in average case.

34.55) Efficient Parallel Sorting - improvement of Cole's Parallel Sort and AKS sorting networks: sorting networks can be implemented in EREW PRAM - [Goodrich] - https://arxiv.org/pdf/1306.3000v1.pdf - [Godel's Lost Letter and P=NP - Richard Lipton - Galactic Sorting Networks - https://rjlipton.wordpress.com/2014/04/24/galactic-sortning-networks/]

34.56) Computational Pixelation Geometry Theorems - https://arxiv.org/pdf/1105.2831v1.pdf - defines planar pixelation of a continuous curve and proves Intermediate Value Theorem for Pixelation geometry. This is exactly the discretization step of this hyperbolic factorization algorithm. This formalizes the discretization mentioned in 34.29 where each pixelated tile is a set of intermediate values between f(x) and f(x+1) for interval [x,x+1] where f = N/x for N to be factorized.

34.57) Approximating a function graph by planar pixelation - http://www3.nd.edu/~lnicolae/Pixelations-beam.pdf - this applies advanced topology and geometry for pixelation of any function graph. Presently pixelation of hyperbola graph is done in very simple, primitive way as mentioned in 34.29.

34.58) Resource Oblivious Sorting on Multicores - [RichardCole-VijayaRamachandran] - https://arxiv.org/pdf/1508.01504v1.pdf - New parallel merge sort algorithm for multicore machines (SPMS - SamplePartitionMergeSort) with parallel execution time in O(logN*loglogN). Multicores are equivalent to Asynchronous PRAMs (each core executes asynchronously) and thus it is an NC sorting algorithm.

34.59) NC and various categories of PRAMs - https://pdfs.semanticscholar.org/5354/99cf0c2faaaa6df69529605c87b511bd2226.pdf - CRCW PRAM can be simulated by EREW PRAM with polylogarithmic increase in time.

34.60) PRAM-NC equivalence, definition of input size - ftp://ftp.cs.utexas.edu/pub/techreports/tr85-17.pdf - atom is an indivisible unit of bit string, input is sequence of atoms, input size is length of this sequence.

34.61) PRAM model and precise definition of input size in PRAM models - https://www.cs.fsu.edu/~engelen/courses/HPC-adv-2008/PRAM.pdf - input size is length of sequence n = 2^k ,not number of bits k=logn, and each element in the sequence is a set of bits - input size to a PRAM algorithm is exponential in number of bits and differs from sequential algorithms which have input size in just number of bits.

34.62) Simulation of Parallel RAMs by Circuits - http://www.cstheory.com/stockmeyer@sbcglobal.net/sv.pdf

34.63) PRAM Models and input size - [Joseph JaJa] - https://www.cs.utah.edu/~hari/teaching/bigdata/book92-JaJa-parallel.algorithms.intro.pdf - "...ALGORITHM 1.3 (Matrix Multiplication on the PRAM) Input: Two n x n matrices A and B stored in the shared memory, where n = 2^k . The initialized local variables are n, and the triple of indices (i, j, I) identifying the processor. ...", https://people.ksp.sk/~ppershing/data/skola/JaJa.pdf - Chapter 4 - Searching, Merging, Sorting

34.64) Input Size for PRAMs, NC-PRAM equivalence, Cook-Pippenger Thesis, Brent's Speedup Lemma - www.csl.mtu.edu/cs5311.ck/www/READING/pram.ps.gz - Section on Input/Output Convention in PRAMs and Parallel MergeSort in EREW PRAM - Input size to Parallel Merge Sort is the length of array of integers to be

sorted and is not logarithmic in length of array. This requires parallel time O((logN)^2) which is the depth of equivalent NC circuit. Computational Geometric Discrete Hyperbolic Factorization described above discretizes hyperbola into segments of sorted tiles of numbers on a 2 dimensional plane and merge-sorts them. This obviously requires O((logN)^2) time to find a factor with additional O(logN) binary search. Brent Speedup Lemma applies because, the tile segments can be statically allocated to each specific processor in tiling phase.

34.65) Pixelation of Polygons - On Guarding Orthogonal Polygons with Bounded Treewidth - [Therese Biedl and Saeed Mehrabi] - Page 160 - http://2017.cccg.ca/proceedings/CCCG2017.pdf - CCCG 2017, Ottawa, Ontario, July 26–28, 2017 - Computational Geometric definition of standard pixelation divides a polygon into rectangles and adjoining rectangles are vertices connected to form a planar pixelation graph. Similar notion of pixelation holds for pixelating hyperbola pq=N for finding factors p and q of integer N. Art Gallery problem in computational geometry finds minimum number of vantage points to guard the art gallery. If hyperbolic curve is considered as an art gallery, and guards have visibility only in vertical and horizontal directions, pixelation creates set of rectangles bounded by vantage points ensconcing the hyperbola. This illuminates the hyperbola completely.

34.66) A Comparison of Parallel Sorting Algorithms on Different Architectures- [NANCY M. AMATO, RAVISHANKAR IYER, SHARAD SUNDARESAN, YAN WU] - https://parasol.tamu.edu/publications/download.php?file_id=191

34.67) Bitonic Sort Implementation - [Amrutha Mullapudi] - https://www.cse.buffalo.edu//faculty/miller/Courses/CSE633/Mullapudi-Spring-2014-CSE633.pdf - input size is n=2^k and parallel execution time O(logn^2) for Batcher Bitonic Sort - Hyperbolic Pixelated Factorization is in NC because bitonic sort is in NC and is also optimal if O(logn) depth AKS sorting networks are used, because O(n) * O(logn) = O(nlogn) which is serial sorting time lowerbound.

34.68) PRAMs and Multicore architectures - http://blog.computationalcomplexity.org/2008/04/revenge-of-parallelism.html

34.69) NC and PRAMs - http://www.cse.iitd.ernet.in/~subodh/courses/CSL860/slides/3pram.pdf

34.70) Geometric Searching, Ray Shooting Queries - [Michael T.Goodrich] - http://www.ics.uci.edu/~goodrich/pubs/42.pdf - Section 42.5 - Ray Shooting is defined as searching a point location in set of line segments by shooting a ray from a point outside the set of line segments. It is an open problem to find an efficient data structure for parallel ray shooting which searches n points in parallel by shooting n light rays. Previous Hyperbolic Factorization can also be rephrased as parallel shooting problem which searches the factor points on the pixelated hyperbolic tile segments in parallel from an external point outside the hyperbola and answers the open question in affirmative.

34.71) Planar Point Location in Parallel - ftp://ftp.cs.brown.edu/pub/techreports/89/cs89-45a.pdf - Section 4 - Point Location problem is defined as searching a query point q in a set of line segments. Discretized/Pixelated Hyperbola has number of line segments in O(loglogN). Finding factors (p,q) such that pq=N is planar point location problem which searches for the factors in the tiled hyperbolic line segments.Point Location can be done in parallel by PRAM.

34.72) One dimensional Point Location - https://www.csun.edu/~ctoth/Handbook/chap38.pdf - Section 38.1 - List searching - Tiled hyperbolic line segments are one dimensional intervals and number of intervals are O(loglogN). Searching this list amounts to finding a factor of N.

34.73) Hoover-Klawe-Pippenger Algorithm for converting arbitrary circuit to bounded fanout 2 - [Ogihara-Animesh] - http://www.cs.rochester.edu/u/ogihara/

research/DNA/cec.ps.gz

34.74) PRAM definition of NC - [Kristoffer Arnsfelt Hansen] - https://users-cs.au.dk/arnsfelt/CT08/scribenotes/lecture7.pdf

34.75) Factorization, Planar Point Location in parallel, Cascading, Polygon pixelation of hyperbola - [Mikhail Atallah] - https://pdfs.semanticscholar.org/1043/702e1d4cc71be46388cc12cd981ee5ad9cb4.pdf - Section 4.5 - Art Gallery
Pixelation of Hyperbola pq=N (described in 465) creates a polygon covering it and Factors of N are points located in the vertices of this polygon: Parallel Planar Point Location algorithms involve Cascading technique to find the factor query points p and q on the faces of this polygon. Cascading algorithms execute in multiple levels of the processor tree per stage and there are logarithmic number of stages - O(logN) parallel time. Cascading derives its nomenclature from relay nature of computation. Node at height h is "woken-up" and computes its subtree only after certain number of stages e.g h. This reduces integer factorization to Parallel Planar Point Location on Polygon.

34.76) Multicores,PRAMs,BSP and NC - [Vijaya Ramachandran] - https://womenintheory.files.wordpress.com/2012/05/vijaya-wit12.pdf

34.77) Word Size in a PRAM - https://hal.inria.fr/inria-00072448/document - Equation 5 - logN <= w (N is input size and w word size)

34.78) Prefix Sum Computation and Parallel Integer Sorting - [Sanguthevar Rajasekaran, Sandeep Sen] - http://www.cse.iitd.ernet.in/~ssen/journals/acta.pdf - "...Specifically, we show that if the word length is sufficiently large, the problem of integer sorting reduces to the problem of prefix sum computation. As a corollary we get an integer sorting algorithm that runs in time O(log n) using n/log n processors with a word length of n^epsilon, for any constant epsilon > 0...."

34.79) AMD Kaveri Heterogeneous System Architecture and hUMA - treats CPU and GPUs equally, best suited for algorithms involving binary searches, zero-copy (GPU and CPU have access to memory by pointers) - an example for Parallel RAM - https://www.anandtech.com/show/7677/amd-kaveri-review-a8-7600-a10-7850k/6, https://pdfs.semanticscholar.org/90ec/1e63da44821d94c86047fa2f728504d89a4d.pdf

34.80) Philosophical implications of Factorization in NC - NC in BPNC in RNC in QNC in BQP in DQP in EXP - https://www.math.ucdavis.edu/~greg/zoology/diagram.xml - Presently factorization is known to be in BQP and NC-PRAM factorization implies factorization is in NC, implying a decoherence from Quantum to Classical world - Rabi Oscillations - https://www.youtube.com/watch?v=OONFQgwbV9I - superposed quantum state |p> = Sigma(|ai>si) settles in one of the classical states si on interaction with classical world. No cloning theorem implies quantum state cannot be replicated. Since NC-PRAM equivalence is in BQP, a natural question arises: Are Parallel RAMs replicated quantum states and thus violate No Cloning?

34.81) An Oracle Result for relativized BQP and P - [Fortnow-Rogers] - https://arxiv.org/pdf/cs/9811023.pdf - There exists an oracle A for which P^A = BQP^A - Theorems 4.1,4.2,4.3 - Pages 9-10 - This is significant because with respect to some oracle, Quantum computation is equivalent to and can be simulated by Deterministic Classical Computation in polynomial time. In the context of PRAM-NC factorization, this theorem applies because NC^A is in P^A (Christopher Wilson - https://link.springer.com/chapter/10.1007/3-540-16486-3_111) for any oracle A. Sequential Optimization for Computational Geometric Factorization mentioned in 506 implies a weaker result - Factorization is in P. An example Oracle for Factorization is the set of beacons (approximate factors) obtained by Ray-Shooting queries. Shor's Quantum Factorization (mentioned in 350) involves period finding - finding period r such that f(x)=f(x+r). Ray shooting queries are also period finding algorithms in the sense that gaps

between prime factors are estimated approximately - if f is a ray shooting query oracle A, querying f returns the next prime factor pf(n+1) = pf(n) + gap (exactly or approximately) in classical deterministic polynomial time which is in NC^A or P^A. Ray shooting can be performed in both P and BQP from the previous equivalence of period finding and prime factor gap ray shooting i.e in P^A and BQP^A.

34.82) Factorization by Planar Point Location and Persistent Binary Search Trees - [Sarnak-Tarjan] - https://pdfs.semanticscholar.org/731f/a2d06bd902b9fd7f3d5c3a17485edfe4133c.pdf - Planar Point Location problem is defined as locating the polygon containing the query point in a subdivision of 2D plane by the line segments creating the polygons. Persistent Binary Search Trees are versioned Binary Trees which support insert/delete and preserve the past history of the Binary Search Tree for queries. Factorization is a Planar Point Location problem - Pixelated hyperbolic tile segments create set of polygons (arrays of pixels forming a rectangle of dimension 1 * length_of_tile_segment) and these polygons divide 2D plane. Factor points (pixels) are located in some of these polygons. Planar Point Location retrieves the polygon containing the factor point in O(logN) time. This retrieved polygon tile segment can in turn be binary searched in O(logN) time because of implicit sortedness (either x or y axis points are strictly ascending). To be in NC, Parallel RAM Construction of Persistent Binary Search Tree is necessary.

34.83) Factorization by Planar Point Location - [PARALLEL TRANSITIVE CLOSURE AND POINT LOCATION IN PLANAR STRUCTURES, ROBERTO TAMASSIA AND JEFFREY S. VITTER] - Parallel RAM construction of Bridge Separator Tree for Planar Point Location - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.72.4319&rep=rep1&type=pdf - Separators are st-paths in planar subdivision graph formed by the hyperbolic line segments of pixel-array polygons on 2D plane. "...The separator tree uses O(n) space and supports point location queries in O((logn)^2) time, where n is the number of vertices of S [33]..." - Lemma 4.5 - "... Let S be a regular subdivision with n vertices. The bridged separator tree for point location in S can be constructed by an EREW PRAM in O(log n) time using n/log n processors, which is optimal ...". Thus a factor point N=pq within some of the hyperbolic pixel-array polygons can be found in O((logN)^2 + logN) = O((logN)^2) PRAM time and O(N/logN) processors. Number of rectangle polygons in planar subdivision created by the hyperbolic pixel-array = N of vertices 4*N. This concurs with O((logN)^2) to O((logN)^3) PRAM time for Segment Trees. This factorization by parallel planar point location is work-optimal and is in NC requiring N/logN processors.

34.84) Criticism and defence of PRAM model - [Casanova-Legrand-Robert] - Parallel Algorithms - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.466.8142&rep=rep1&type=pdf - Section 1.5 - "...NC is the class of all problems which, with a polynomial number of PUs, can be solved in polylogarithmic time. An algorithm of size n is polylogarithmic if it can be solved in O(log(n)^c) time with O(n^k) PUs, where c and k are constants. Problems in NC can be solved efficiently on a parallel computer ... common criticism of the PRAM model is the unrealistic assumption of an immediately addressable unbounded shared parallel memory. ... • Theory is not everything: theoretical results are not to be taken as is and implemented by engineers. • Theory is also not nothing: the fact that an algorithm cannot in general be implemented as is does not mean it is meaningless. ..."

34.85) Parallel Point in Planar Subdivision - Design and Analysis of Parallel Algorithms - [Selim Akl] - https://dl.acm.org/citation.cfm?id=63471 - Section 11.2.2 - Point in Planar Subdivision

34.86) Alternatives to Parallel RAM - QSM - http://theory.stanford.edu/~matias/papers/qsm.pdf - [Philip b.Gibbons, Yossi Matias, Vijaya Ramachandran] - Parallel RAM algorithms lay stress on sharing memory by processing elements while Bridging models like QSM, LogP and BSP (Bulk Synchronous Parallel) rely on Message Passing between processors having local

non-shared memory and communicating with peer elements within finite number of messages (h-relation). Each computation is termed a superstep constrained by barrier synchronization amongst the processors performing local computation.

34.87) Communication Efficient Parallel Sorting - [Michael T.Goodrich] - https://www.ics.uci.edu/~goodrich/pubs/parsort-pre.pdf - Parallel Sorting on BSP instead of PRAMs - footnote 1 - page 247: PRAM is a BSP model having h-relation = 1 and therefore BSP is in NC by this PRAM simulation.

34.88) PRAM and Bulk Synchronous Parallel - http://www.cse.iitd.ernet.in/~subodh/courses/COL730/pdfslides/6pram.pdf - Schematic diagram for Barrier Synchronization Superstep

34.89) Bridging model for Parallel Computation - http://web.mit.edu/6.976/www/handout/valiant2.pdf - [Leslie G.Valiant] - "... that even the most general model, the CRCW PRAM, can be simulated optimally on the BSP model given sufficient slack if g is regarded as a constant..." - slack is the ratio of number of virtual processors (v=plogp) to physical processors (p) = plogp/p and g is the ratio of total computation performed by all local nodes per second to total number of data words delivered per second.

34.90) Definition of NC - Computational Complexity - [Christos Papadimitriou] - Page 375 - Problems solvable by polynomial number of PRAMs in polylogarithmic depth.

34.91) NOW-Sort - https://people.eecs.berkeley.edu/~fox/summaries/database/nowsort.html - High-Performance Sorting on Networks of Workstations - MinuteSort

34.92) NSort - Parallel Sorting - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.23.1823

34.93) Improved Sorting Networks - [R.L. (Scot) Drysdale, Frank H. Young] - https://search.proquest.com/openview/3d400dff715e308df20e8812113844d7/1?pq-origsite=gscholar&cbl=666313 - SIAM Journal of Computing Volume 4, Number 3, 1975

34.94) Definition of Sorting Networks - Runtime of Sorting Networks is its depth and there exists an O(logN) depth Sorting Network - https://courses.engr.illinois.edu/cs573/fa2012/lec/lec/19_sortnet.pdf - implies sorting networks is in NC.

34.95) Ajtai-Komlos-Szemeredi (AKS) Sorting Network - Proof of existence of O(logN) depth Sorting Network - https://www.cl.cam.ac.uk/teaching/1415/AdvAlgo/lec2_ann.pdf - Sorting Networks are graphs consisting of input and comparator wires. Input wires are connected by comparator at specific points. Inputs traversing this graph while finding a comparator are swapped if top wire has greater value than bottom wire. This process propagates till the output is reached. Output gates contain sorted integers. Logdepth sorting networks are constructed by Bipartite Expander Graphs - approximate halvers - which split the inputs into two sets of smallest and largest in two halves of the wires (bitonic). This implies computational geometric sorting networks of O(logN) time for sorting the hyperbolic tile segments and finding the factors by binary search but constant in O(logN) is notoriously huge.

34.96) Complexity of Boolean Functions - [Ingo Wegener] - Blue Book - https://eccc.weizmann.ac.il/resources/pdf/cobf.pdf - Circuit lowerbounds are difficult because efficiency has to be proved over all possible circuits computing a function - Theorem 7.5 - Page 402 - Boolean function on n variables can be computed by a PRAM time lowerbound of Omega(loglogn). This lowerbound is relevant to Computational geometric factorization by binary searching hyperbolic tile pixelation on PRAMs(number of variables = number of bits in integer to factorize). Integer factorization circuits on PRAMs (e.g PRAM circuits for

parallel planar point location algorithms to locate a factor point on hyperbolic arc bow) have polylogarithmic depth (parallel time) and for each PRAM in the longest path from root to leaf of the circuit, this lowerbound applies. Factorization lowerbound for all PRAMs along this longest path is Omega((logN)^k*logloglogN) if n=logN.

34.97) List of Parallel Planar Point Location Algorithms - Table 42.5.1 - Parallel Geometric Searching Algorithms - https://www.ics.uci.edu/~goodrich/pubs/42.pdf

34.98) Parallel Planar Point Location - [Atallah-Goodrich-Cole] - Purdue e-Pubs - https://pdfs.semanticscholar.org/f80e/bfe40dfbfad40659ff82e80cb632a3feeeb8.pdf?_ga=2.211071314.671597020.1548674481-1949614478.1548674481 - Section 4.2 - Factorization is a planar point location problem - Hyperbolic arc pixelation creates juxtaposed rectangular polygons of tile segments each of them internally sorted. Factorization N=pq amounts to finding the polygon (segment) containing the point N which is O(logN) in CREW PRAM time. Binary Searching this implicitly sorted polygon yields the factor point (p,q) and additional O(logN) time.Tiles are computed by the relation delta = N/[x(x+1)]

34.99) Optimal Bounds for Decision Problems on the CRCW PRAM - [Beame-Hastad] - https://homes.cs.washington.edu/~beame/papers/crcwparity.pdf - "...Optimal Q(logn/log logn) lower bounds on the time for CRCW PRAMS with polynomially bounded numbers of processors or memory cells to compute parity and a number of related problems are proven..." - Corollary 4.2 - these lowerbounds apply to all sorting problems on CRCW PRAMs and to k-mergesort/segment tree of hyperbolic segments in computational geometric factorization.

34.100) BSP versus PRAM - Survey - https://web.njit.edu/~alexg/courses/cis668/Fall2000/handsub5.pdf

34.101) Multipoint Planar Point Location - Algorithmic Motion Planning and Related Geometric Problems on Parallel Machines - [Suneeta Ramaswami] - Section 5.1 - https://repository.upenn.edu/cgi/viewcontent.cgi?article=1271&context=cis_reports - Parallel Multipoint planar point location on rasterized hyperbolic arc bow finds multiple factor points spread across multiple pixel array polygons

35. (THEORY) Above Discrete Hyperbolic Factorization can be used as a numerical dataset analysis technique. Discrete Hyperbolic Factorization uses only elementary geometric principles which can be fortified into an algebraic geometry result, because of strong connection between geometry (hyperbola) and algebra (unique factorization theorem) for integer rings - excluding Kummer's theorem for counterexamples when Unique Factorization is violated(e.g (sqrt(5)+1)(sqrt(5)-1)/4 = 2*3 = 6).

##
B. EXPERIMENTAL NON-STATISTICAL INFERENCE MODEL BASED ON ALREADY KNOWN THEORETICAL COMPUTER SCIENCE RESULTS:
##

36.1. (FEATURE - DONE-WordNet Visualizer implementation for point 15) The classification using Interview Algorithm Definition Graph (obtained from Recursive Gloss Overlap algorithm described in http://arxiv.org/abs/1006.4458 , https://sites.google.com/site/kuja27/TAC2010papersubmission.pdf?attredirects=0 and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf) wordnet node indegrees can classify a same document in multiple classes without any training set (unsupervised). Thus same document will be in multiple classes. This can be visualized as one stack per class and documents in same class are pushed into a stack corresponding to each class. When a same document

is across multiple classes, a document can be visualized as a "hyperedge" of a hypergraph that transcends multiple class nodes. Here each class node in this non-planar graph (or multi-planar graph) is a stack.

36.2. (THEORY) Thus if number of classes and documents are infinite, an infinite hypergraph is obtained. This is also somewhat a variant of an inverted index or a hashtable where in addition to hashing ,the chained buckets across the keys are interconnected (Quite similar to https://sites.google.com/site/kuja27/SurvivalIndexBasedTxnTimeoutManager.pdf?attredirects=0 and https://sites.google.com/site/kuja27/PhDThesisProposal.pdf?attredirects=0). This is also similar to "Connectionism" in Computational Psychology which allows cycles or interconnections in Perceptrons and Neural Networks. An old version of this was written in an old deleted blog few years ago in 2006 at: http://shrinivaskannan.blogspot.com. If the relation between Hash Functions and Integer Partitions is also applied as described using Generating Functions in https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions.pdf?attredirects=0 then an upperbound on number of possible Hypergraphs (visualizing them as interconnected Hash tables) can be derived which is a stronger notion to prove.

37.(FEATURE - PoC WordNet Visualizer Closure DONE) Above multiplanar hypergraph is quite related as a theoretical construct to a Pushdown Automata for Context Free Grammar. Also the Definition Graph Construction using Recursive Gloss Overlap is a Context-Sensitive counterpart of Parse Trees obtained for Context Free Grammar, but ignoring Parts-Of-Speech Tagging and relying only on the relation between words or concepts within each sentence of the document which is mapped to a multipartite or general graph. Infact this Definition Graph and Concept Hypergraph constructed above from indegrees quite resemble a Functional Programming Paradigm where each relationship edge is a Functional Program subroutine and the vertices are the parameters to it. Thus a hyperedge could be visualized as Composition Operator of Functional Programs including FP routines for Parts-of-Speech if necessary [WordNet or above Concept Hypergraph can be strengthened with FPs like LISP, Haskell etc.,]. RecursiveNeuralNetworks(MV-RNN and RNTN) have a notion of compositionality quite similar to Functional Programming in FPs (Reference: http://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf) but for tree structures. Applying the compositionality for Concept Hypergraphs constructed above using FPs is a possible research direction.

38.(THEORY) But above is not context free because context is easily obtainable from the hyperedge which connects multiple classes. As an experimental gadget above seems to represent an alternative computational model for context sensitivity and also process of human thought. This needs HyperEdge Search algorithms for hypergraph. A Related Neuropsychological notion of Hippocampus Memory Allocation in Human Brain can be found in http://people.seas.harvard.edu/~valiant/Hippocampus.pdf. Also a similar gadget is a special case of Artificial Neural Network - http://en.wikipedia.org/wiki/Hopfield_network - Hopfield Network - used to model human associative memory (pattern based memory retrieval than address based retrieval). Hyperedges of the above hypergraph can be memory vectors retrieved in an associative memory.

39.(THEORY) An interesting academic question to pose is - "Does the above hypergraph have a strong connectivity and a diameter upperbounded by a constant?". Equivalently, is the collection of universal knowledge close-knit or does nature connect seemingly unrelated concepts beyond what is "observable"? For example, path algorithms for Hypergraphs etc., can be applied to check s-t connectivity of two concepts. Some realworld linear programming and optimization problems which have strong natural applications are KnapSack problem,Tiling Problem or Packing problem that occur in everyday life of humanbeings (packing items for travel, arranging on a limited space etc.,). Thus the concept hypergraph might have a path connecting these.

40.(THEORY) A recent result of Rubik's cube (http://www.cube20.org/)

permutations to get solution having a constant upperbound (of approximately 20-25 moves) indicates this could be true (if all intermediary states of Rubik's transitions are considered as vertices of a convex polytope of Concepts then isn't this just a Simplex algorithm with constant upperbound? Is every concept reachable from any other within 20-25 moves? Or if the above hypergraph is considered as a variant of Rubik's Cube in higher dimensions, say "Rubik's Hypercube" then is every concept node reachable from the other within 20 logical implication moves or is the diameter upperbounded by 20?). Most problems for hypergraph are NP-Complete which are in P for graphs thus making lot of path related questions computationally harder. Counting the number of paths in hypergraph could be #P-Complete. There was also a recent study on constant upperbound for interconnectedness of World Wide Web document link graph which further substantiates it (http://www9.org/w9cdrom/160/160.html - paragraph on diameter of the Strongly Connected Component Core of WWW). Probably this is a special case of Ramsey theory that shows order emerging in large graphs (monochromatic subgraphs in an arbitrary coloring of large graph).

41.(THEORY) Mapping Rubik's Cube to Concept Wide Hypergraph above - Each configuration in Rubik's Cube transition function can be mapped to a class stack node in the Concept Hypergraph. In other words a feature vector uniquely identifies each node in each class stack vertex of the Hypergraph. Thus move in a Rubik's cube is nothing but the hyperedge or a part of the hyperedge that contains those unique vectors. Thus hyperedges signify the Rubik's cube moves. For example if each class stack vertex is denoted as c(i) for i-th class and each element of the stack is denoted by n(i) i.e. nth element of stack for class i, then the ordered pair [c(i),n(i)] uniquely identifies a node in class stack and correspondingly uniquely identifies an instantaneous face configuration of a Rubik's Hypercube(colored subsquares represent the elements of the feature vector). Thus any hyperedge is a set of these ordered pairs that transcend multiple class stacks.

42.(THEORY) The Multiplanar Primordial Field Turing Machine model described by the author in "Theory of Shell Turing Machines" in [http://sites.google.com/site/kuja27/UndecidabilityOfFewNonTrivialQuestions.pdf?attredirects=0] with multiple dimensions seems to be analogous to the above. If dimension info is also added to each plane in above hypergraph then both above formulations and Shell Turing Machines look strikingly similar. One more salient feature of this hypergraph is that each class node stack indirectly encodes timing information for events from bottom to top of the stack and height of each stack independently varies. Thus this is more than just a hypergraph. There also exists a strong similarity with Evocation WordNet (one word "reminding" or "evocative" of the other) - http://wordnet.cs.princeton.edu/downloads.html - difference being above is a hypergraph of facts or concepts rather than just words. The accuracy of sense disambiguation is high or even 100% because context is the hyperedge and also depends on how well the hyperedges are constructed connecting the class stack vertices.

43.(THEORY) Instead of a wordnet if a relationship amongst logical statements (First Order Logic etc.,) by logical implication is considered then the above becomes even more important as this creates a giant Concept Wide Web as mentioned in [http://sourceforge.net/projects/acadpdrafts/files/ImplicationGraphsPGoodEquationAndPNotEqualToNPQuestion_excerpts.pdf/download] and thus is an alternative knowledge representation algorithm.

44.(THEORY) Implication graphs of logical statements can be modelled as a Random Growth Network where a graph is randomly grown by adding new edges and vertices over a period of learning.

45.(THEORY) If a statistical constraint on the degree of this graph is needed, power law distributions (Zipf, Pareto etc.,) can be applied to the degrees of the nodes along with Preferential Attachment of newborn nodes.

46.(THEORY) Set cover or Hitting Set of Hypergraph is a Transversal of

hypergraph which is a subset of set of vertices in the hypergraph that have non-empty intersection with every hyperedge. Transversal of the above Concept Wide Hypergraph is in a way essence of the knowledge in the hypergraph as in real world what this does is to grasp some information from each document (which is a hyperedge of class vertices) and produces a "summary nucleus subgraph" of the knowledge of the encompassing hypergraph.

47.(THEORY) Random walk on the above concept hypergraph, Cover time etc., which is a markov process. Probably, Cover time of the hypergraph could be the measure of time needed for learning the concept hypergraph.

48.(THEORY - IMPLEMENTATION - DONE) Tree decomposition or Junction Trees of concept hypergraph with bounded tree width constant.[If Junction tree is constructed from Hypergraph modelled as Bayesian Network, then message passing between the treenodes can be implemented, but still needs to be seen as what this message passing would imply for a junction tree of concept hypergraph above.]. A tree width measure computing script for Recursive Gloss Overlap graph for a text document has been added in python-src/InterviewAlgorithm which is a standard graph complexity measure.

49.(THEORY) 2(or more)-dimensional random walk model for thinking process and reasoning (Soccer model in 2-dimensions):

As an experimental extension of point 47 for human reasoning process, random walk on non-planar Concept hypergraph (collective wisdom gained over in the past) or a planar version of it which is 2-dimensional can be theorized. Conflicts in human mind could mimick the bipartite competing sets that make it a semi-random walk converging in a binary "decision". Semi randomness is because of the two competing sets that drive the "reasoning" in opposite directions. If the planar graph is generalized as a complex plane grid with no direction restrictions (more than 8), and the direction is anything between 0 to 2*pi radians, then the distance d after N steps is sqrt(N) (summation of complex numbers and the norm of the sum) which gives an expression for decision making time in soccer model.(related: http://web.archive.org/web/20041210231937/http://oz.ss.uci.edu/237/readings/EBRW_nosofsky_1997.pdf). In other words, if mental conflict is phrased as "sequence of 2 bipartite sets of thoughts related by causation" then "decision" is dependent on which set is overpowering. That is the corresponding graph vertices are 2-colored, one color for each set and yet not exactly a bipartite graph as edges can go between nodes of same set. Brownian Motion is also a related to this. Points (53.1) and (53.2) describe a Judge Boolean Function (with TQBF example) which is also a decision making circuit for each voter in P(Good) summation with interactive prover-verifier kind of adversarial notions based on which a voter makes a choice.

##
##
C.   Slightly Philosophical - Inferences from conflicting opinions:
##
##

50. (DONE) As an example, if there are "likes" and "dislikes" on an entity which could be anything under universe - human being, machine, products, movies, food etc., then a fundamental and hardest philosophical question that naturally has evaded an apt algorithm: Is there a way to judge an entity in the presence of conflicting witnesses - "likes" and "dislikes" - and how to ascertain the genuineness of witnesses. In http://arxiv.org/abs/1006.4458 and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf this has been the motivation for Interview Algorithm and P(Good) error probability for majority voting which is the basis for this. Moreover the opinions of living beings are far from correct due to inherent bias and prejudices which a machine cannot have.

51. (DONE) Another philosophical question is what happens to the majority voting when every voter or witness is wrong intentionally or makes an error by innocous mistake.[http://sourceforge.net/projects/acadpdrafts/files/ImplicationGraphsPGoodEquationAndPNotEqualToNPQuestion_excerpts.pdf/download]. This places a theoretical limitation on validity of "perceptional judgement" vis-a-vis the "reality".

52. (THEORY) There is a realword example of the above - An entity having critical acclaim does not have majority acclaim often. Similarly an entity having majority acclaim does not have critical acclaim. Do these coincide and if yes, how and when? This could be a Gaussian where tails are the entities getting imperfect judgements and the middle is where majority and critical acclaim coincide.

53. (THEORY) Items 50,51,52 present the inherent difficulty of perfect judgement or perfect inference. On a related note,Byzantine algorithms have similar issues of deciding on a course of action in the presence of faulty processors or human beings and faulty communications between them. Thus, probably above problem reduces to Byzantine i.e Faulty human beings or machines vote "like" or "dislike" on an entity and a decision has to be taken. This is also in a way related to Boolean Noise Sensitivity where collective intelligence of an entity can be represented as a "judging" boolean function and its decision tree. This "judge" boolean function accepts arguments from two opposing parties and outputs 1 or 0 in favour of one of them. How perfect is this "judge" depends on NoiseSensitivity of its boolean function. Thus a perfect judge boolean function is 100% NoiseStable even though attempts are made to confuse it through noise inputs.

53.1 (THEORY) Judge Boolean Function (there doesn't seem to be an equivalent to this in literature - hence it is defined as below):

A Judge Boolean function f:(0,1)^n->(0,1) has two classes of input sets of bit sets (x1,x2,x3,...) and (y1,y2,y3,...) corresponding to two adversarial parties x and y and outputs 0 (in favor of x) or 1 (in favor of y) based on the decision tree algorithm internal to f. Each xi and yi are sets of bits and hence input is set of sets (Intuitively such classes of boolean functions should be part of interactive proof systems.). This function is quite relevant to P(Good) summation because:

- LHS PRG/Dictator boolean function choice can also be thought of as randomly chosen Judge boolean function (from a set of judge boolean functions) defined above. NoiseStability of this Judge boolean function is its efficacy.
- In RHS, each voter has a judge boolean function above and NoiseStability of Boolean Function Composition of Maj*Judge is its efficacy.
- An example: From https://sites.google.com/site/kuja27/InterviewAlgorithmInPSPACE.pdf?attredirects=0, Judge Boolean Function can be a Quantified Boolean Formula (QBF) i.e (There exists f(xi) (For all g(yk) ...(((...))). Intuitively this simulates a debate transcript (prover-verifier) between x and y that is moderated by the QBF judge boolean function. Of interest is the NoiseStability of this QBF.
- QBF is PSPACE-complete and hence Judge Boolean Function is PSPACE-complete.
- Thus LHS of P(good) is in PSPACE (a pseudorandomly chosen Judge Boolean Function) while RHS of P(good) is still in PH=DC where each voter's Judge Boolean Function is fed into Majority circuit inputs, because, if the QBF is of finite depth then RHS is depth-restricted and due to this RHS is in PH=DC. Thus if P(Good) probability is 1 on either side then there exists a PSPACE algorithm for PH implying PSPACE=PH than an intimidatingly unacceptable P=PH. If there is no quantifier depth restriction on either side both LHS and RHS can grow infinitely making both of them EXP. This makes Judge Boolean Function somewhat a plausible Voter Oracle than simple boolean functions - each voter is intrinisically allowed an intelligence to make a choice amongst 2 adversaries.

- Since Judgement or Choice among two adversaries is a non-trivial problem, it is better to have each Voter's SAT in RHS and that of pseudorandom choice in LHS to be in the hardest of complexity classes known. Thus interpretation of convergence of P(Good) series depends on hardness of judgement boolean function (it could be from simple 2-SAT,3-SAT,k-SAT upto PSPACE-QBFSAT and could be even more upto Arithmetic Hierarchy) some of which might have circuit_complexity(LHS) == circuit_complexity(RHS) while some might not.

- Reference: QBF-SAT solvers and connections to Circuit Lower Bounds - [RahulSanthanam-RyanWilliams] - http://web.stanford.edu/~rrwill/QBFSAT_SODA15.pdf

- A complex Judging scenario: Doctrinal Paradox (or) Discursive dilemma - Paradoxes in Majority voting where both Yes and No are possible in judgement aggregations - https://en.wikipedia.org/wiki/Discursive_dilemma. [Open question: What is the majority circuit or boolean equivalent of such a paradox with both Yes and No votes]

53.2 (THEORY) P(Good) Convergence, Voter TQBF function, PH-completeness and EXP-completeness - Adversarial reduction

---

P(Good) RHS is in PH if depth restricted and in EXP if depth unrestricted. LHS is a pseudorandomly chosen Judge Boolean Function. PH-completeness or EXP-completeness of RHS remains to be proven (if not obvious). PH is the set of all problems in polynomial hierarchy - PH=U sigma(p,i). For every i there exists a problem in sigma(p,i) that is sigma(p,i)-complete. If there exists a PH-complete problem then PH collapses to sigma(p,i) for some level i.[Arijit Bishnu - http://www.isical.ac.in/~arijit/courses/spring2010/slides/complexitylec12.pdf] . Hence it remains an open question if there exists a PH-complete problem. Thus instead of depth restriction, the hardest depth unrestricted EXP circuit class(EXP=DC) is analyzed as a Judge Boolean Function. There are known EXP-complete problems e.g. Generalized Chess with n*n board and 2n pawns - [Fraenkel - http://www.ms.mff.cuni.cz/~truno7am/slozitostHer/chessExptime.pdf]. In RHS, each voter has a Judge Boolean Function, output of which is input to Majority circuit. This Judge Boolean Function can be shown to be EXP-complete by reduction from Chess. Reduction function is as below:

Each Voter can be thought of as simulator of both adversaries in Chess i.e Voter TQBF has two sets of inputs x1,x2,x3,... and y1,y2,y3,... Each xi is a move of adversary1-simulated-by-voter while each yi is a move of adversary2-simulated-by-voter. Voter thus makes both sides of moves and votes for x or y depending on whichever simulation wins. Thus there is a reduction from generalized Chess EXP-complete problem to Voter Judge Boolean Function (special cases of Chess with n constant are PSPACE-complete). Hence RHS is a Majority(n) boolean function composed with EXP-complete Judge Boolean Function (assumption: voters vote in parallel). When P(Good) summation converges to 1(i.e Judge Boolean Function has zero noise and 100% stability existence of which is still an open problem), LHS is an EXP-complete algorithm for RHS which is harder than EXP-complete - it is still quite debatable as to how to affix notation to above RHS - probably it is P/EXP(access to an oracle circuit) or P*EXP(composition).RHS of P(Good) belongs to a family of hardest of boolean functions. [Kabanets - http://www.cs.sfu.ca/~kabanets/papers/thes-double.pdf] describes various classes of hard boolean functions and Minimum Circuit Size Problem [is there a circuit of size at most s for a boolean function f] and also constructs a read-once(each variable occurs once in the branching) branching program hard boolean function [Read-Once Branching Programs - http://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/RWY97/proc.pdf, Bounded Width polysize Branching Programs and non-uniform NC1 - http://www.complexity.ethz.ch/education/Lectures/ComplexityHS10/ScriptChapterTwelve] . In a general setting, RHS of P(Good) summation is m-EXP-complete (staircase exponents). RHS always has a fixed complexity class (PH-complete or EXP-complete or NEXP-complete if RHS is non-deterministically evaluated) while only complexity class of LHS changes due to pseudorandomly chosen Judge Boolean Function. Even Go is EXP-complete which has close to 10^170 possibilities which is greater than chess - same person

simulates both sides of adversaries and captures territory by surrounding.

A special case of Complexity of Quantified 3-CNF Boolean Forumlas is analyzed in http://eccc.hpi-web.de/report/2012/059/download/ - if 3CNF quantified boolean formulas of n variables and poly(n) size can be decided in 2^(n-n^(0.5+e)) time deterministically, NEXP is not contained in NC1/poly. This has direct implications for P(Good) majority voting circuit - if LHS is 100% noise stable NC1/poly percolation boolean function circuit and RHS is Quantified 3-CNF QBFs as Judge Boolean Functions decidable in 2^(n-n^(0.5+e)) time then, since NEXP in RHS can not have NC1/poly circuits, RHS P(Good) summation shouldn't be equal to LHS which implies that RHS NEXPTIME Quantified 3-CNF decidable in 2^(n-n^(0.5+e)) time cannot be 100% noise stable. There are other NEXP-complete problems viz., Succinct 3-SAT. The 100% noise stability regime for percolation boolean function is described in [http://arxiv.org/pdf/1102.5761.pdf - page 68]

Depth Hierarchy Theorem ,a recent result in 2015 in http://arxiv.org/pdf/1504.03398.pdf proved PH to be infinite relative to some and random oracles i.e PH does not collapse with respect to oracles, which mounts the evidence in favour of non-existence of PH-Complete problems, but does not rule them out.

53.3 (THEORY- *IMPORTANT*) Conditions for complexity classes of LHS and RHS of P(Good) summation being equated

---

RHS of P(Good) is either PH-complete or EXP-complete as mentioned in 53.2. LHS of P(Good) is a pseudorandomly chosen Judge Boolean Function while RHS is a huge mixture of judge boolean functions of almost all conceivable complexity classes depending on voter which together make the DC-circuit. LHS thus can belong to any of the complexity classes depending on the pseudorandom choice.

Probability of LHS Judge Boolean Function belonging to a complexity class C with NoiseStability p is =
Probability of LHS PRG choice Judge Boolean Function in complexity class C * Probability of Goodness in terms of NoiseStability of PRG choice =
 c/n * p where c is the number of judge boolean functions in complexity class C and n is the total number of judge boolean functions

When p=1 and c=n LHS is 1 and is a complexity class C algorithm to RHS PH-complete or EXP-complete DC circuit when RHS P(Good) summation in terms of NoiseStabilities also converges to 1. Thus complexity_class(LHS) has a very high probability of being unequal to complexity_class(RHS). This is the perfect boolean function problem (in point 14 above) rephrased in probabilities. This is very tight condition as it requires:
  - boolean function composition in RHS is in some fixed circuit complexity class (PH-complete or EXP-complete) and only LHS complexity class varies depending on pseudorandom choice
  - LHS has to have 100% noise stability
  - RHS has to have 100% noise stability
Thus above conditions are to be satisified for equating complexity classes in LHS and RHS to get a lowerbound result. Also convergence assumes binomial complementary cumulative mass distribution function as mentioned in 14 above which requires equal probabilities of successes for all voters (or equal noise stabilities for all voter boolean functions). Hence voters may have different decision boolean functions but their noise stabilities have to be similar. If this is not the case, P(Good) series becomes a Poisson trial with unequal probabilities.

If LHS and RHS of P(Good) are not equatable, probability of goodness of LHS and RHS can vary independently and have any value. LHS and RHS are equatable only if the probability of goodness (in terms of noise stabilities on both sides) is equal on both sides. (e.g 0.5 and 1). Basically what this means is that LHS is

an algorithm as efficient as that of RHS. If a pseudorandom choice judge boolean function has 100% noise stability then such a choice is from the huge set of boolean functions on RHS .This is an assumption based on the fact that any random choice process has to choose from existing set. Also when LHS pseudorandom judge boolean function choice is 100% noise stable, if RHS had LHS as one of its elements it would have forced RHS to be 100% and noise stabilities of all other judge boolean functions in RHS to 0%. Because of this RHS has to exclude LHS pseudorandom choice judge boolean function. That is:

cardinality(RHS) + cardinality(LHS) = cardinality(Universal set of judge boolean functions) (or)

cardinality(RHS) + 1 = cardinality(Universal set)

which is the proverbial "one and all".

Assuming there exists a boolean percolation function with noise stability 100%, voter with such a boolean function could be a pseudorandom choice in LHS and RHS has to exclude this function.

53.4 (THEORY) Variant 1 of Percolation as a Judge boolean function (related to 14 above)

Percolation boolean function returns 1 if there is a left-right crossing event in a path within the Z^2 grid random graph else 0. The definition of percolation is modified slightly as below to obtain a ranking of 2 adversaries:
There are n parameters on which the voter or a judge weighs two adversaries a and b (each take the 2 coordinates in the grid) and scores them as set of ordered pairs (a1,b1),(a2,b2),(a3,b3),...,(an,bn) where each ai is the score given by voter for adversary a on parameter i and each bi is the score given by voter for adversary n on parameter i. These coordinates are plotted on the grid and a left-right traversal of these coordinates is done via a path that covers all of them. At each coordinate, total score summations of ai(s) and bi(s) so far traversed are computed. The boolean function outputs 1 if sum(ai) > sum(bi) favouring adversary a and outputs 0 if sum(ai) < sum(bi) favouring adversary b. This is just a 2 candidate ranking variant of 3-candidate condorcet elections. In terms of circuit complexity, this is an iterated integer addition circuit(in non-uniform NC1) which inputs to a comparator circuit(in AC0 - [Chandra-Stockmeyer-Vishkin] - http://cstheory.com/stockmeyer@sbcglobal.net/csv.pdf) which together make non-uniform NC1 (since AC0 is contained in NC1). Thus this variant of percolation decides based on sum of integral scores than left-right crossing. It always has a left-right crossing and there are no random graph edges.

53.5 (THEORY) Variant 2 of Percolation as a Judge boolean function (related to 14 above)

53.5.1 Both x and y axes have the criteria common to both adversaries a and b as coordinates.
53.5.2 If adversary a is equal to b for a criterion c, then there is a horizontal left-right edge with probability p.
53.5.3 If adversary b is better than a for criterion c, then there is a vertical edge (down-top) with probability p.
53.5.4 If adversary a is better than b for criterion c, then there is a vertical edge (top-down) with probability p.

53.5.5 Example 1:

---

criterion	a	b
c1	5	4 (top-down)
c2	3	5 (down-top)

```
c3           2    2 (left-right)
c4           4    6 (down-top)
c5           7    8 (down-top)
      --------------------
          21    25  (a < b)
        --------------------
----------------
53.5.6 Example 2:
----------------
criterion    a    b
c1           2    1 (top-down)
c2           7    2 (top-down)
c3           5    5 (left-right)
c4           3    8 (down-top)
c5           9    7 (top-down)
        --------------------
            26    23  (a > b)
        --------------------
```

53.5.7 Length of each edge is proportional to the difference between score(a) and score(b) at criterion c.

53.5.8 Lemma: a is better than b implies a top-down crossing event and b is better than a implies down-top crossing event.

Proof: If a is better than b in majority of the criteria then the path is more "top-down"-ish due to 53.5.4 (top-left to bottom-right) and if b is better than a then path is more "down-top"-ish due to 53.5.3 (bottom-left to top-right). Thus this formulation of percolation subdivides left-right paths into two classes top-down and down-top and all paths are left-right paths.

53.5.9 Above variant of percolation boolean function returns 1 if path is top-down (a > b) else 0 if path is down-top (a < b).

53.5.10 Previous description is an equivalence reduction of percolation and judging two adversaries.

53.5.11 To differentiate top-down and down-top left-right paths, comparator circuits (constant depth, polysize) and non-uniform NC1 circuit for integer addition of scores are required. Together this is also a non-uniform NC1 circuit (for infinite grid) similar to Variant1 in 53.4.

53.6 (THEORY) Circuit for Percolation as a Judge Boolean Function

---

For an m * m grid with random edges, maximum length of the left-right path is m^2 (traverses all grid cells). Each boolean function is a path that is represented as set of coordinates on grid. If these coordinates form a left-right crossing, then circuit has value 1. Each coordinate on the grid requires 2logm bits. Hence maximum input bits required is m^2 * 2logm = O(m^2*logm). Coordinates are sorted on left coordinate left-right ascending by a Sorting Netwok (e.g Batcher-sortingnetworks, Ajtai-Komlos-Szemeredi-AKS-sortingnetworks, Parallel Bitonic Sort - http://web.mst.edu/~ercal/387/slides/Bitonic-Sort97.pdf). This function outputs, with the sorted coordinates as inputs:

- 1 if both x-axis coordinate of the leftmost ordered pair is 0 and x-axis coordinate of the rightmost ordered pair is m
- else 0

which requires comparators. Sorting networks of depth O((logn)^2*n) and size O(n(logn)^2) exist for algorithms like Bitonic sort, Merge sort, Shell sort et.,. AKS sorting network is of O(logn) depth and O(nlogn) size. For n=m^2, AKS sorting network is of O(log(m^2)) depth and O(m^2log(m^2)) size. Thus percolation circuit consists of:

- Sorting network that sorts the coordinates left-right in NC1 or NC2
- Comparator circuits in constant depth and polynomial size

which together are in NC. This is non-uniform NC with advice strings as the circuit depends on percolation grid - e.g BP.(NC/log) where advice strings are the grid coordinates and the circuit has bounded error (Noise+delta). [If the left-right paths are classified as top-down and down-top paths for ranking two

adversaries (in 53.5 above) additional comparator circuits are required for finding if right coordinate is in top or down half-space. This is by comparing the y-axis of the leftmost and rightmost in sorted ordered pairs and output 1 if y-axis of leftmost lies between m/2 and m and y-axis of rightmost lies between 0 and m/2 (top-down) and output 0 else (down-top).]

Above percolation circuit based on Sorting networks is not optimal for arbitrarily large inputs. As per Noise Stability Regime [page 68 - http://arxiv.org/pdf/1102.5761.pdf - Christophe Garban, Jeffrey E.Steif], for epsilon=o(1/n^2*alpha4), Pr[fn(crossing event sequence)=fn(crossing event sequence with epsilon perturbation)] tends to zero at infinity where alpha4 is the four-arm crossing event probability (paths cross at a point on the percolation random graph creating four arms to the boundary from that point). In terms of fourier expansion coefficients this is: summation(fourier_coeff(S)^2) tending to zero, |S| > n^2*alpha4. Circuit for this arbitrarily large percolation grid would be infinitely huge though can be termed as non-uniform NC sorting-network+comparator and hence a non-uniform circuit for an undecidable problem. Noise stability tending to zero depends not just on input, but on "infinite-input".

53.7 (THEORY) P/Poly and P(Good) LHS and RHS - Karp-Lipton and Meyer's Theorems
---

P/poly is the circuit class of polynomial size with polynomial advice strings and unrestricted depth. Thus P/poly is non-uniform circuit class. Hence non-uniform NC is in P/poly. In 53.4, 53.5 and 53.6 example non-uniform NC1 circuits (subject to 100% noise stablility) Percolation boolean function were described. These are in P/poly as non-uniform NC is in P/poly and thus LHS of P(Good) is in P/poly. Hence when LHS and RHS binomial coefficient summations converge to 100% noise stability satisfying the conditions in 53.3, LHS is a P/poly algorithm to RHS EXPTIME DC uniform circuit (because both sides are depth unrestricted). To be precise LHS is an NC/poly circuit. If the advice string is logarithmic size, it is NC/log and there is a known result which states that NP in P/log implies P=NP. For percolation boolean functions with 100% noise stability regime (page 68 - http://arxiv.org/pdf/1102.5761.pdf), if the grid boundaries are the only advice strings, boundary coordinates can be specified in log(N) bits and hence LHS is in NC/log.

If RHS is simply NP (if depth restricted and unlikely for arbitrary number of electorate) then LHS is a P/poly algorithm to RHS NP implying NP is contained in P/poly. By Karp-Lipton theorem NP in P/poly implies that PH collapses to second-level i.e sigma(p,2) and from [Arvind, Vikraman; Köbler, Johannes; Schöning, Uwe; Schuler, Rainer (1995)] NP in P/poly implies AM=MA.

EXPTIME version of this is Meyer's Theorem which states that if EXPTIME is in P/poly (which is the hardest case for RHS of P(good) and can be m-EXPTIME) then EXPTIME = sigma(p,2) \intersection pi(p,2) and EXPTIME=MA. When RHS of P(good) is a DC uniform EXPTIME circuit, P(good) convergence subject to availability of 100% noise stable percolation boolean functions satisfying conditions in 53.3, implies that EXPTIME=MA.

References:

53.7.1 Relation between NC and P/poly - http://cs.stackexchange.com/questions/41344/what-is-the-relation-between-nc-and-p-poly
53.7.2 P/poly - https://en.wikipedia.org/wiki/P/poly
53.7.3 Karp-Lipton Theorem - https://en.wikipedia.org/wiki/Karp%E2%80%93Lipton_theorem
53.7.4 NP in P/poly implies AM=MA - [Arvind, Vikraman; Köbler, Johannes; Schöning, Uwe; Schuler, Rainer (1995),] - http://www.informatik.hu-berlin.de/forschung/gebiete/algorithmenII/

Publikationen/Papers/ma-am.ps.gz
53.7.5 BPP,PH,P/poly,Meyer's theorem etc., - http://www.cs.au.dk/~arnsfelt/CT10/scribenotes/lecture9.pdf
53.7.6 Descriptive Complexity - https://books.google.co.in/books?id=kWSZ0OWnupkC&pg=PA177&lpg=PA177&dq=NC1/poly&source=bl&ots=6oGGZVk4fa&sig=Y_4syLFZ-nTj4L0qrBxxWve75EA&hl=en&sa=X&ved=0CBsQ6AEwAGoVChMI66v6tqqSyQIVyZCOCh2uMwyr
53.7.7 Non-uniform computation - https://courses.engr.illinois.edu/cs579/sp2009/slides/cc-s09-lect10.pdf

---

53.8 (THEORY) An example Majority+SAT majority voting circuit, P(Good) convergence and some observations on Noise Stability

---

Let the number of voters be 8 (=2^3). Each voter has a boolean function of 3 variables - x1,x2 and x3. From Boolean Function Composition, this is defined as Maj(SAT1(x1,x2,x3),SAT2(x1,x2,x3),...,SAT8(x1,x2,x3)). Each variable xi corresponds to a criterion i and xi=1 if and only if the criterion i is satisfied in 2-candidate voting (candidate0 and candidate1). Thus all voters should vote xi=1 if the candidate1 satisfies criterion xi. If some of the voters vote xi=0 even though the candidate1 satisifes criterion xi, then it becomes noise in the boolean function. This composition has exponential size in number of inputs (32 = 2^3 + 3*2^3) = O(n*exp(n))). For negation NOT gates are allowed and it is not necessarily a monotone circuit. If the variables are different for each voter, there are 24 variables and circuit is polynomial in input size - O(n^k).This illustrates the pivotality of noise when voters have a variable in common.

If the RHS doesn't converge to 1 and thus is in BPP and Noise Stability of LHS is 100% (e.g by PRG choice of a percolation function) then LHS is a P/poly algorithm for RHS BPP. This implies BPP is in P/poly , an already known amplification result. Infact this result implies something stronger: Since BPP is in P/poly, even if RHS doesn't converge, there is always a P/poly algorithm in LHS by amplification.

If the RHS has a perfect k-SAT boolean decision function for all voters, size of the circuit is super-polynomial and sub-exponential(if exponential, circuit is DC uniform), P(Good) series converges to 100% and LHS is 100% noise stable Percolation function, then LHS is a P/poly algorithm for RHS NP-complete instance. Special case is when RHS has only one voter - LHS is P/poly (or NC/poly or NC/log) and RHS is NP-complete(assuming k-SAT of RHS voter is 100% noise stable) - and thus there is a P/poly circuit for NP in such a special case. From [Karp-Lipton] (53.7) if NP is in P/poly, PH collapses to second level and AM=MA. If LHS is P/log and RHS is NP, then P=NP. Thus 100% noise stable boolean function in LHS has huge ramifications and the P(Good) summation, its circuit version devour complexity arena in toto and variety of results can be concluded subject to noise stability.

If LHS of P(Good) is in NC/log (Percolation as Judge boolean function) and RHS doesn't converge to 100% (i.e Majority*SAT boolean function composition has < 100% noise stability), LHS is a more efficient NC/log algorithm for RHS BP* hard circuit(e.g RHS is single voter with a SAT judge boolean function without 100% noise stability and thus in BPP). BPP is already known to have P/poly circuits and this implies a better bound that BPP is in NC/log. BPP is not known to have complete problems in which case , NC/log algorithm for RHS BPP-complete problem would imply NC/log = BPP for the single voter RHS with 3-SAT judging function with < 100% noise stability.

Subtlety 1: For single voter in RHS with a 3-SAT voter judge boolean function, RHS is NP-complete from voting complexity standpoint irrespective of noise stability(Goodness of voting). But if noise stability is the criterion and 3-SAT is not noise stable 100%, RHS is a BPP problem. This is already mentioned in

point 14 above(2 facets of voting - judging hardness and goodness).

Subtlety 2: [Impagliazzo-Wigderson] theorem states that if there exists a decision problem with O(exp(n)) runtime and Omega(exp(n)) circuit size, then P=BPP. P(Good) RHS being a DC uniform circuit can have exponential size if variables are common across voter judge boolean functions, but is questionable if it is a decision problem of runtime O(exp(n)) - for m-EXPTIME circuits this bound could be O(staircase_exp(n)).

NC/log circuits can be made into a uniform circuits by hardwiring advice strings which are logarithmic size. There are 2^(log(n)) = n advice strings possible for NC/log circuit with O(logn) depth and O(n^k) size. The n advice strings are added as hardwired gates into NC/log circuit which doesnot alter the polynomial size of the circuit. Due to this NC/log is in NC and hence in P. For NC/log LHS with 100% noise stability, if RHS is BPP-complete with < 100% noise stability, LHS is NC/log algorithm for BPP-complete problem (or) BPP is in NC/log and hence in P. But P is in BPP. Together this implies P=BPP if LHS is NC/log and RHS is a BPP-complete majority voting circuit(special case: one voter with less than 100% noise stability and 3-SAT boolean function). This leads to the question: Is there a 3-SAT voter judge boolean function with 100% noise stability? Is percolation with 100% noise stability regime reducible to 3-SAT? If yes, then LHS [P/log, NC/log or P/poly] percolation boolean function with 100% noise stability solves RHS NP-complete problem and hence NP is in P/poly, PH collapses to second level and AM=MA (or) NP is in [P/log, NC/log] and P=NP.

References:

53.8.1 Pseudorandomness - [Impagliazzo-Wigderson] - http://people.seas.harvard.edu/~salil/pseudorandomness/pseudorandomness-Aug12.pdf
53.8.2 XOR Lemma [Yao] and [Impagliazzo-Wigderson] theorem - theory.stanford.edu/~trevisan/pubs/stv99stoc.ps

---

53.9 (THEORY) Possible Scenarios in P(Good) LHS and RHS summation

---

53.9.1 LHS = RHS is 100% - LHS and RHS are equally efficient (and can be equated to get a lowerbound if equal error implies a lowerbound which is still doubtful. More about this in 53.16).
53.9.2 LHS is < 100% and RHS is < 100% - RHS is more efficient than LHS and both sides are in BP* classes
and LHS < RHS
53.9.3 LHS is < 100% and RHS is < 100% - LHS and RHS are equally efficient (and can be equated to get a lowerbound if equal error implies a lowerbound which is still doubtful. More about this in 53.16). Both are in BP* classes
and LHS = RHS
53.9.4 LHS is < 100% and RHS is < 100% - LHS is more efficient than RHS and both sides are in BP* classes
and LHS > RHS
53.9.5 LHS is 100% and RHS is < 100% - LHS is more efficient than RHS. LHS is non-BP algorithm to RHS BP* algorithm. (e.g P/poly algorithm for BPP)
53.9.6 LHS is < 100% and RHS is 100% - RHS is more efficient than LHS. RHS is non-BP algorithm to LHS BP* algorithm.

53.10 (THEORY) Dictatorship, k-juntas testing and P(Good) circuit - simulation

---

LHS of P(Good) is a pseudorandom choice judge boolean function. It has to be tested for dictatorship and k-junta property. This step adds to
the hardness of LHS. What is the probability of LHS pseudorandom choice being a

dictator or k-junta boolean function? This probability is independent of Goodness probability of LHS which is the noise stability. If the size of the population is n and k out of n are having dictators or k-juntas as judge boolean functions, probability of LHS being dictator or k-juntas is k/n. Following steps are required:

- PRG Choice outputs a Judge boolean function candidate for LHS.
- Dictatorship or k-juntas testing algorithm is executed for this candidate PRG choice.

Expected number of steps required for choosing a dictator via PRG is arrived at by Bernoulli trial. Repeated PRG choices are made and property tested for dictatorship and k-juntas. Probability of PRG choice being a dictator is k/n. Probability p(t) that after t trials a dictator or k-juntas is found:
p(t) = (1-k/n)^(t-1)(k/n) = (n-k)^(t-1) * k/n^t

If t is the random variable, expectation of t is:
E(t) = sigma(t*p(t)) = sigma((n-k)^(t-1)*tk/n^t)

For a property testing algorithm running in O(q), time required in finding a dictator or k-juntas is O(q*sigma((n-k)^(t-1)*tk/n^t))

For k=0:
E(t) = 0
For k=1:
E(t) = sigma((n-1)^(t-1) * t/n^t) = 1/n + (n-1)*2/n^2 + (n-1)^2*3/n^3 + ...
< 1/n + 2n/n^2 + 3n^2/n^3 + ...
= 1+2+3+...+n/n
= n(n+1)/2n
= (n+1)/2

Thus to find one dictator approximately (n+1)/2 PRG expected choices are required. Above is not necessarily an exact process by which LHS arises but just a simulation of one special scenario.

[Axiomatically k shouldn't be greater than 1 i.e there shouldn't be more than one dictator or juntas. Philosophical proof of this is by contradiction. Every element in a population set is subservient to a dictator by definition. The meaning of "dictator" here is global uniqueness for all voters and not locally unique to a voter boolean function. If there are 2 dictators population-wide, this definition is violated.]

References:

53.10.1 Dictatorship,k-juntas testing - [Oded Goldreich] - http://www.wisdom.weizmann.ac.il/~oded/PDF/pt-junta.pdf

---
(THEORY-*IMPORTANT*) 53.11 Hastad-Linial-Mansour-Nisan (HLMN) Theorem and Noise Stable boolean functions - Denoisification 1

Hastad-Linial-Mansour-Nisan Theorem states that for a boolean function f: {0,1}^n->{0,1} computed by circuit of size < s and depth < d, if the fourier coefficients for large S are epsilon-concentrated:
sigma(fourier_coeff(S)^2) < epsilon for |S| > t where t = O((log s/epsilon)^(1/d-1)).
Above implies that:
t = O(log s/epsilon)^(d-1) = k*(log s/epsilon)^(d-1) for some constant k.
(t/k)^(1/d-1) = log(s/epsilon)
s = epsilon*2^((t/k)^1/(d-1))
For a noisy boolean function f with rho probability of flip, the noise operator T is applied to the Fourier expansion of f:
T(f) = sigma(rho^|S|*fourier_coeff(S)*parity(S)) for S in [n] and each Fourier coefficient of T(f) = rho^|S|*fourier_coeff(S)

From HLMN theorem, applying noise operator above (this needs to be verified) to noisy boolean function circuit Fourier spectrum:
if sigma([rho^|S|*fourier_coeff(S)]^2) <= epsilon, |S| > t where t=O((log s/epsilon)^(d-1)),
size s = sigma([rho^|S|*fourier_coeff(S)]^2)*2^[(t/k)^1/(d-1)] substituting for epsilon and |S| > t.
Thus size of the circuit exponentially depends on t - size of parity sets S in [n] - |S| - which can have maximum value of n(number of inputs).

If f is 100% noise stable, from Plancherel theorem:
sigma(rho^|S|*f(S)^2) = 1, S in [n]
Stability can be written as:
sigma_S_<_t(rho^|S|*f(S)^2) + sigma_S_>=_t(rho^|S|*f(S)^2) = 1
[1 - sigma_S_<=_t(rho^|S|*f(S)^2)] = sigma_S_>_t(rho^|S|*f(S)^2)
Since rho < 1:
epsilon = sigma_S_>_t_((rho^|S|*f(S))^2) < epsilon_stable = sigma_S_>_t_(rho^|S|*f(S)^2) , |S| > t
epsilon_stable can also be written as = sigma_S_<_t_(1-rho^|S|*f(S)^2), |S| < t

s = sigma[f(S)^2] * 2^(t/k)^1/(d-1)) , t=O((log s/epsilon)^(d-1)), |S| > t
s_stable = [1-sigma_S_<_t(rho^|S|*f(S)^2)]*2^((t/k)^1/(d-1)) , t=O((log s_stable/epsilon_stable)^(d-1)), |S| < t
(or)
s_stable = sigma(rho^|S|*[fourier_coeff(S)]^2)*2^[(t/k)^1/(d-1)] , t=O((log s_stable/epsilon_stable)^(d-1)), |S| > t.
s_stable is the size of the 100% noise stable circuit for f.
From above, t with noise operator is:
t(noise_operator)=O(log s/sigma(rho^|S|*f(S)^2)), |S| > t (or)
t(noise_operator)=O(log s/(1-sigma_S_<_t(rho^|S|*f(S)^2))), |S| < t
which is substituted for t in s_stable above and t without noise operator is:
t=O(log s/sigma(f(S)^2)), |S| > t
which implies that t(noise_operator) > t since denominator summation of fourier coefficients is attenuated by rho^|S| i.e the lowerbound of t in exponent for size increases due to noise operator. This should be the case for both expressions of s_stable above using Plancherel version of stability. This has strong relation to non-uniformity as rho is set of pseudorandom number strings for flip corresponding to values of rho where stability=1 which are advice strings to the circuit. The exponent t(noise_separator) increases as rho in denominator of log(s/epsilon_stable) increases - correlation due to random flip increases - implying that circuit size depends on increased lowerbound for log(s/epsilon_stable) which is a logarithmic increase in exponent that becomes a polynomial increase in size (because of 2^log()). Thus 100% stability requires polynomial increase in circuit size which implies a P/poly circuit for 100% noise stability. For example, if LHS of P(Good) has Percolation circuit in P/Poly, with polynomial increase in size 100% noise stability can be obtained. If LHS of P(Good) has percolation circuit in NC/poly, polynomial increase in size for 100% noise stability would still be in NC/poly, with additional gates adding to depth with bounded fanin. Moreover, from (53.8), an LHS with 100% noise-stable NC/poly circuit (if percolation is in NC/poly and not P/poly) for an RHS BPP-complete (i.e RHS binomial summation need not converge with 100% noise stability) means BPP is in P and P=BPP.

*Implication 1 of above: If RHS is in BPP (e.g a 3-SAT voter judge boolean function circuit with < 100% noise stability or bounded error) with polynomial size additional gates, it becomes 100% noise stable i.e a BPP circuit becomes NP-complete instance. But BPP is already known to be in P/poly and hence BPP has polynomial size circuits. Hence size(BPP_circuit) + polynomial size addition for stability is still a polynomial = P/poly circuit which makes RHS to be 100% noise stable NP-complete from <100% BPP problem. This implies NP has P/poly circuits surprisingly and NP in P/poly implies PH collapses to second level sigma(pi,2), and also AM=MA.

*Implication 2 of above: If LHS of P(Good) has BP.(NC/log) circuit due to <100%

noise stability, with polynomial size increase it can be derandomized or denoisified to 100% noise stable circuit. NC/log is in P/log. With polynomial size increase P/log is still P/log. If RHS is BPP it can be denoisified as mentioned in Implication 1 to NP-complete instance. Thus both LHS and RHS converge in P(Good) binomial summation and RHS is NP-complete and LHS is P/log. Thus NP is in P/log implying P=NP.

Points 53.4, 53.5, 53.6 and 53.7 describe P/poly, NC/poly and NC/log circuits for Percolation Boolean Functions subject to 100% noise stability regime. Important disclaimer is that above considers only denoisification and just denoisification may not be sufficient to remove errors in BP* circuits. As described in 14 on "Noise Stability and Complexity Class BPP" table of error scenarios, there might be cases where both denoisification(removing errors in column three) and derandomization(removing errors in column two) might be required that could add to the circuit size. But since BPP is in P/poly (Adleman's theorem) this is just another polynomial overhead in size. Thus increase in size due to denoisification + derandomization = poly(n) + poly(n) = poly(n) in P/poly.

Noise sensitive NC/log percolation circuit in LHS of P(Good) is in BP(NC/log) which requires denoisification and derandomization. Is BP(NC/log) in P/log? If yes, this could imply P=NP when RHS is an error-free NP-complete k-SAT instance (denoisified and derandomized BPP). BP(NC/log) can be denoisified with poly(n) additional gates from derivation above from HMLN theorem which makes NC/log to be in P/log. Next step of derandomization is by naive enumeration of all possible pseudorandom advice strings. Assumption here is that length of pseudorandom strings required for derandomizing percolation is log(n) which is sufficient to specify any point on an axis in the percolation grid. If log(n) bit pseudorandom advice is enumerated there are 2^(logn)=n possible pseudorandom strings which is poly(n) addition in circuit size again. P/log circuit is executed for all 2^(logn) psuedorandom strings which makes the total runtime n*poly(n)=poly(n). Thus derandomization takes as input a denoisified P/log circuit , adds poly(n) gates to it and outputs a circuit that has runtime poly(n) with logarithmic advice i.e P/log. This has a strange consequence that NP can have P/log circuits and P=NP (Implication 2) contrary to popular belief if logarithmic advice is valid. Equivalently for error-free , denoisified and derandomized PH-Complete problems in RHS (assuming PH-complete problems exist - from (53.2) above), denoisified and derandomized P/log circuit in LHS implies P=PH - collapse of polynomial hierarchy.

(THEORY-*IMPORTANT*) 53.12 Hastad-Linial-Mansour-Nisan (HLMN) Theorem and Noise Stable boolean functions - Denoisification 2

Above version derived by rewriting HLMN in terms of s(size) and t(size of fourier basis) is not too appealing. Basically, rate of change of circuit size with respect to noise is required and may not be necessary if denoisification is a subproblem of larger derandomization. It then reduces to the question of how size of circuit increases with derandomization, specifically for BPNC percolation circuits - Since BPP is in P/poly, is BPNC in NC/poly or NC/log. From Linial-Mansour-Nisan Theorem - Lemma 7 (Main Lemma) - http://www.ma.huji.ac.il/~ehudf/courses/anal09/LMN.pdf - Circuit size and Fourier coefficients are related as:

For |S| > t, Summation(fourier_coeff(S)^2) <= 2*(Size)*2^(-t^(1/d)/20)

For Noise-operated Fourier Spectra, the summation is augmented with rho(noise operator):

For |S| > t, Summation(rho^|S|*fourier_coeff(S)^2) <= 2*(Size)*2^(-t^(1/d)/20)

Applying Plancherel theorem and stability=1:

```
    | For |S| < t, 1-Summation(rho*fourier_coeff(S)^2) <=
2*(Size)*2^(-t^(1/d)/20)                                  |
    | => [1-Summation(rho^|S|*fourier_coeff(S)^2)] * 2^((t^(1/d)/20)-1) <= Size
                                  |
```

which is the circuit size lowerbound in terms of noise probability and fourier coefficients. As rho increases from 0 to 1, lowerbound for size decreases, that is, low noise requires high circuit size. Partial Derivative of size bound wrt rho:

```
    -Summation(fourier_coeff(S)^2)] * 2^((t^(1/d)/20)-1) <= doe(Size)/doe(rho)
```

which points to exponential - in t and d - decrease in size (because t=O(n) and d=O(function(n)) with linear increase in rho (noise) due to negative sign and vice versa. Consequences of this are significant. Though derandomization could result in polysize circuits, denoisification does not necessarily add poly(n) size and in worst case could be exponential too. For NC circuits, t=O(n) and d=O(polylog(n)) , denoisification results in 2^O(n^(1/polylog(n))) increase in size which is superpolynomial. Stability=1 is the point when LHS and RHS of P(Good) binomial summation converge i.e both sides are equally efficient and are perfect voter boolean functions not affected by input noise. Therefore BPNC can not be denoisified in polynomial circuit size even if it can be derandomized to be in NC with polynomial size (i.e if BPNC is in NC/poly). Thus percolation NC circuit in LHS cannot be made 100% stable with denoisification with mere polynomial size and has superpolynomial size lowerbound. This implies when RHS is also 100% stable circuit size of RHS is atleast superpolynomial. Rephrasing:

```
      | P(Good) binomial summation in its circuit version can not have
polynomial size circuits           |
      | when LHS and RHS converge with 100% noise stability.
                       |
```

For example, if LHS is a percolation logdepth circuit and RHS is single voter with 3-SAT NP-complete circuit, when LHS and RHS are both 100% noise stable and derandomized (rho=0):

```
      LHS circuit size >= O(2^n^(1/polylog(n))) = superpolynomial,
quasiexponential lowerbound, depth is polylog(n)
```

implying LHS percolation circuit could become superpolynomial sized when noise stability is 1 and not in NC despite being logdepth.

```
      RHS circuit size >= O(2^n^(1/d)) = higher superpolynomial,
quasiexponential lowerbound, depth is arbitrary ,could be constant for 3-CNF
```

and thus LHS and RHS circuits are equally efficient algorithms with differing circuit sizes. As RHS is NP-complete, LHS is size lowerbound for RHS - LHS has lesser superpolynomial size than RHS where d(depth of RHS which could be a constant 3) < polylog(n)(depth of LHS).
[Open question: Does this imply superpolynomial size lowerbounds for NP and thus P != NP]

Reference:

53.12.1 Derandomizing BPNC - [Periklis Papakonstantinou] - http://iiis.tsinghua.edu.cn/~papakons/pdfs/phd_thesis.pdf
53.12.2 Derandomization basics - [Salil] - Enumeration - http://people.seas.harvard.edu/~salil/pseudorandomness/basic.pdf
53.12.3 Simplified Hitting Sets Derandomization of BPP - [Goldreich-Vadhan-Wigderson] - http://www.wisdom.weizmann.ac.il/~oded/COL/gvw.pdf - where hitting sets are set of strings at least one element of which is accepted by any circuit that accepts atleast half of its inputs (i.e randomized algorithms in RP and

BPP). Hitting Set generators H expand k(n) to n so that a function f returns f(H(k(n)))=1 always where f is in RP or BPP.
53.12.4 Derandomizing BPNC - Existence of Hitting Set Generators for BPNC=NC - [Alexander E.Andreev, Andrea E.F.Clementi, Jose D.P.Rolim] - http://ac.els-cdn.com/S0304397599000249/1-s2.0-S0304397599000249-main.pdf?_tid=e56d3a1e-ad3d-11e5-a4ef-00000aacb35e&acdnat=1451291885_fbff7c6ae94326758cf8d6d7b7a84d7b
53.12.5 Linial-Mansour-Nisan Theorem - Lemma 7 (Main Lemma) - http://www.ma.huji.ac.il/~ehudf/courses/anal09/LMN.pdf:
For |A| > t, Summation(fourier_coeff(A)^2) <= 2*(Size)*2^(-t^(1/d)/20)
53.12.6 The Computational Complexity Column [Allender-Clementi-Rolim-Trevisan] - theory.stanford.edu/~trevisan/pubs/eatcs.ps

---

53.13. (THEORY) Partial Derivative of Circuit size wrt rho (noise probability) - increase in circuit size for Denoisification

---

From 53.11 above, size of denoisified circuit from HMLN theorem, and Stability in terms of Fourier coefficients with noise operator is:
s_stable = [1-sigma_S_<_t(rho^|S|*f(S)^2)]*2^((t/k)^1/(d-1)) , t=O((log s_stable/epsilon_stable)^(d-1)), |S| < t
(or)
s_stable = sigma(rho^|S|*[fourier_coeff(S)]^2)*2^[(t/k)^1/(d-1)] , t=O((log s_stable/epsilon_stable)^(d-1)), |S| > t.
Lower bound of exponent increases as sigma(rho^|S|*f(S)^2) decreases in denominator of exponent subject to stability constraint:
[1 - sigma_S_<=_t(rho^|S|*f(S)^2)] = sigma_S_>_t(rho^|S|*f(S)^2)
which implies that sigma_S>t_(rho^|S|*f(S)^2) decreases when sigma_S<t_(rho^|S|*f(S)^2) increases. This is Consequence of HMLN theorem that t acts as an inflexion point for Fourier spectrum with noise operator - area integral from 0 to t is concentrated greater than area integral from t to n.

Partial derivative of Circuit size wrt to rho can be derived as:
delta_s/delta_rho = sigma_S>t_(|s|*rho^(|s|-1) * f(S)^2) / (1 - sigma_S<t_(rho^|S|*f(S)^2)) where delta_s is increase in size of circuit with delta_rho increase in noise probability. When rho=0, increase in circuit size is 0 (obvious base case). When rho=1, which is the worst case, increase in circuit size is = sigma_S>t_(|S|*f(S)^2) / f(S)^2 which is upperbounded by n - poly(n) increase in size.

53.14. (THEORY) Hastad-Linial-Mansour-Nisan Theorem, Circuit size, Noise Stability and BP* classes

---

Percolation circuit defined previously has:
- Sorting Network for creating path sequentially on grid (NC logdepth and polysize)
- Comparators (Constant depth and polysize) are required for comparing the leftmost and rightmost coordinates.
Together Percolation is in non-uniform NC. This circuit depends on number of points on the left-right path which is poly(n).
Therefore percolation is in NC/poly because size of list of advice strings could be O((n^2)log(n^2)) and it may not have NC/log circuits.
From Linial-Mansour-Nisan Theorem - Lemma 7 (Main Lemma) - http://www.ma.huji.ac.il/~ehudf/courses/anal09/LMN.pdf:
For |A| > t, Summation(fourier_coeff(A)^2) <= 2*(Size)*2^(-t^(1/d)/20)
With noise operator, noise stability=1 from Plancherel's theorem:

| For |S| < t, 1-Summation(rho^|S|*fourier_coeff(S)^2) <= 2*(Size)*2^(-t^(1/

d)/20) |
    | Corollary: For |S| > t, Summation(rho^|S|*fourier_coeff(S)^2) <=
2*(Size)*2^(-t^(1/d)/20) |
    | => [1-Summation(rho^|S|*fourier_coeff(S)^2)] * 2^((t^(1/d)/20)-1) <= Size
                                  |

When denoisified, rho=0 and lowerbound for size:
      [1-(0^0*f(phi)^2+0+0+..)]*2^((t^(1/d)/20)-1) <= size
      [1-f(phi)^2]*2^((t^(1/d)/20)-1) <= size (as all other coefficients vanish)
when t=n, the size is lowerbounded by :
      [1-f(phi)^2]*2^((n^(1/d)/20)-1)
which is superpolynomial. This implies an NC/poly polysize circuit when
denoisified becomes O(2^((n^(1/polylog(n)/20)-1) .

From the table that describes relation between Noise and Error(BP* and PP) in 14
previously, it is evident that Noise intersects Error
if following is drawn as a Venn Diagram i.e. Noise stability solves just a
fraction of total error, and remaining requires derandomization.
In other words, Noisy circuit could be error-free and Noise-free circuit could
have error.

---
      x | f(x) = f(x/e) | f(x) != f(x/e) Noise
|

---
x in L, x/e in L | No error | Error |

---
x in L, x/e not in L | Error | No error if
f(x)=1,f(x/e)=0 |
                              | | else Error |

---
x not in L, x/e in L | Error | No error if
f(x)=0,f(x/e)=1 |
                              | | else Error |

---
x not in L, x/e not in L | No error | Error
      |

Noise percolation circuit in P(Good) summation converges in noise stability
regime:
      lt Pr[f(w(n)) != f(w(n)/e)] = 0, e=1/(alpha4*n^2)
      n -> inf
Infinite f(n) is representable as Non-uniform NC/poly circuit though undecidable
because turing machine computing the above limit is not in recursive but
recursively enumerable languages.

For a 3-CNFSAT boolean function with noise, size of circuit is lowerbounded as:
     [1-Summation(rho^|S|*fourier_coeff(S)^2)] * 2^((t^(1/d)/20)-1) <= Size ,
for constant depth d.
For rho=1,
     [1-Summation(fourier_coeff(S)^2)] * 2^((t^(1/d)/20)-1) <= Size , for
constant depth d, |S| < t.

Gist of the above:
53.14.1 Size of percolation logdepth circuit (with noise) rho=1:
      (f([n])^2) * 2^(n^(1/polylog(n-1)/20)-1) <= size

53.14.2 Size of depth d circuit (with noise) rho=1:
      (f([n])^2) * 2^((n-1)^(1/d/20)-1) <= size
53.14.3 Size of percolation logdepth circuit (with noise) rho=0:
      (1-f([phi])^2) * 2^(n^(1/polylog(n)/20)-1) <= size, phi is empty set
53.14.4 Size of depth d circuit (with noise) rho=0:
      (1-f([phi])^2) * 2^(n^(1/d/20)-1) <= size, phi is empty set

---

53.15 (THEORY) Special Case of HLMN theorem for PARITY as 3-SAT and counterexample for polynomial circuit size

---

From HMLN theorem:
     Summation(fourier_coeff(S)^2) * 2^((t^(1/d)/20)-1) <= Size , for constant depth d, |S| > t.
When t=n-1, fourier series has only coefficient for the parity set that has all variables:
      fourier_coeff([n])^2 * 2^(((n-1)^(1/d)/20)-1) <= size, for constant depth d.
For a circuit of size polynomial (n^k) and depth 3:
      summation(fourier_coeff([n])^2) <= n^k / 2^((t^(1/3)/20)-1).
and for t=n-1:
      fourier_coeff([n])^2 <= n^k / 2^(((n-1)^(1/3)/20)-1).
From definition of fourier coefficients:
      fourier_coeff([n]) = summation(f(x)*(-1)^(parity(xi)))/2^n for all x in {0,1}^n
Maximum of fourier_coeff([n]) occurs when f(x)=1 for (-1)^parity(xi) = 1 and when f(x)=-1 for (-1)^parity(xi) = -1 together summing to 2^n.
      Maximum(fourier_coeff([n])) = 2^n/2^n = 1 (this implies all other fourier coefficients are zero)
Which happens when f(x) is equivalent to Parity function.
An example PARITY as 3-CNFSAT from DIMACS32 XORSAT [Jing Chao Chen - http://arxiv.org/abs/cs/0703006] that computes parity of n=3 variables x1,x2,x3 is:
      (x1 V not x2 V not x3) /\ (not x1 V x2 V not x3) /\ (not x1 V not x2 V x3) /\ (x1 V x2 V x3)
Previous PARITY SAT accepts odd parity strings. Complementing variables in each clause makes it accept even parity:
      (not x1 V x2 V x3) /\ (x1 V not x2 V x3) /\ (x1 V x2 V not x3) /\ (not x1 V not x2 V not x3)
Assumption: PARITY as 3-CNFSAT has polynomial size.
PARITY 3-SAT is satisified by odd parity bit strings and is NP-complete because this is search version of parity - it searches for strings that have odd parity. Complementing each clause would make it satisfy even parity bit strings. This example PARITY as 3-SAT with variables in each clause complemented maximizes the fourier coefficient to 1 at S=[n].For |S| < n by pigeonhole principle if one coefficient at |S|=n is enough to make it 1 others have to be either all zero or cancel out each other. An extension of this 3-CNF for arbitrary n should be recursively obtainable from the above example by XOR-ing with additional variables and simplifying and also by Tseitin Transformation (https://en.wikipedia.org/wiki/Tseytin_transformation) which converts arbitrary combinatorial logic to CNF.
=>   1^2 <= n^k / 2^(((n-1)^(1/3)/20)-1) which is a contradiction to assumption (denominator is exponential) that PARITY 3-SAT has polynomial size (n^k) circuits.

From above, for some circuit size of PARITY as 3-CNFSAT, there exists t(=O(n)) such that:
      2^(((t)^(1/3)/20)-1) <= size / summation(f(S)^2) , |S| > t
      2^(((t)^(1/3)/20)-1) <= size (for all t < n-1, fourier coefficients of PARITY 3-SAT are 0, and this bound is meaningful only for t=n-1).
i.e for all values of t, circuit size is bounded from below by a quantity exponential in function of n (which could be O(n), W(n) or Theta(n).If this is

O(n), might imply a lowerbound by upperbound - reminiscent of Greatest Lower Bound in a Lattice if Complexity Classes form a Lattice). LHS of the inequality can have values of t from 1 to n-1 and only one value of this can be chosen lowerbound. If LHS is not a function of n, it leads to a contradiction that 2 PARITY 3SAT circuits of varying sizes can have constant lowerbound. Also finding Minimum Circuit Size problem is believed to be in NP but not known to be NP-Complete(http://web.cs.iastate.edu/~pavan/papers/mcsp.pdf).
(How can a special case of NP-complete problem have superpolynomial size circuits? Is this a counterexample and thus implies superpolynomial size circuits for all problems in NP and P != NP. Because of this correctness of above counterexample has to be reviewed)

References:

53.15.1 Circuit Complexity Lecture notes - [Uri Zwick] - http://cs.tau.ac.il/~zwick
53.15.2 Lecture notes - [Nader Bshouty] - http://cs.technion.ac.il/~bshouty

---
53.16 (THEORY) P/poly, Percolation and PARITY as 3-CNFSAT in 53.15

53.16.1 NP in P/poly (NP having polynomial size circuits) implies AM=MA and PH collapses to sigma2 /\ pi2.
53.16.2 PARITYSAT in 53.15 which is a special case of NP-complete has superpolynomial size circuits.
53.16.3 If Percolation is in NC/poly and both LHS and RHS of P(Good) converge to 100% noise stability, does it imply an NC/poly lowerbound for arbitrary circuit size in RHS? In other words what does it imply when 2 circuits of varying size have equal stability - Do two boolean circuits of equal decision error (NoiseStability + or - delta) with different sizes imply a lowerbound? E.g NC/poly circuit for PH=DC (if PH-complete) circuit implying PH in NC/poly.

53.16.3.1 If equal error (NoiseStability +/- delta) with different circuit sizes implies a lowerbound:

There is an inherent conflict between 53.16.1, 53.16.3 which are conditions for polynomial size circuits for NP and 53.16.2 which implies superpolynomial circuit size for a special case of NP-complete 3-SAT problem. This contradiction is resolveable if following is true:

- LHS of P(Good) is percolation NC/poly circuit in 100% noise stability regime.
- RHS of P(Good) is non-percolation circuit of arbitrary size (e.g. DC circuit of exponential size , superpolynomial size NP circuit from 53.15 and 53.16.2 etc.,).
- LHS NC/poly 100% noise stability implies a lowerbound when RHS (e.g superpolynomial size NP circuit from 53.15 and 53.16.2) is also 100% noise stable contradicting superpolynomial size lowerbound for NP from 53.16.2. Contradiction is removed only if RHS of P(Good) is never 100% noise stable and hence LHS is not equatable to RHS i.e P(Good) summation has to be divergent and never converges to 100% while LHS is 100%. Major implications of this divergence are:
- the individual voter error/stability cannot be uniform 0.5 or 1 across the board for all voters - voters have varied error probabilities.
- varied stability/sensitivity per voter implies RHS majority voting is Poisson Distribution.
- for probabilities other than 0.5 and 1, RHS P(good) summation becomes hypergeometric series requiring non-trivial algorithms which might or might not converge.
- This assumes that RHS does not have a boolean function that is 100%

noise stability regime similar to LHS Percolation.
- Superpolynomial size for NP-complete PARITY3SAT (kind of a promise problem) implies P != NP and P != NP implies either a Poisson distribution or a hypergeometric series that might or might not converge.
In other words a democratic decision making which does not involve percolation is never perfect and thus BKS conjecture has to be true berating majority i.e there exists atleast one function - percolation - stabler than stablest majority.

53.16.3.2 If equal error (NoiseStability +/- delta) with different circuit sizes doesn't imply a lowerbound:

- LHS of P(Good) is percolation NC/poly circuit in 100% noise stability regime.
- RHS of P(Good) is of arbitrary size (e.g. superpolynomial size NP circuit from 53.16.2).
- LHS NC/poly 100% noise stability doesn't imply a lowerbound when RHS (e.g superpolynomial size NP circuit from 53.16.2) is also 100% noise stable. It doesn't contradict convergence of P(Good) summation to 100%, superpolynomial size lowerbound for NP from 53.16.2 and BKS conjecture has to be trivially true i.e Both LHS and RHS of P(Good) are equally stable.

53.17 (THEORY) Oracle results and P=NP and PH

From [Baker-Gill-Solovay] there are oracles A and B such that P^A = P^B and P^A != P^B implying natural proofs involving relativizing oracles cannot answer P=NP question. Also PH is shown to be infinite with oracle access recently. Because of this all points in this document avoid oracle arguments and only use fourier series of boolean functions and boolean function compositions for circuit lowerbounds for P(Good) voting circuits.

53.18 (THEORY) P(Good) circuit special case counterexample in PH collapse

---

LHS of P(Good) - a sigma(p,k) hard (could be dictatorial or PRG) circuit with 100% (noise stability +/- delta) (assuming such exists)
RHS of P(Good) - a sigma(p,k+1) hard majority voting circuit with 100% (noise stability +/- delta) (assuming such exists)

If above converging P(good) LHS and RHS imply a lowerbound , LHS sigma(p,k) lowerbounds RHS sigma(p,k+1) implying PH collapses to some level k.
The matrix of noise versus BPP in 53.14 and 14 imply noise and BPP intersect, but if only classes in (sigma(p,k) - BPP) are chosen for voter boolean functions, derandomization is not required and noise stability is sufficient to account for total error. If there is a PH-complete problem, PH collapses to some level,but implication in other direction is not known i.e collapse of PH implying PH-completeness - if both directions are true then this proves non-constructively that there exists a PH-complete problem. Every level k in PH has a QBFSAT(k) problem that is complete for sigma(p,k) class. To prove PH-completeness from PH-collapse, all problems in all PH levels should be reducible to a PH-complete problem i.e all QBFSAT(k) complete problems should be reducible to a QBFSAT(n)-complete problem for k < n. Intuitively this looks obvious because QBFSAT(n) can generalize all other QBFSAT(k) by hardcoding some variables through a random restriction similar to Hastad Switching Lemma.

Equating sigma(p,k) with sigma(p,k+l) is probably the most generic setting for

P(Good) circuits discounting arithmetic hierarchy.

Example sigma(p,k)-complete QBFSAT(k) boolean function:
      there exists x1 for all x2 there exists x3 ... QBF1(x1,x2,x3,...,xk)

Example sigma(p,k+1)-complete QBFSAT(k+1) boolean function:
      there exists x1 for all x2 there exists x3 ...
QBF2(x1,x2,x3,...,xk,x(k+1))

53.19 (THEORY) Depth Hierarchy Theorem for P(Good) circuits

Average case Depth Hierarchy Theorem for circuits by [Rossman-Servedio-Tan] - http://arxiv.org/abs/1504.03398 - states that any depth (d-1) circuit that agrees with a boolean function f computed by depth-d circuit on (0.5+o(1)) fraction of all inputs must have exp(n^omega(1/d)) size. This theorem is quite useful in election forecast and approximation where it is difficult to learn a voter boolean function exactly.

53.20 (THEORY) Ideal Voting Rule

[Rousseau] theorem states that an ideal voting rule is the one which maximizes number of votes agreeing with outcome [Theorem 2.33 in http://analysisofbooleanfunctions.org/]. The ideal-ness in this theorem still doesn't imply goodness of voting which measures how "good" is the outcome rather than how "consensual" it is - fact that majority concur on some decision is not sufficient to prove that decision is correct which is measured by noise stability +/- delta. Because of this P(good) summation is quite crucial to measure hardness alongwith goodness.

53.21 (THEORY) Plancherel's Theorem, Stability polynomial and Lowerbounds in 53.18

Stability polynomial can be applied to QBFSAT(k) and QBFSAT(k+1) in 53.18.

By Plancherel's Theorem, Stability of a boolean function with noise rho is written as polynomial in rho:
      Stability(f) = sigma(rho^|S|*f(S)^2) , S in [n]
which can be equated to 1 to find roots of this polynomial - noise probabilities - where 100% stability occurs.

From HLMN theorem for sigma(p,k)-complete QBFSAT(k):
      sigma(f(A1)^2) <= 2*(M1)*2^(-t1^(1/d1)/20) , |A1| > t1

From HLMN theorem for sigma(p,k+1)-complete QBFSAT(k+1):
      sigma(f(A2)^2) <= 2*(M2)*2^(-t2^(1/d2)/20) , |A2| > t2

When noise stability is 100%:
      Stability(QBFSAT(k)) = sigma(rho1^|S1|*f(S1)^2) = 1, S1 in [n]
      Stability(QBFSAT(k+1)) = sigma(rho2^|S2|*f(S2)^2) = 1, S2 in [n]
where rho1 and rho2 are variables each in the polynomial for QBFSAT(k) and QBFSAT(k+1). Solving these polynomials gives the points where 100% stability occurs in both QBFSATs. By choosing non-zero, real, positive roots of these polynomials as noise probabilities, 100% stability can be attained for QBFSAT(k) in LHS and QBFSAT(k+1) in RHS. Noise probabilities need not be equal in LHS and RHS. This is not an average case bound but lies as special case somewhere

between best case and worst case. If delta due to BPP is ignored, equal stability implies lowerbound - LHS sigma(p,k) lowerbounds RHS sigma(p,k+1) and PH collapses to level k. If collapse implies completeness as mentioned in 53.18 , this suffices to prove existence of a PH-complete problem non-constructively. Thus Stability implying Lowerbound has enormous implications for separation results.

Above deduction on PH collapse and possible PH-completeness contradicts 53.15 counterexample for NP having superpolynomial size circuits and hence P !=NP, because PH-completeness could imply P=PH (when LHS of P(Good) is an [NC/log in P] percolation circuit - unlikely if percolation has only NC/poly circuits - which is 100% stable and RHS is PH-complete and 100% stable) and therefore P=NP. Resolution of this contradiction requires:

- Equal decision error (stability +/- delta) does not imply lowerbound.
- Roots of sigma(rho^|S|*f(S)^2) - 1 = 0 can only be complex with Re+Im parts and can't be real.
- sigma(rho^|S|*f(S)^2) - 1 = 0 is zero polynomial where all coefficients are zero.
- PH-collapse does not imply PH-completeness.

Stability polynomial above with rho as variable can be applied to any circuit to find the noise probabilities where 100% stability is attained without machinery like size and depth hierarchies, if equal stability implies lowerbounds. Indirectly 100% stability implies size bound - when rho=1, Stability polynomial grows and concides with HLMN theorem for circuit size lowerbound as mentioned above.

54(THEORY). Would the following experimental gadget work? It might, but impractical:
A voter gadget spawns itself into liker, disliker and neutral parallel threads on an entity to be judged. The neutral thread gathers inputs from liker and disliker threads and gets a weighted sum on them to classify the entity as "like-able" or "dislike-able". Thus real-world classification or judgement and perception seem mythical. (And how is weightage decided? Can this be using SentiWordNet score?). Theoretically, this is similar to Judge Boolean Functions defined in 53.* above, specifically when a voter decides by interactive proof system adversarial simulation (likes and dislikes are reducible to provers and verifiers) which is EXPTIME-complete. Implementation of this gadget is thus EXPTIME-complete.

55(THEORY). Evocation is a realworld everyday phenomenon - a word reminding of the other. Positive thought chooses the right evocation and a negative thought chooses the negative evocation in the absence of context to disambiguate. For example, for a pessimist "fall" means "falling down" while for an optimist "fall" could mean "windfall or sudden gain". For a pestimist((optimist+pessimist)/2), fall could mean one of the "seasons" or even nothing. Mind sees what it chooses to see. Pessimism and Optimism are probably mathematically definable as accumulated weighted thoughts of past occurrences and any future thought or decision is driven by this "accumulated past" which acts as an implicit "disambiguator". This accumulated past or a Karma is reminiscent of above hypergraph of class stacks.

56(THEORY). A crude way to automatically disambiguate is to lookup the corresponding class stack and analyze only bounded height of events from the top of the stack. Bounded height would imply analyzing only a small window of the past events. For example, looking up the class stack from top to bottom for limited height of 2 for "fall" might have hyperedges "tree fall, heavy rain fall". Thus automatic disambiguation context could be the tuple of tuples [[tree], [heavy, rain]] gathered from each hyperedge analyzed from top. Then this tuple of tuples has to be weighted to get majority. Alternatively, each tuple in this tuple set, as a random variable, could be assigned a belief potential or a conditional probability of occurrence based on occurrence of

other tuples in the set. Thus a Bayesian Belief Network can be constructed and by propagating belief potential, probability of each tuple can be computed. Most disambiguating tuple is the one with high belief propagation output. Alternatively, elements of the set of tuples above can be construed as Observations and the State - the most disambiguating tuple - is Hidden and needs to be measured. Thus a Hidden Markov Model can be built. Emission probabilities for an Observation at a State are to be given as priors - probability of a tuple being observed for a word or "class". Importantly each stack is grown over time and is a candidate for Markov process with the restriction - node at height h is dependent only on node at height (h-1) - height encodes timestamp. (But this makes the inference semi-statistical). Consequently, this gives a Trellis from which a Viterbi path can be extracted. Forward-Backward probabilities can be computed (Either generative or discriminative model can be applied, generative being costlier) using this Markov property and deeming the stack itself as a dynamically grown Hidden Markov Model where the Observations are the tuples(hyperedges) and the hidden state is the document that disambiguates. Using the Forward-Backward algorithm, the State(hyperedge) which maximizes the Probability of ending up in that State is the most disambiguating document hyperedge = argmax [Pr(State(t)=Hyperedge(e) / ObservedTuples(1..t)] i.e The tuple or hyperedge that has highest forward conditional probability in the Trellis transition. This assumes the Hypergraph node stack as a Markov Chain. Moreover, State need not be a hyperedge. State is rather a "meaningful context". As in above example, uttering "fall" would kickoff a Forward-Backward computation on the Hypergraph node class stack for "fall" considering the stack as a Hidden Markov Model and would output the argmax State (which is not limited to hyperedge) as above. (Handwritten illustration at: https://sites.google.com/site/kuja27/NotesOnConceptHypergraphHMM_and_ImplicationGraphConvexHulls_2013-12-30.pdf?attredirects=0&d=1). Above assumption that node at height (h-1) implies node at height h is quite experimental and is just to model how the human neuropsycological process of "evocation" works on uttering a word. There is no real-life application to this - an event at height (h-1) need not imply an event at height h.

57(THEORY). Thus Sentiment Mining techniques can be used for the above hypergraph to get the overall sentiment of the hypergraph past. Probably each hyperedge might have to be analyzed for positive or negative sentiment to get grand total. Citation graph maxflow in http://arxiv.org/abs/1006.4458 already gives a SentiWordnet based algorithm to weight each edge of a citation graph. Citations can be generalized as "like" or "dislike" opinions on entities as mentioned above.

58(THEORY). Thus not only documents but also events in human life can be modelled as above hypergraph to get a graphical model of the past and a PsychoProfiling of an individual could be arrived at using Sentiment mining. Thus, item 18 about "AstroPschoAnalysis" is feasible by two parallel paths which converge - Psychoprofiling by above concept or event hypergraph for a human being and equating it with results from Astronomical+Astrological analysis done through String Multiple Sequence Alignment mining.

##
##
59. Initial Design Notes for - Mundane Predictive Model:
##
##

- (FEATURE - DONE) DecisionTree, NaiveBayes and SVM classifiers and horoscope encoders are already in the AstroInfer codebase.
- (FEATURE - DONE) Encode the dataset which might be USGS or NOAA(Science on a Sphere) datasets or anyother dataset available after getting text horos for these using Maitreya's Dreams textclient (AstroInfer version)
- (FEATURE - DONE) Above gives encoded horoscope strings for all classes of mundane events in both Zodiacal and AscendantRelative formats (autogenerated by Python scripts in python-src/)
- (FEATURE - DONE) Above set is a mix of encoded strings for all events which

can be classified using one of the classifiers above.
- (FEATURE - DONE) For each class the set of encoded horo strings in that class can be pairwise or multiple-sequence aligned to get the pattern for that class
- (FEATURE - DONE) There are two sets for each class after running the classifiers - one set is AscendantRelative and the other is Zodiacal
- (FEATURE - DONE) The above steps give the mined astronomical patterns for all observed mundane events - Pattern_Mundane_Observed_AscRelative and Pattern_Mundane_Observed_Zodiacal

60. (FEATURE - DONE) Above has implemented a basic Class and Inference Model that was envisaged in 2003. Class is the set-theoretic notion of sets sharing common underlying theme. Using VC Dimension is a way to determine accuracy of how well the dataset is "shattered" by the classes.

61. (THEORY) Now on to mining the classics for patterns: For all classes of events, running the classifier partitions the rules in a classic into set of rules or patterns for that class. Here again there are two sets for each class - Pattern_Mundane_Classic_AscRelative and Pattern_Mundane_Classic_Zodiacal

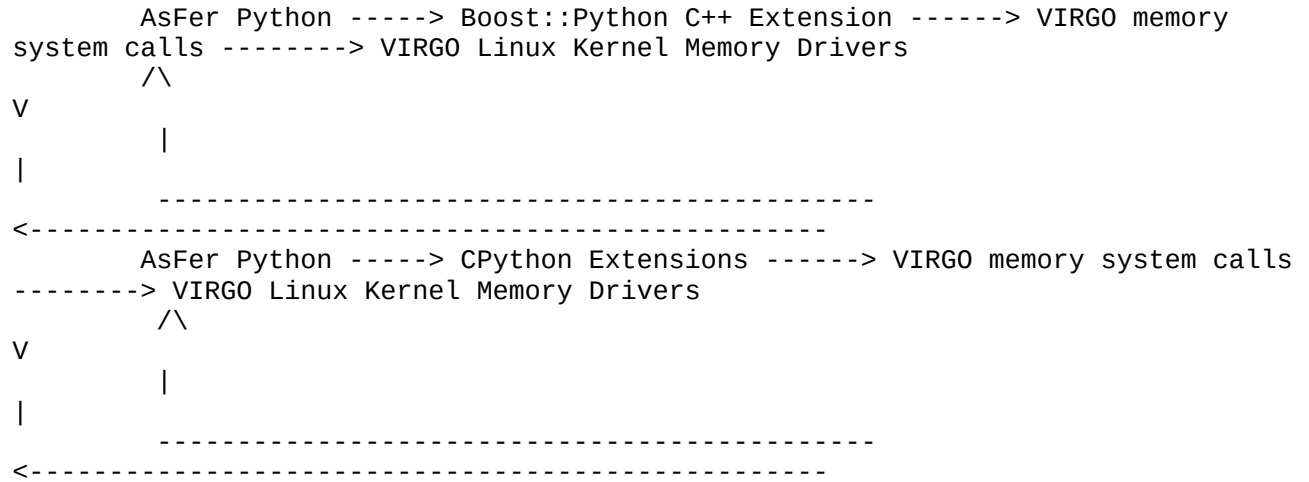
62. (THEORY) Thus correlation of the two sets *_Observed_* and *_Classic_* (each set has 2 subsets) for percentage similarity using any known similarity coefficient would unravel any cryptic pattern hidden astronomical datasets for that event and would be a circumstantial and scientific proof of rules in astrological classics with strong theoretical footing and would also bring to light new rules earlier unknown.

63. (THEORY and IMPLEMENTATION) More importantly, for example, if a non-statistical,astrological model of rainfall is computed based on a classical rule which says that "Venus-Mercury in single sign or Venus-Sun-Mercury in close proximity degrees with Sun in the middle causes copious rainfall" is used as a model, expected output of the model could be "Between 28October2013 and 15December2013 there could be peak monsoon activity in ----- longitude ---- latitude with --- percentage probability". And thus this could be a Medium Range Weather Forecasting tool. A python script that searches the Astronomical Data for mined rules from Sequence Mining of Astronomical Data has been added to python-src/. It uses Maitreya's Dreams Ephemeris text client for getting astronomical data for range of date, time and longitude and latitude. This script prints matching date, time and longitude and latitude when a rule for a weather phenomenon occurs as mined by sequence mining.

64. (FEATURE - DONE-related to 65 and 139) Integrate AstroInfer,USB-md,KingCobra and VIRGO into an approximately intelligent cloud OS platform that not just executes code statically but also dynamically learns from the processes (say through log files, executables, execution times etc., and builds predictive models).
USB-md Design Document:
http://sourceforge.net/p/usb-md/code-0/HEAD/tree/USBmd_notes.txt
VIRGO Design Document:
http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VirgoDesign.txt
KingCobra Design Document:
http://sourceforge.net/p/kcobra/code-svn/HEAD/tree/KingCobraDesignNotes.txt
This AsFer+USBmd+VIRGO+KingCobra would henceforth be known as "Krishna iResearch Intelligent Cloud OS - NeuronRain "

65. (THEORY and IMPLEMENTATION) Related to point 64 - Software Analytics - source code and logs analytics - related to Program comprehension - point 139 on BigData Analytics has more on this. Recently added VIRGO kernel_analytics module is in a way a software analytics module which reads config file set by the machine learning software after mining the kernel logs and objects. Kernel Analytics VIRGO driver module at present reads key-value pairs written to by the AsFer Machine Learning code from /etc/virgo_kernel_analytics.conf. Optionally, AsFer code can directly modify the kernel tunable parameters learnt by AsFer (https://www.kernel.org/doc/Documentation/kernel-parameters.txt) through

modprobe or "echo -n ${value} > /sys/module/${modulename}/parameters/${parm}" for existing modules while virgo_kernel_analytics.conf is read for VIRGO specific modules. Graph Isomorphism of Program Slice Dependency Graphs for 2 codebases mentioned in https://sites.google.com/site/kuja27/PhDThesisProposal.pdf is also a Software Analytics problem. There are Slicing and Graph Isomorphism tools already available. Recently there has been a SubExponential Time algorithm for GI - [Lazlo Babai]. Kernel Analytics config in filesystem requires reloading. New feature to set the kernel analytics key-value pairs directly from userspace to kernel memory locations dynamically has been added via boost::python C++ and CPython extensions - described in 217.SATURN Program Analysis Framework has been integrated into VIRGO Linux - error logs of SATURN are AsFer analyzable - more on this in 232. SourceForge VIRGO repository does not contain SATURN .db files in saturn_program_analysis/saturn_program_analysis_trees/. They are committed only in GitHub saturn_program_analysis/saturn_program_analysis_trees/ . Spark code computing Cyclomatic Complexity of codebase from SATURN generated .dot files has been committed in python-src/software_analytics/ . This requires VIRGO Linux saturn program analysis driver generated .dot files.



References:

65.1 Analytics of Device Driver code - http://research.microsoft.com/en-us/groups/sa/drivermine_asplos14.pdf
65.2 Logging - http://research.microsoft.com/en-US/groups/sa/loggingpractice.pdf
65.3 Law of leaky abstraction - http://www.joelonsoftware.com/Articles/LeakyAbstractions.html
65.4 Function Point Analysis - http://www.bfpug.com.br/Artigos/Albrecht/MeasuringApplicationDevelopmentProductivity.pdf
65.5 Function Point Analysis and Halstead Complexity Measures - https://en.wikipedia.org/wiki/Halstead_complexity_measures
65.6 Cyclomatic Complexity of a Program - Euler Characteristic of program flow graph (E - V + 2) - Approximate code complexity.

66. (THEORY) Encoding a document with above Hypergraph of Class stack vertices and Hyperedges - This hypergraph of concepts can also be used as a steganographic encoding tool for obfuscating a document (somewhat an encryption of different kind). For example, each document is a hyperedge in this hypergraph of class stack vertices. By choosing a suitable encoding function every concept or a word can be replaced with some element within each class stack vertex. This encoding function chooses some element in each class stack - f(word or concept C1 in a document) = a different word or concept C2 in the class stack vertex

that contains C1. This function f has to be one-one and onto so that it is invertible to get the original document. This function can be anything modulo the height of that stack. Thus a plain text meaningful document can encrypt another meaningful hidden document without any ciphertext generation.

67. An automaton or Turing machine model for data or voice (or musical) samples - Voice samples or musical notations are discrete in time and occur in stream. As an experimental research, explore on possibility of automatically constructing an automaton or Turing machine that recognizes these languages (RE or CFG) where the notes are the alphabets.

(FEATURE - DONE) 68. Music Pattern Mining - Computational Music - Alternatively, a Discrete Fourier Transform on a set of data or voice samples gives a set of frequencies - a traditional DSP paradigm.

References:

68.1 https://ccrma.stanford.edu/~jos/st/

69. (FEATURE - Audacity FFT - DONE) Music Pattern Mining - Experimental Idea of previous two points is to mine patterns in discrete data and voice(music for example) samples. Music as an expression of mathematics is widely studied as Musical Set Theory - Transpositions and Inversions (E.g http://en.wikipedia.org/wiki/Music_and_mathematics, http://www.ams.org/samplings/feature-column/fcarc-canons, http://www-personal.umd.umich.edu/~tmfiore/1/musictotal.pdf). Items 56,57 and 58 could help mining deep mathematical patterns in classical and other music and as a measure of similarity and creativity. In continuation of 68, following experiment was done on two eastern Indian Classical audio clips having similar raagas:

69.1 FFT of the two audio files were done by Audacity and frequency domain files were obtained with sampling size of 65536, Hanning Window, Log frequency

69.2 FFTs of these two have been committed in music_pattern_mining/. Similarity is observed by peak decibel frequency of ~ 500Hz in both FFTs - Similarity criterion here is the strongest frequency in the sample though there could be others like set of prominent frequencies, amplitudes etc.,. Frequencies with low peaks are neglected as noise.

70-79 (THEORY - EventNet Implementation DONE) EventNet and Theoretical analysis of Logical Time:
---

70. In addition to Concept wide hypergraph above, an interesting research could be to create an EventNet or events connected by causation as edge (there is an edge (x,y) if event x causes event y). Each event node in this EventNet is a set of entities that take part in that event and interactions among them.This causation hypergraph if comprehensively constructed could yield insights into apparently unrelated events.This is hypergraph because an event can cause a finite sequence of events one after the other, thereby including all those event vertices. For simplicity,it can be assumed as just a graph. This has some similarities with Feynman Diagrams and Sum-over-histories in theoretical physics.

71. Each event node in the node which is a set as above, can have multiple outcomes and hence cause multiple effects. Outcome of an interaction amongst the elements of an event node is input to the adjacent event node which is also a set. Thus there could be multiple outgoing outcome edges each with a probability. Hence the EventNet is a random, directed, acyclic graph (acyclic assuming future cannot affect past preventing retrocausality). Thus EventNet is nothing but real history laid out as a graph. Hypothetically, if every event from beginning of the universe is causally connected as above, an indefinite ever growing EventNet is created.

72. If the EventNet is formulated as a Dynamic Graph with cycles instead of Static Graph as above where there causal edges can be deleted, updated or added, then the above EventNet allows changing the past theoretically though impossible in reality. For example, in an EventNet grown upto present, if a causal edge is added from a present node to past or some causal edge is deleted or updated in the past, then "present" causes "past" with or without cycles in the graph.

73. EventNet can be partitioned into "Past","Present" and "Future" probably by using some MaxFlow-MinCut Algorithms. The Cut of the graph as Flow Network is the "Present" and either side of the Cut is "Past" and "Future".

74. A recreational riddle on the EventNet: Future(Past) = Present. If the Future is modelled as a mathematical function, and if Past is fixed and Present is determined by 100% freewill and can have any value based on whimsical human actions, then is the function Future() well-defined?

75. Conjecture: Undirected version of EventNet is Strongly Connected or there is no event in EventNet without a cause (outgoing edge) or an effect(incoming edge).

76. Events with maximum indegree and outdegree are crucial events that have deep impact in history.

77. Events in each individual entity's existence are subgraphs of universal EventNet. These subgraphs overlap with each other.

78. There is an implicit time ordering in EventNet due to each causation edge. This is a "Logical Clock" similar to Lamport's Clock. In a cloud (e.g VIRGO Linux nodes, KingCobra MAC currency transactions) the EventNet is spread across the nodes and each node might have a subgraph of the giant EventNet.

79. EventNet can also be viewed as a Bayesian Network.

---
80. (THEORY) Mining EventNet for Patterns:

As an old saying goes "history repeats itself". Or does it really? Are there cycles of events? Such questions warrant checking the EventNet for cycles. But the above EventNet is acyclic by definition.This probably goes under the item 48 above that models the events as a hypergraph based on classification of events which is different from EventNet altogether. Alternatively, the EventNet nodes which are events with set of partakers and their interactions,can be mined for commonalities amongst them. Thus checking any pair of event nodes separated by a path of few edges (and thus separated in logical time) for common pattern suffices and this recursively continues.

Frequent Subgraph Mining of above EventNet (i.e whether a subgraph "repeats" with in the supergraph) as mentioned in Graph Search feature above could find patterns in events history.(Reference: Graph growing algorithms in chapter "Graph Mining", Data Mining, Han and Kamber)

There is a striking resemblance of EventNet to sequence mining algorithm CAMLS - a refined Apriori algorithm - (http://www.cs.bgu.ac.il/~yarongon/papers/camls.dasfaa2010.pdf) which is for mining a sequence of events - each event has intraevent partakers occurring at different points in time with gaps. EventNet mining is infact a generalization of linear time in CAMLS to distributed logical clock defined as EventNet causation infinite graph above. The topological orderings of EventNet can be mined using CAMLS. Sequence Mining python script that implements Apriori GSP has been added to repository and it can be applied to topologically sorted EventNet graphs as well.

81-86. (THEORY) Fractal nature of events:
---
81. Are events self-similar or exhibit fractal nature? This as in item 71, needs mining the event nodes which are sets of partakers for self-similarity. A fractal event is the one which has a cycle and any arbitrary point on the cycle has a cycle attached at that point and so on. An example of fractal event is the Hindu Vedic Calendar that has Major cycles, subcycles, cycles within cycles that occur recursively implying that calendar is self-similar. How to verify if the EventNet graph has a fractal nature or how to prove that a graph is "Fractal" in general? Would it be Ramsey theory again?

82. EventNet is a Partial Order where there may not be edges of causality between some elements. Thus a Topological Sort on this EventNet directed,acyclic graph gives many possible orderings of events. If history of the universe assuming absolute time is formulated as EventNet, then the topological sort does not give a unique ordering of events which is counterintuitive to absolute time. (Is this sufficient to say there is no absolute time?).

83. EventNet on history of events in the universe is quite similar to Directed Acyclic Graph based Scheduling of tasks with dependencies on parallel computers. Thus universe is akin to a Infinitely Massive Parallel Computer where events are dynamically scheduled with partakers, outcomes and with non-determinism. Even static scheduling is NP-Complete.

84. In the above EventNet, each node which is a set can indeed be a graph also with freewill interactions amongst the set members as edges. This gives a graph G1 with each node being replaced by a graph G2. This is a fractal graph constructed naturally and notions of Outer products or Tensor products or Kronecker products with each entry in adjacency matrix replaced by another adjacency matrix apply. By definition each node can be replaced by different graph.

85. Similar to Implication graphs as Random Growth Graphs, EventNet also qualifies as a Random Growth Network as events happen randomly and new edges are added whenever events happen (with previous Kronecker Tensor model). Rich-Get-Richer paradigm can also hold where nodes with more adjacent nodes are more likely to have future adjacent nodes. (A related notion of Freewill Interactions in
http://sites.google.com/site/kuja27/UndecidabilityOfFewNonTrivialQuestions.pdf?attredirects=0 is worth mentioning here).

86. Regularity Lemma can be used as a tool to test Implication and EventNet Random Growth Graphs - specifically seems to have an application in EventNet Kronecker Graph model above where each node is replaced by a graph and thus can be construed as epsilon-partition of the vertices that differ infinitesimally in density. Each element in the partition is the event.

****
87. COMMIT RELATED NOTES
****

(FEATURE- DONE) commits as on 3 September 2013

DecisionTree, NaiveBayes and SVM classifier code has been integrated into AstroInfer toplevel invocation with boolean flags

(FEATURE- DONE) commits as on 6 September 2013

In addition to USGS data, NOAA Hurricane dataset HURDAT2 has been downloaded and added to repository. Asfer Classifier Preprocessor script has been updated to classify set of articles listed in articlesdataset.txt using NaiveBayesian or DecisionTree Classifiers and training and test set text files are also

autogenerated. New parser for NOAA HURDAT2 datset has been added to repository. With this datasets HTML files listed in articlesdataset.txt are classified by either classifiers and segregated into "Event Classes".Each dataset document is of proprietary format and requires parser for its own to parse and autogenerate the date-time-longlat text file.Date-time-longlat data for same "Event Class" will be appended and collated into single Date-time-longlat text file for that "Event Class".

(FEATURE- DONE) commits as on 12 September 2013

1.asfer_dataset_segregator.py and asfer_dataset_segregator.sh - Python and wrapper shell script that partitions the parsed datasets which contain date-time-long-lat data based on classifier output grepped by the invoker shell script - asfer_dataset_segregator.sh - and writes the names of parsed dataset files which are classified into Event Class "<class>" into text files with names of regular expression "EventClassDataSet_<class>.txt"

2.DateTimeLongLat data for all datasets within an event class (text file) need to be collated into a single asfer.enchoros.<classname>.zodiacal or asfer.enchoros.<classname>.ascrelative. For this a Python script MaitreyaToEncHoroClassified.py has been added to repository that reads the segregated parsed dataset files generated by asfer_dataset_aggregator.sh and invokes maitreya_textclient for all date-time-long-lat data within all parsed datasets for a particular event class - "EventClassDataset_<class>.txt" and also creates autogenerated asfer.enchoros.<class>.zodiacal and asfer.enchoros.<class>.ascrelative encoded files

(FEATURE- DONE) commits as on 2 November 2013

Lot of commits for implementation of Discrete Hyperbolic Factorization with Stirling Formula Upperbound (http://sourceforge.net/p/asfer/code/HEAD/tree/cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound.cpp) have gone in. Overflow errors prevent testing large numbers.(URLs:
1) Multiple versions of Discrete Hyperbolic Factorization uploaded in http://sites.google.com/site/kuja27/
2) Handwritten by myself - http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicFactorization_UpperboundDerivedWithStirlingFormula_2013-09-10.pdf/download and
3) Latex version - http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_updated_rectangular_interpolation_search_and_StirlingFormula_Upperbound.pdf/download
)

(FEATURE- DONE) commits as on 20 November 2013 and 21 November 2013
---
1. Lots of testing done on Discrete Hyperbolic Factorization sequential implementation and logs have been added to repository.

2. An updated draft of PRAM NC version of Discrete Hyperbolic Factorization has been uploaded at: http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound.pdf/download that does PRAM merge of discrete tiles in logarithmic time before binary search on merged tile.

3. Preliminary Design notes for CRCW PRAM implementation of the above is added to repository at: http://sourceforge.net/p/asfer/code/237/tree/ImplementationDesignNotesForDiscreteHyperbolicFactorizationInPRAM.jpg. This adds a tile_id to each tile so that during binary search, the factors are correctly found since coordinate info gets shuffled after k-tile merge.

88.Tagging as on 16 December 2013

AsFer version 12.0 and VIRGO version 12.0 have been tagged on 6 December 2013 (updated the action items above).

(THEORY) 89. More reference URLs for Parallel RAM design of above NC algorithm for Discrete Hyperbolic Factorization for merging sorted lists:

1.http://arxiv.org/pdf/1202.6575.pdf?origin=publication_detail
2.https://electures.informatik.uni-freiburg.de/portal/download/3/6951/thm15%20-%20parallel%20merging.ppt (Ranks of elements)
3.http://cs.brown.edu/courses/csci2560/syllabus.html (Lecture Notes on CREW PRAM and Circuit Equivalence used in the NC algorithm for Discrete Hyperbolic Factorization above - http://cs.brown.edu/courses/csci2560/lectures/lect.24.pdf)
4.http://cs.brown.edu/courses/csci2560/lectures/lect.22.ParallelComputationIV.pdf
5.http://www.cs.toronto.edu/~bor/Papers/routing-merging-sorting.pdf
6.http://www.compgeom.com/~piyush/teach/AA09/slides/lecture16.ppt
7.Shift-and-subtract algorithm for approximate square root computation implemented in Linux kernel (http://lxr.free-electrons.com/source/lib/int_sqrt.c) which might be useful in finding the approximate square root in discretization of hyperbola (Sequential version of Discrete Hyperbolic Factorization)
8.http://research.sun.com/pls/apex/f?p=labs:bio:0:120 - Guy Steele - approximate square root algorithm
9.http://cstheory.stackexchange.com/questions/1558/parallel-algorithms-for-directed-st-connectivity - related theoretical discussion thread on PRAM algorithms for st-connectivity
10.PRAM and NC algorithms - http://www.cs.cmu.edu/afs/cs/academic/class/15750-s11/www/handouts/par-notes.pdf
11. An Efficient Parallel Algorithm for Merging in the Postal Model - http://etrij.etri.re.kr/etrij/journal/getPublishedPaperFile.do?fileId=SPF-1043116303185
12. Parallel Merge Sort - http://www.inf.fu-berlin.de/lehre/SS10/SP-Par/download/parmerge1.pdf
13. NC and PRAM equivalence - www.cs.tau.ac.il/~safra/ACT/CCandSpaceB.ppt
14. Extended version of BerkmanSchieberVishkin ANSV algorithm - http://www1.cs.columbia.edu/~dany/papers/highly.ps.Z (has some references to NC, its limitations, higly parallelizable - loglog algorithms) - input to this ANSV algorithm is elements in array (of size N) and not number of bits (logN) which is crucial to applying this to merge tree of factorization and proving in NC.
15. Structural PRAM algorithms (with references to NC) - http://www.umiacs.umd.edu/users/vishkin/PUBLICATIONS/icalp91.ps
16. Parallel Algorithms FAQ - NC, RAM and PRAM - Q24 - http://nptel.ac.in/courses/106102114/downloads/faq.pdf

17. Defintions related to PRAM model - http://pages.cs.wisc.edu/~tvrdik/2/html/Section2.html - Input to PRAM is N items stored in N memory locations - (For factorization, this directly fits in as the O(N) ~ Pi^2/6 * N coordinate product integers stored in as many locations - for discretized tiles of hyperbola)

18. Reduction from PRAM to NC circuits - http://web.cse.ohio-state.edu/~gurari/theory-bk/theory-bk-sevense6.html#Q1-80006-21 (The SIMULATE_RAM layer in each step is a subcircuit that is abstracted as a node in NC circuit. This subcircuit performs the internal computation of PRAM processor in that NC circuit node at some depth i. The NC circuit is simulated from PRAM model as - depth is equal to PRAM time and size is equal to number of processors.). Thus for Discrete Hyperbolic Factorization, the ANSV PRAM mergetree of polylogdepth is simulated as NC circuit with this reduction. Thus though each array element in ANSV is a logN bit integer, the SIMULATE_RAM

abstraction reduces it to a node in NC circuit that processes the input and propagates the output to next step towards root.

19. Length of input instance in PRAM (n or N) - http://web.cse.ohio-state.edu/~gurari/theory-bk/theory-bk-sevense2.html#Q1-80002-3 - "... When no confusion arises, because of the obvious relationship between the number N of values in the input and the length n of the input, N and n are used interchangeably. ...". For ANSV algorithm, the number N of values in the input array and the length n of the input are the same (N=n) and due to this it runs in polyloglogtime in input size O(loglogN) and with polynomial processors N/loglogN.

20. Quoted abstract - "... The all nearest smaller values problem is defined as follows. Let A = (a1 ; a2 ; : : : ; an) be n elements drawn from a totally ordered domain. For each ai , 1 <= i <= n, find the two nearest elements in A that are smaller than ai (if such exists): the left nearest smaller element aj (with j <= i) and the right nearest smaller element ak (with k >= i). We give an O(log log n) time optimal parallel algorithm for the problem on a CRCW PRAM ...". Thus N=n.

21. SIMULATE_RAM subcircuit in point 18 - Number of bits allowed per PRAM cell in PRAM to NC reduction - http://hall.org.ua/halls/wizzard/books2/limits-to-parallel-computation-p-completeness-theory.9780195085914.35897.pdf - pages 33-36 - quoted excerpts - "... Let M be a CREW-PRAM that computes in parallel time t(n) = (log n)^O(1) and processors p(n) = n^O(1) . Then there exists a constant c and a logarithmic space uniform Boolean circuit family, {αn }, such that αn on input x1 , . . . , xn computes output y1,1 , y1,2 , . . . , yi,j , . . . , where yi,j is the value of bit j of shared memory cell i at time t(n), for 1 ≤ i ≤ p(n) ∗ t(n) and 1 ≤ j ≤ c*t(n) ...". Thus number of bits allowed per PRAM memory location is upperbounded by polylogn. For ANSV algorithm underlying the Discrete Hyperbolic Factorization, maximum number of bits per PRAM cell is thus logN where N is the integer to factorize (and maximum number of PRAM cells is ~ pi^2/6 *N) thereby much below the above polylogN bound for number of bits per PRAM cell for creating a uniform NC circuit from PRAM model.

(THEORY) 90.Some notes on extensions to Integer partitions and Hash Functions (https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions.pdf?attredirects=0)

---

1. Riemann sums (discrete approximation of Riemann integral) of all the functions corresponding to the hash functions are same. Thus all such functions form an equivalence class.(Assuming each partition created by the hash functions as a function plot)

2. Hardy-Ramanujan asymptotic bound for partition function p(n) is ~ O(e^(pi*sqrt(0.66*n))/(4*1.732*n) which places a bound on number of hash functions also.(http://en.wikipedia.org/wiki/Partition_(number_theory))

3. If m-sized subsets of the above O(m!*e^(sqrt(n))/n) number of hash functions are considered as a (k,u)-universal or (k,u)-independent family of functions(Pr(f(x1)=y1...) < u/m^k,then follwing the notation in the link mentioned above, this m-sized subset family of hash functions follow the Pr(f(x1)=y1 & ...) < u/m^n where n is number of keys and m is the number of values. (m! is for summation over (m,lamda(i)) for all partitions)

4. Thus deriving a bound for number of possible hash functions in terms of number of keys and values could have bearing on almost all hashes including MD5 and SHA.

5. Birthday problem and Balls and Bins problem - Since randomly populating m

bins with n balls and probability of people in a congregation to have same birthday are a variant of Integer partitioning and thus hash table bucket chaining, bounds for birthday problem and Chernoff bounds derived for balls and bins could be used for Hash tables also (http://en.wikipedia.org/wiki/Birthday_problem, http://www.cs.ubc.ca/~nickhar/W12/Lecture3Notes.pdf)

6. Restricted partitions which is the special case of integer partitions has some problems which are NP-complete. Money changing problem which is finding number of ways of partitioning a given amount of money with fixed denominations(Frobenius number) is NP-complete(http://citeseer.uark.edu:8080/citeseerx/showciting;jsessionid=92CBF53F1D9823C47F64AAC119D30FC4?cid=3509754, Naoki Abe 1987). Number of partitions with distinct and non-repeating parts follow Roger-Ramanujan identities (2 kinds of generating functions).

7. The special of case of majority voting which involves integer partitions described in https://sites.google.com/site/kuja27/IndepthAnalysisOfVariantOfMajorityVotingwithZFAOC_2014.pdf requires a hash function that non-uniformly distributes the keys into hashes so that no two chains are equal in size (to simulate voting patterns without ties between candidates). This is the special case of restricted partitions with distinct and non-repeating parts of which money changing is the special case and finding a single solution is itself NP-complete.

8. Thus Majority voting can be shown to be NP-complete in 2 ways:
    8.1 by Democracy circuit (Majority with SAT) in http://sourceforge.net/projects/acadpdrafts/files/ImplicationGraphsPGoodEquationAndPNotEqualToNPQuestion_excerpts.pdf/download and https://sites.google.com/site/kuja27/PhilosophicalAnalysisOfDemocracyCircuitAndPRGChoice_2014-03-26.pdf
    8.2 by reduction from an NP-hard instance of Restricted Partition problem like Money changing problem for Majority voting with constituencies described in https://sites.google.com/site/kuja27/IndepthAnalysisOfVariantOfMajorityVotingwithZFAOC_2014.pdf

9. Infact the above two ways occur in two places in the process of democratic voting: The democracy circuit is needed when a single candidate is elected while the restricted partition in second point is needed in a multi-partisan voting where multiple candidates are voted for.

10. Point 8.2 above requires a restricted partition with distinct non-repeating parts. There are many results on this like Roger-Ramanujan identities, Glaisher theorem and its special case Euler's theorem which equate number of partitions with parts divisible by a constant and distinctiveness of the parts (odd, differing by some constant etc.,). Such a restricted partition is needed for a tiebreaker and hence correspond bijectively to hash collision chaining.

11. An interesting manifestation of point 10 is that nothing in real-life voting precludes a tie and enforces a restricted partition, with no two candidates getting equal votes, where all voters take decisions independent of one another(voter independence is questionable to some extent if swayed by phenomena like "votebank","herd mentality" etc.,) thereby theoretically invalidating the whole electoral process.

12. Counting Number of such restricted partitions is a #P-complete problem - https://www.math.ucdavis.edu/~deloera/TALKS/denumerant.pdf.

13. If a Hash table is recursive i.e the chains themselves are hashtables and so on... then this bijectively corresponds to a recurrence relation for partition function (expressing a partition of a higher integer in terms of lower integer).

14. If the hash table chains are alternatively viewed as Compositions of an integer (ordered partitions) then there are 2^(n-1) maximum possible

compositions.(http://en.wikipedia.org/wiki/Composition_(number_theory))

15. In the summation over all parts of partitions derived in https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions.pdf if m==n then it is the composition in point 14 above and thus summation over all parts of partitions is greater than or equal to 2^(n-1) since some permutations might get repeated across partitions. Thus the summation expresses generalized restricted composition(summation_over_all_partitions_of_n((n,lamda(i))>=2^(n-1)).

16. Logarithm of above summation then is equal to (n-1) and thus can be equated to any partition of n. Thus any partition can be written as a series which is the combinatorial function of parts in all individual partitions.

---
91. Updated drafts on Integer partitions and hash function (with points in 80 above) , Circuits for Error probability in Majority Voting
---
1. https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions_2014.pdf?attredirects=0&d=1
2. https://sites.google.com/site/kuja27/CircuitForComputingErrorProbabilityOfMajorityVoting_2014.pdf?attredirects=0&d=1

---
92. Reference URLs for restricted integer partitions with distinct parts
---
(for http://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions_2014.pdf?attredirects=0&d=1)
1. Generating function - http://math.berkeley.edu/~mhaiman/math172-spring10/partitions.pdf
2. Schur's theorem for asymptotic bound for number of denumerants - http://en.wikipedia.org/wiki/Schur's_theorem
3. Frobenius problem - http://www.math.univ-montp2.fr/~ramirez/Tenerif3.pdf
4. Locality Sensitive Hashing (that groups similar keys into a collision bucket chain using standard metrics like Hamming Distance) - http://hal.inria.fr/inria-00567191/en/, http://web.mit.edu/andoni/www/LSH/index.html, http://en.wikipedia.org/wiki/Locality-sensitive_hashing
5. Locality Sensitive Hashing and Lattice sets (of the diophantine form a1*x1+a2*x2+...+an*xn) - http://hal.inria.fr/docs/00/56/71/91/PDF/paper.pdf
6. Minhash and Jaccard similarity coefficient - http://en.wikipedia.org/wiki/MinHash#Jaccard_similarity_and_minimum_hash_values (similar to LSH that emphasizes on hash collisions for similarity measures between sets e.g bag of words of URLs)

(THEORY) 93. Reduction from Money Changing Problem or 0-1 Integer LP to Restricted Partitions with distinct parts
---
If the denominations are fixed as 1,2,3,4,5,....,n then the denumerants to be found are from the diaphantine equation:
a1*1 + a2*2 + a3*3 + a4*4 + a5*5 + ...+ an*n
(with ai = 0 or 1). GCD of all ai(s) is 1. Thus Schur's theorem for MCP or Coin Problem applies.
Integet 0-1 LP NP-complete problem can also be reduced to above diophantine format instead of MCP. Finding one such denumerant with

boolean values is then NP-complete and hence finding one partition with distinct non-repeating parts is NP-complete (needed in multipartisan majority vote).

(FEATURE- DONE) 94. Commits as on 23 April 2014

Updated pgood.cpp with some optimizations for factorial computations and batched summation to circumvent overflow to some extent.
For Big decimals IEEE 754 with 112+ bits precision is needed for which boost::multiprecision or java.math.BigDecimal might have to be used.

(FEATURE- DONE) 95. Commits as on 7 July 2014

Initial implementation of a Chaos attractor sequence implementation committed to repository.

(FEATURE- DONE) 96. Commits as on 8 July 2014

Python-R (rpy2) code for correlation coefficient computation added to above Chaos attractor implementation.

(FEATURE- DONE) 97. Time Series Analysis - Commits as on 9 July 2014

DJIA dataset, parser for it and updated ChaosAttractor.py for chaotic and linear correlation coefficient computation of DJIA dataset
have been committed.

(FEATURE- DONE) 98. Commits as on 10 July 2014

Python(rpy2) script for computing Discrete Fourier Transform for DJIA dataset has been added (internally uses R). [python-src/DFT.py]

(FEATURE- DONE) 99. Commits as on 11 July 2014

Python(rpy2) script for spline interpolation of DJIA dataset and plotting a graph of that using R graphics has been added. [python-src/Spline.py]

(FEATURE- DONE) 100. Commits as on 15 July 2014 and 16 July 2014

Doxygen documentation for AsFer, USBmd, VIRGO, KingCobra and Acadpdrafts have been committed to GitHub at
https://github.com/shrinivaasanka/Krishna_iResearch_DoxygenDocs. LOESS local polynomial regression fitting code using loess() function in R has been added at python-src/LOESS.py. Approximate linear interpolation code using approx() and approxfun() in R has been added at python-src/Approx.py

---
(FEATURE- DONE) 101.Commits as on 23 July 2014

Draft additions to http://arxiv.org/pdf/1106.4102v1.pdf are in the Complement Function item above. An R graphics rpy2 script RiemannZetaFunctionZeros.py has been added to parse the first 100000 zeros of RZF and plot them using R Graphics.

(FEATURE - THEORY - Visualizer Implementation DONE) 102. Dense Subgraphs of the WordNet subgraphs from Recursive Gloss Overlap

---
http://arxiv.org/abs/1006.4458 and
http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf have been extended for a summary creation algorithm from WordNet subgraphs of Recursive Gloss Overlap by filtering out low degree vertices(https://sites.google.com/site/kuja27/DocumentSummarization_using_SpectralGraphTheory_RGOGraph_2014.pdf?attredirects=0&d=1). For undirected graphs there is a notion of k-core of a graph which is an induced subgraph of degree k vertices. Since Wordnet is a directed graph, recently there are measures to measure degeneracy(D-cores - http://www.graphdegeneracy.org/dcores_ICDM_2011.pdf). Graph peeling is a process of removing edges to create a dense subgraph. This can be better visualized using visual wordnet implementations:

102.1 http://kylescholz.com/projects/wordnet/
102.2 http://www.visuwords.com
102.3 http://www.visualthesaurus.com/browse/en/Princeton%20WordNet

---
103. AsFer version 14.9.9 release tagged on 9 September 2014

(FEATURE- DONE) 104. Commits as on 9 October 2014

Initial python script implementation for Text compression committed with logs and screenshot. Decompression is still hazy and needs some more algorithmic work.

(FEATURE- DONE) 105. Commits as on 21 October 2014

An experimental POC python implementation that uses Hidden Markov Model for decompression has been committed. But it depends on one-time computing a huge number of priors and their accuracy.

(THEORY) 106. Creating a summary graph from EventNet

EventNet graph for historical cause and effect described above can be huge of the order of trillion vertices and edges. To get a summary of the events might be necessary at times - similar to a document summarization. Summarizing a graph of event cause-effects requires giving importance to most important events in the graph - vertices with high number of degrees (indegrees + outdegrees) . This is nothing but finding k-core of a graph which is a maximal connected subgraph of EventNet where every vertex is of degree atleast k.

(FEATURE- DONE) 107. Commits as on 1 November 2014

A minimal PAC learning implementation in python to learn a Boolean Conjunction from dataset has been added to repository.

(FEATURE- DONE) 108. Commits as on 4 November 2014

Initial implementation for Minimum Description Length has been added to repository.

(FEATURE- DONE) 109. Commits as on 6 November 2014

Python Shannon Entropy implementation for texts has been added to repository and is invoked in MinimumDescLength.py MDL computation.

(FEATURE- DONE) 110. Commits as on 7 November 2014

Python Kraft Inequality MDL implementation has been committed.

(FEATURE- DONE) 111. Commits as on 11 November 2014

C++ implementation for Wagner-Fischer Dynamic Programming algorithm that iteratively computes Levenshtein Edit Distance than recursively has been added to repository.

(FEATURE - DONE) 112. Expirable objects

Setting expiry to an object is sometimes essential to control updates and access to an object. Example application of this - A JPEG image
should be accessible or displayable for only a finite number of times and after that it has to self-destruct. Though doing this in C++ is
straightforward with operator overloading, in C it looks non-trivial. Presently only C++ implementation is added to repository.

(FEATURE- DONE) 113. Commits as on 11 November 2014 - Expirable template class implementation
---
Similar to weak_ptr and shared_ptr, a generic datatype that wraps any datatype and sets an expiry count to it has been added to repository
in cpp-src/expirable/ . For example, a bitmap or a JPEG image can be wrapped in expirable container and it can be access-controlled.
If expiry count is 1, the object (an image for example) can be written to or displayed only once and thus a singleton. Presently this is
only for rvalues. Implementing for lvalues (just a usage without assignment should be able to expire an object) seems to be a hurdle as
there is no operator overloading available for lvalues - i.e there is no "access" operator to overload. This might be needed for KingCobra
MAC electronic money also.(std::move() copy-by-move instead of a std::swap() copy-by-swap idiom)

(FEATURE- DONE) 114. Commits as on 12 November 2014

Expirable template in asferexpirable.h has been updated with 2 operator=() functions for copy-assignment and move-assignment which interally invoke std::swap() and std::move(). Explicit delete(s) are thus removed. Also example testcase has been updated to use a non-primitive datatype.

(FEATURE- DONE) 115. Commits as on 13 November 2014

Overloaded operator*() function added for expiry of "access" to objects. With this problem mentioned in 113 in tracking access or just usage is circumvented indirectly.

(FEATURE- DONE) 116. Commits as on 15 November 2014

Python implementation for Perceptron and Gradient has been added to repository.

(FEATURE- DONE) 117. Commits as on 17 November 2014

Python implementation for Linear and Logistic Regression in 2 variables.

(FEATURE- DONE) 118. Commits as on 20 November 2014

C++ implementation of K-Means clustering with edit distance as metric has been added to
repository.

(FEATURE- DONE) 119. Commits as on 21 November 2014

C++ implementation of k-Nearest Neighbour clustering has been added to
repository.

---
120. Commits as on 24 November 2014

Bugfixes to kNN implementation.

(FEATURE- DONE) 121. Commits as on 25 November 2014

Python script for decoding encoded horoscope strings (from Maitreya's Dreams) has been added to
repository.

---
122. (FEATURE - DONE) Commits as on 3 December 2014

Initial implementation for EventNet:
---
1. EventNet python script has inputs from two text files - EventNetEdges.txt and EventNetVertices.txt - which define the event vertices and the causations amongst them with partakers in each event node
2. This uses GraphViz (that in turn writes a dot file) and python-graph+gv packages
3. GraphViz writes to EventNet.graphviz.pdf and EventNet.gv.png rendered and visualized graph files.
4. Above text files are disk-stored and can be grown infinitely.
5. EventNetVertices.txt has the format:
        <event vertex> - <csv of partakers in the event vertex>
   EventNetEdges.txt has the format:
        <ordered pairs of vertices for each causation edge>
6. Topological Sorting of EventNet using python-graph algorithms package

---
(THEORY) 123. EventNet - partakers of events and an application of EventNet to a related problem
---
Event vertices have partakers of event as defined in 122. Point 84-86 previously defined EventNet as
a fractal graph tensor where each vertex is a graph or partakers. Alternative formulation of this is
where the partakers are just key words or persons in that event. For example, a fictitious story can be translated into a giant EventNet where each vertex is an event with corresponding partakers in it. It could be difficult to find edges among the partakers (Ideally the edges are conversations among the partakers of an event specific to this example). As an approximation, the partakers could be simply the keywords parsed from that set of conversations. Complexity or connectedness and number of topological orderings possible for this translated EventNet is a measure of elegance.

(FEATURE- DONE) 124. Commits as on 4 December 2014

EventNet - a cloudwide event ordering with unique id implementation
---
EventNet has 2 input files for vertices and edges with partakers and
writes an output file with ordering of the events

Input - EventNetVertices.txt has the format:
	<event vertex> - <csv of partakers> - <tuples of conversations amongst
the partakers # separated>
	partakers could be machine id(s) or IP addresses and thread id(s) and the
conversations being the	messages to-and-fro across the partakers, which
create an IntraEvent Graph of Conversations
	within each event vertex

Input - EventNetEdges.txt has the format:
	<event vertex1, event vertex2>

Output - EventNetOrdering.txt has the format:
	<index in chronologically ascending order> - <event id>

EventNet script thus is run in a central node which has the input files above
that is
updated by all the nodes in cloud. Outgoing edge from an event vertex has
partakers from multiple
events and thus is an outcome of the event. If the input files are split and
stored in multiple cloud
nodes, the topological sorts for multiple input files have to be merged to
create a single cloudwide
ordering.

(THEORY) 125. Massive EventNet computation

Each conversation of the events needs to create a log message that is sent to
the EventNet service
which updates the input vertices and edges files. The python EventNet script run
optionally on a hadoop
cluster mapreduce recomputes the topological ordering periodically. This is
quite tedious a process that can flood the cloud with log messages enabled by
config boolean flag or compile time #ifdef.

(FEATURE- DONE) 126. Commits as on 5 December 2014

Initial C++ Boost::graph based implementation for EventNet has been added to
repository.

(THEORY) 127. 2-dimensional random walks for decision making (experimental)

Continued from point 49 above, the psychological process of decision making (in
a mental conflict with
two opposing directions) is akin to a game theoretical notion of Nash
Equilibrium - where a huge payoff
matrix is constructed for the random walk directions as strategies for the two
conflicting decisions. There could be Nash Equilibria where decision1 doesn't

gain from changing the random walk direction and decision2 doesn't gain from changing the direction (max(column),max(row)). These equilibria are like a win-win or a dilemma . For that matter this should apply to decision trees as well. Related application of equilibria for epidemic and malware are at: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8478

---

128. Commits as on 9 December 2014

Bugfixes for DOT file generation and toposort in C++ EventNet implementation.

---

129. (THEORY) Draft Updates to https://sites.google.com/site/kuja27/CircuitForComputingErrorProbabilityOfMajorityVoting_2014.pdf?attredirects=0&d=1 - P(Good) series computation

---

For even number of finite population (electorate) the binomial coefficient summation in uniform distribution has an extra coefficient which when summed up tends to zero at infinity. Thus P(Good) series converges to,

(2^n - nC(n/2))/2^(n+1) = 0.5 - epsilon where epsilon=sqrt(1/(2*n*pi)) limit for which vanishes at infinity

(or) probability of good choice is epsilon less than 50% for finite even number of electorate for uniform distribution - a weird counterintuitive special case. Thus the counterexample for P Vs NP is still valid at infinity if infinite majority is decidable(related to decidability of infinite-candidate condorcet election , May's theorem and Arrow's theorem) . Written notes for this are at: http://sourceforge.net/p/asfer/code/568/tree/python-src/ComplFunction_DHF_PVsNP_Misc_Notes.pdf. What this implies is the LHS is PRG circuit in NC or P while the RHS is a humongous infinite majority+SAT (oracle) circuit - unbounded fan-in constant depth circuit. This then becomes an NP-complete Circuit SAT problem - http://www.cs.berkeley.edu/~luca/cs170/notes/lecture22.pdf.

If this infinite-ness breaches the polynomiality then what is quite puzzling is that RHS becomes a Direct Connect DC circuit equivalent to PH(Polynomial Hierarchy). Quoted excerpts from Arora-Barak for DC uniform circuits:

"...
6.6
Circuits of exponential size
As noted, every language has circuits of size O(n^2^n). However, actually finding these circuits may
be difficult— sometimes even undecidable. If we place a uniformity condition on the circuits, that
is, require them to be efficiently computable then the circuit complexity of some languages could
exceed n^2^n . In fact it is possible to give alternative definitions of some familiar complexity classes,
analogous to the definition of P in Theorem 6.7.
Definition 6.28 (DC-Uniform)
Let {C n } n≥1 be a circuit family. We say that it is a Direct Connect uniform (DC uniform) family if,
given hn, ii, we can compute in polynomial time the i th bit of (the representation of) the circuit C n .
More concretely, we use the adjacency matrix representation and hence a family {C n } n∈N is DC
uniform iff the functions SIZE, TYPE and EDGE defined in Remark ?? are computable in polynomial
time.

Note that the circuits may have exponential size, but they have a succinct representation in
terms of a TM which can systematically generate any required node of the circuit in polynomial
time.
Now we give a (yet another) characterization of the class PH, this time as languages computable
by uniform circuit families of bounded depth. We leave it as Exercise 13.

Theorem 6.29
L ∈ P H iff L can be computed by a DC uniform circuit family {C n } that
• uses AND, OR, NOT gates.
O(1)
• has size 2^n
and constant depth (i.e., depth O(1)).
• gates can have unbounded (exponential) fanin.
• the NOT gates appear only at the input level.
If we drop the restriction that the circuits have constant depth, then we obtain exactly EXP
..."

The RHS Majority+SAT circuit has all characteristics satisfying the DC-uniformity in the theorem above - size can be exponential, unbounded fanin, depth is constant (each Voter Circuit SAT CNF is AND of ORs - depth 3) and NOT gates are required only in leaf nodes of the Voter Circuit SAT. Thus the counterexample could imply very,very importantly that P could be equal to PH or EXP (P=PH or P=EXP based on depth restricted or unrestricted) in infinite or exponential case respectively - could have tumultuous ramifications for complexity theory as a whole as it goes beyond P=NP question - for perfect voter scenario described in point 133 - all circumstantial evidences above point to this.

It is not necessary that per voter SAT is same for all voters. Each voter can have unrestricted depth SAT clauses (in real world ,each voter decides on his-her own volition and depth of their SAT circuits can vary based on complexity of per-individual decision making algorithm) - due to which RHS zero-error DC circuit need not be in PH but in EXP.

even if a BQP algorithm is used in voting outside the purview of PH but in EXP, it doesn't change the above unless:
 - perfection is impossible i.e there cannot be zero-error processes in the universe
 - DC-circuit is not drawable (or undecidable if it can be constructed)
 - infinite majority is undecidable (so circuit is DC non-uniform and not DC-uniform)
 - the voter CNF can only be 2-SAT (which is highly unlikely) and not k-SAT or 3-SAT

129.1 Toda's theorem and P(good) circuit above:

PH is contained in P^#P (or) P with #no-of-solutions-to-SAT oracle (Toda's theorem).
If zero-error Majority+SAT voting DC uniform circuit is in PH then due to LHS=RHS of the P(good) series convergence (in cases of p=0,p=1 and p=0.5 as derived in http://sourceforge.net/p/asfer/code/916/tree/cpp-src/miscellaneous/MajorityVotingErrorProbabilityConvergence.JPG),
PH collapses(?) to P (quite unbelievably):
LHS Pseudorandom choice is in P while RHS Majority+SAT is in PH=DC circuit (restricted depth) or EXP (unrestricted depth)
(i.e there is a P algorithm for PH).
if P=PH:
  P=PH is in P^#P by Toda's theorem

if P=EXP:
 P=PH=EXP in P^#P or P=PH=EXP=P^#P (which is a complete collapse)

[p=0.5 is the uniform distribution which is a zero bias space while for other probabilities some bit patterns are less likely - epsilon-bias spaces]

---
130. (THEORY) Counterexample definition for P Vs NP

Majority Circuit with SAT voter inputs with finite odd number of voters, in uniform distribution
converges to 1/2 same as LHS for pseudorandom choice. Even number of voters is a special case described
previously. Also both LHS and RHS converge to 1 if probability is 1 (without any errors in pseudorandom choice and majority voting) which is counterintuitive in the sense that LHS nonprobabilistically achieves in PTIME what RHS does in NPTIME.

---
131. (THEORY) Infinite majority and SAT+Majority Circuit

Infinite version of majority circuit is also a kind of heavy hitters problem for streaming algorithms where majority has to be found in an infinite bitstream of output from majority circuits for
voters(http://en.wikipedia.org/wiki/Streaming_algorithm#Heavy_hitters). But nonuniform distribution still requires hypergeometric series. Moreover the SAT circuit is NP-complete as the inputs are unknown and is P-Complete only for non-variable gates(Circuit Value Problem). Odd number of electorate does not give rise to above extra coefficient in summation and is directly deducible to 0.5.

---
132. (THEORY) May's Theorem of social choice

May's Theorem: In a two-candidate election with an odd number of voters, majority rule is the only voting system that is anonymous, neutral, and monotone, and that avoids the possibilites of ties.

May's theorem is 2-candidate analog of Arrow's Theorem for 3-candidate condorcet voting. May's theorem for 2 candidate simple majority voting defines a Group Decision Function which is valued at -1, 0 or 1
(http://www.jmc-berlin.org/new/theorems/MaysTheorem.pdf) for negative, abstention and positive vote for a candidate. For 2 candidates, positive vote for one candidate is negative for the other (entanglement).In the democracy circuit (SAT+Majority) the SAT clauses are kind of Group Decision Functions but with only binary values without any abstention. This is kind of alternative formulation - corollary - for May's theorem. Arrow's theorem does not apply for 2 candidate simple majority voting. May's theorem stipulates the conditions of anonymity,neutrality,decisiveness,monotonicity which should apply to the Majority+SAT circuit decision function as well. Neutrality guarantees that all outcomes are equally probable without bias which might imply that only uniform distribution is allowed in 2 candidate Majority+SAT voting. Thus there is a reduction from May's theorem to SAT+Majority democracy circuit. May's theorem does not apply to ties.This has been generalized to infinite electorate by Mark Fey. Anonymity is secret balloting. Monotonicity is non-decremental (e.g by losing votes a losing candidate is not winner and vice versa).

Additional References:

132.1 May's theorem -
http://www.math.cornell.edu/~mec/Summer2008/anema/maystheorem.html

---
133. (THEORY) Pseudorandom number generator and Majority voting for choice on a set
---
Let S be a set of elements with "goodness" attribute for each element. Choice using LHS and RHS on this set is by a PRG and a majority voting circuit with SAT inputs. LHS for pseudorandom choice consists of two steps:
133.1 Accuracy of the PRG - how random or k-wise independent the PRG is - this is usually by construction of a PRG (Blum-Micali, Nisan etc.,)
133.2 Goodness of chosen element - PRG is used to choose an element from the set - this mimicks the realworld phenomenon of non-democractic social or non-social choices. Nature itself is the PRG or RG in this case. After choice is made, how "good" is the chosen is independent of (133.1). Proof is by contradiction - if it were dependent on PRG used then a distinguisher can discern the PRGs from True Randomness by significant fraction.
133.3 Thus if the set S is 100% perfect - all elements in it are the best - LHS of P(Good) is 1. Similarly the RHS is the majority voting within these "best" elements which can never err (i.e their SAT clauses are all perfect) that converges to 1 by binomial coefficient summation in uniform distribution.

From the aforementioned,it is evident that for a 100% perfect set,both PRG(LHS) and Majority+SAT(RHS) Voting are two choice algorithms of equal outcome but with differing lowerbounds(with p=0.5 both LHS and RHS are in BPP or BPNC,but when p=1 then RHS is NP and LHS is P or NC). The set in toto is a black-box which is not known to the PRG or individual voters who are the constituents of it.

(FEATURE- DONE) 134.Commits as on 8 January 2015

134.1 C++ implementation for Longest Common Substring has been added to repository with logs. Datasets were the clustered encoded strings output by KMeans clustering C++ implementation.
134.2 AsFer+USBmd+VIRGO+KingCobra+Acadpdrafts - version 15.1.8 release tagged.

---
135. (THEORY) Updates, Corrigenda and References to Space Filling Algorithm in : https://sites.google.com/site/kuja27/Analysis%20of%20a%20Randomized%20Space%20Filling%20Algorithm%20and%20its%20Linear%20Program%20Formulation.pdf?attredirects=0

135.1 The standard LP form can be obtained by slack variables (with only equalities) - mentioned as free variables in above - http://ocw.mit.edu/courses/sloan-school-of-management/15-053-optimization-methods-in-management-science-spring-2013/tutorials/MIT15_053S13_tut06.pdf

135.2 Pseudorandom generator with linear stretch in NC1 - http://homepages.inf.ed.ac.uk/mcryan/mfcs01.ps
- ... " The notion of deterministically expanding a short seed into a long string that ... the question of whether strong pseudorandom generators actually exist is a huge ... hardness assumption, that there is a secure pseudorandom generator in NC1 " ... [Kharitonov]

135.3 This space filling algorithm is in NC (using Tygar-Rief) with an assumption that multiplicative inverse problem is almost always not in RNC.

135.4 Tygar-Rief algorithm outputs n^c bits in parallel (through its PRAM processors) for n=logN for some composite N and is internal to the algorithm. For the sake of disambiguation, the number of bit positions in grid corresponding to the LP is set to Z.

135.5 Above pseudorandom bits have to be translated into the coordinate

positions in the grid which is Z. In other words, pseudorandom bits are split into Z substrings (n^c/logZ = Z) because each coordinate in the grid of size 2^logZ/2 * 2^logZ/2 is notated by the tuple of size(logZ/2, logZ/2)

135.6 Constant c can be suitably chosen as previously to be c = log(Z*logZ)/logn (n need not be equal to Z)

135.7 Either the n^c sized pseudorandom string can be split sequentially into (logZ/2+logZ/2) sized substring coordinate tuples (or) an additional layer of pseudorandomness can be added to output the coordinate tuples by psuedorandomly choosing the start-end of substrings from n^c size string, latter being a 2-phase pseudo-pseudo-random generator

135.8 In both splits above, same coordinate tuple might get repeated - same bit position can be set to 1 more than once. Circumventing this requires a Pseudorandom Permutation which is a bijection where Z substrings do not repeat and map one-one and onto Z coordinates on grid. With this assumption of a PRP all Z coordinates are set to 1 (or) the P-Complete LP is maximized in this special case, in NC.

135.9 There are Z substrings created out of the 2^n^c pseudorandom strings generated by Tygar-Rief Parallel PRG. Hence non-repeating meaningful substrings can be obtained only from Z!/2^n^c fraction of all pseudorandom substrings.

135.10 Maximizing the LP can be reduced to minimizing the collisions (or) repetitive coordinate substrings above thus filling the grid to maximum possible. An NC algorithm to get non-repetitive sequence of coordinates is to add the index of the substring to the each split substring to get the hashed coordinates in the grid. For example in the 2*2 grid, if the parallel PRG outputs 01011100 as the pseudorandom bit string, the substrings are 01, 01, 11 and 00 which has a collision. This is resolved by adding the index binary (0,1,2,3) to corresponding substring left-right or right-left. Thus 01+00, 01+01, 11+10, 00+11 - (001,010,101,011) - are the non-repetitive hashed coordinates. The carryover can be avoided by choosing the PRG output string size. This is a trivial bijection permutation.

135.11 The grid filling can be formulated as a Cellular Automaton of a different kind - by setting or incrementing the 8 neighbour bit positions of a coordinate to 1 while setting or incrementing a coordinate to 1.

135.12 Instead of requiring the non-repetitive coordinate substrings mentioned in (10) supra, the grid filling can be a multi-step algorithm as below which uses the Cellular Automaton mentioned in (11).

135.13 Cellular Automaton Algorithm:

(Reference for reductions below:
http://cstheory.com/stockmeyer@sbcglobal.net/csv.pdf [Chandra-Stockmeyer-Vishkin])
[http://sourceforge.net/p/asfer/code/HEAD/tree/asfer-docs/AstroInferDesign.txt and
https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt]

The circuits described in [ChandraStockmeyerVishkin] are constant depth circuits [AC0] and thus in [NC1] (unbounded to bounded fanin logdepth reduction)

From wikipedia - Linear programs are problems that can be expressed in canonical form:
c^T * X
subject to AX <= b
and X >= 0

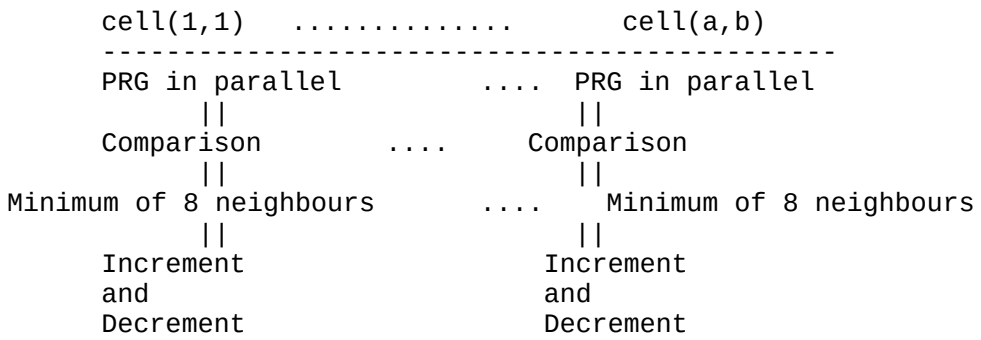
maximize X
subject to AX <= some maximum limit
and X >= 0

Below Cellular Automaton Monte Carlo NC algorithm for grid filling assumes that:
C=[1,1,1,...1]
A=[1,1,1....1] in the above and the vector of variables X is mapped to the grid - sequentially labelled in ascending from top-left to bottom-right, row-by-row and column-by-column. Though this limits the number of variables to be a square, additional variables can be set to a constant.

Grid cells (or variables in above grid) are initialized to 0.

loop_forever
{
(135.13.1) Parallel Pseudorandom Generator (Tygar-Rief) outputs bit string which are split into coordinates of grid and corresponding Grid cells are incremented instead of overwriting. This simulates parallel computation in many natural processes viz., rain. This is in NC. This is related to Randomness Extractors - https://en.wikipedia.org/wiki/Randomness_extractor - if rainfall from cloud can be used a natural weak entropy parallel randomness source (or strong?) - predicting which steam particle of a cloud would become a droplet is heavily influenced by the huge number of natural variables - fluid mechanics comes into force and this happens in parallel. Extractor is a stronger notion than Pseudorandom Generator. Extracting parallel random bits can also be done from a fluid mechanical natural processes like turbulent flows.
(135.13.2) Each grid cell has a constant depth Comparison Circuit that outputs 1 if the grid cell value exceeds 1. This is in AC0(constant-depth, polynomial size).
(135.13.3) If above Comparison gate outputs 1, then another NC circuit can be constructed that finds the minimum of grid neigbours of a cell (maximum 8 neighbours for each cell in 2-dimensional grid), decrements the central cell value and increments the minimum neighbour cell. This can be done by a sorting circuit defined in [ChandraStockMeyerVishkin]. This simulates a percolation of a viscous fluid that flows from elevation to lowlying and makes even. Each grid cell requires the above comparator and decrement NC circuit.
}

Above can be diagramatically represented as: (for N=a*b cells in grid in parallel)



135.14 For example, a 3*3 grid is set by pseudorandom coordinate substrings obtained from Parallel PRG (6,7,7,8,9,1,3,3,2) as follows with collisions (grid coordinates numbered from 1-9 from left-to-right and top-to-bottom):

1 1 2
0 0 1
2 1 1

Above can be likened to a scenario where a viscous fluid is unevenly spread at

random from heavens.
By applying cellular automaton NC circuit algorithm above, grid becomes:
(minimizes collisions,maximizes variables and LP)

```
1  1  1
1  1  1
1  1  1
```

Above can be likened to a scenario where each 8-neighbour grid cell underneath computes the evened-out cellular automaton in parallel. Size of the NC circuit is polynomial in LP variables n and is constant depth as the neighbours are constant (ChandraStockmeyerVishkin)
Thus there are 2 NC + 1 AC circuits which together maximize the special case of P-Complete LP instance without simplex where each grid coordinate is an LP variable. 135.13.1 (randomly strewn fluid) is independent of 135.13.2 and 135.13.3 (percolation of the fluid). This is both a Monte-Carlo and Las Vegas algorithm. The Parallel coordinate generation with Tyger-Reif Parallel PRG is Monte Carlo Simulation where as there is a guaranteed outcome with 100% possibility due to Comparison and Increment-Decrement circuits.

135.15 References on Cellular Automata, Parallel Computation of CA, Fluid Dynamics, Navier-Stokes equation, Percolation Theory etc.,:

135.15.1 http://www.nt.ntnu.no/users/skoge/prost/proceedings/acc11/data/papers/1503.pdf
135.15.2 http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.8377
135.15.3 http://www.stephenwolfram.com/publications/academic/cellular-automaton-fluids-theory.pdf
135.15.4 Constant Depth Circuits - http://www.cs.berkeley.edu/~vazirani/s99cs294/notes/lec5.ps
135.15.5 Complexity Theory Companion - https://books.google.co.in/books?id=ysu-bt4UPPIC&pg=PA281&lpg=PA281&dq=Chandra+Stockmeyer+Vishkin+AC0&source=bl&ots=fzmsGWrZXi&sig=jms3hmJoeiHsf5dq-vnMHUOU54c&hl=ta&sa=X&ei=jjcqVYrNFc25uATkvICoDA&ved=0CCAQ6AEwAQ#v=onepage&q=Chandra%20Stockmeyer%20Vishkin%20AC0&f=false
135.15.6 Pseudorandom Permutation in Parallel - https://www.nada.kth.se/~johanh/onewaync0.ps

---

135.16 A related algorithm - filling a square with infinitely many inscribed circles

---

Let S be a square of unit area. This square is filled with infinitely many circles of varying radii. This
can be expressed in terms of geometric equations for circle - x(i)^2 +y(i)^2 = r(i)^2. Each (x(i),y(i)) is a coordinate on the circle of radius r(i). Since x(i) and y(i) are < 1, let x(i) = 1/a(i) and y(i) = 1/b(i) for some integers a(i) and b(i). At infinity the square is filled with above infinite number of circles that cover the unit area (has some analogies to set cover and VC-dimension-shattering). This can be written as the infinite summation pi*(1/a(1)^2 + 1/b(1)^2 + 1/a(2)^2 + 1/b(2)^2 + ...) = 1 (or sum of areas of all circles equals the unit area of square). Thus this equation is a quadratic program instead of a linear program. This is extensible to higher dimensions also (filling a hypercube with infinite number of n-spheres). For example, in 3 dimensions, set of spherical bubbles of varying radii covers the entire cube.

135.17 Above grid can be alternatively formulated as a Voronoi Diagram where Parallel PRG randomly sets multiple points on the plane and a tesselation of the plane covers these points. Instead of squares a Delaunay Triangulation is

obtained from the dual of the Voronoi diagram.

Reference:

135.18. Circle Packing and Grid Filling - http://11011110.livejournal.com/332331.html - Filling a square with non-overlapping circles of maximum radii which is formulated as Dual of LP relaxation problem for matching where the constraints are that circles do not overlap i.e sum of radii <= distance between centroids and objective function maximizes the radii variables.
135.19. 135.16 is a slight variant of Kepler's Theorem [Proof - [Hales] - https://sites.google.com/site/thalespitt/] - filling a cube with small spheres has maximum density ~74%
135.20 Maximizing Sum of Radii of Balls - [Eppstein] - http://www.ics.uci.edu/~eppstein/pubs/Epp-CCCG-16-slides.pdf
135.21 Thue's Theorem For packing of discs on 2D - http://www.math.stonybrook.edu/~tony/whatsnew/dec00/paper.html - Circle packing in 2-dimension has maximum density (Pi/[2*sqrt(3)] ~ 90.69%) when circles are arranged in hexagonal lattice. The LP formulation in https://5d99cf42-a-62cb3a1a-s-sites.googlegroups.com/site/kuja27/Analysis%20of%20a%20Randomized%20Space%20Filling%20Algorithm%20and%20its%20Linear%20Program%20Formulation.pdf and Cellular Automaton Algorithm Version above for Optimum Space Filling of a 2D plane by Monte-Carlo Parallel Random process are RNC algorithmic solutions to Thue'e theorem (e.g natural process of Rain tiles earth in parallel by droplets of small radii and ultimately whole plane is covered, difference being allowance of circle overlaps) maximizing the variable points on plane having value 1.
135.22 Circle Packing Theorem and Planarity - [Koebe-Andreev-Thurston] - Graph is planar if and only if it is K5,K3,3 minor-free and is an intersection graph which has edges drawn between centroids of every pair of tangential closely packed coins of varied radii as vertices (https://gitlab.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt).Previous Linear Programming and Cellular Automaton algorithms for space filling by circular solids and drops of fluids are random circle packings of 2-D space which are not necessarily close. Random Close Packing is described in 461.
135.23 Cellular Automata and Chaos, Berlekamp-Conway-Guy Life Cellular Automata - https://plato.stanford.edu/entries/cellular-automata/
135.24 Edge of Chaos, Cellular Automata - http://math.hws.edu/eck/js/edge-of-chaos/CA-info.html - each row is an 1D set of squares colored by states and subsequent rows evolve based on rules defined on states of a square and its neighbours. This is precisely similar to evolving tableau of Turing Machine states in Cook-Levin Theorem - https://en.wikipedia.org/wiki/Cook%E2%80%93Levin_theorem#/media/File:CookLevin_svg.svg and thus cellular automata are as powerful as Universal Turing Machines
135.25 Edge of Chaos, Lambda - Graphic illustration - http://math.hws.edu/eck/js/edge-of-chaos/CA.html - changing the fraction of rules (Christopher Langton's lambda) from 0 to 1 creates a phase transition from orderly to chaotic filling of 2D plane. Changing the previous Cellular Automaton algorithm for space filling in 2D to fill the neighbouring squares of a pseudorandomly chosen square by set of user defined rules (as against plain simple increment-decrement earlier) could exhibit chaotic behaviour especially on multichromatic filling of 2D plane.

(FEATURE- DONE) 136. Commits as on 28,29 January 2015

Python parser script to translate Longest Common Substring extracted from clustered or classified
set of encoded strings has been added to repository.

(FEATURE - DONE) 137. Mining the Astronomical Datasets using KMeans and kNN -

sequence of algorithms in AstroInfer
- Commits as on 4 February 2015
---
1. MaitreyaToEnchoros.py - This scripts reads a text file with set of date,time and long/lat for a specific class of events (at present it has earthquakes of 8+ magnitude from 1900). This script creates two encoded files asfer.enchoros.zodiacal and asfer.enchoros.ascrelative using Maitreya's Dreams opensource CLI.
2. asfer.cpp is executed with doClustering=true by which kNN and KMeans clustered data are obtained.
3. asferkmeansclustering.cpp and asferkNNclustering.cpp implement the KMeans and kNN respectively.
4. Set of strings from any of the clusters has to be written manually in asfer.enchoros.clustered
5. asferlongestcommonsubstring.cpp - Within each cluster a pairwise longest common substring is found out.
6. TranslateClusterPairwiseLCS.py - Translates the longest common substring to textual rule description mined from above dataset. Mined rules are grep-ed in python-src/MinedRulesFromDatasets.earthquakes. Thus the astronomy dataset cpp-src/earthquakesFrom1900with8plusmag.txt (or cpp-src/magnitude8_1900_date.php.html) ends up as the set of automatically mined rules.

(FEATURE - DONE) 138. Commandline sequence for Mining Astronomical Datasets (e.g Earthquakes as above)
using KMeans and kNN - Commits as on 4 February 2015

138.1 In asfer.cpp, set doClustering=true and build asfer binary
138.2 Set asfer.enchoros to asfer.enchoros.ascrelative or asfer.enchoros.zodiacal
138.3 $./asfer 2>&1 > logfile1
138.4 From logfile1 get maximum iteration clustered data for any cluster and update asfer.enchoros.clustered
138.5 In asfer.cpp set doLCS=true and build asfer binary
138.6 $./asfer 2>&1 > logfile2
138.7 grep "Longest Common Substring for" logfile2 2>&1 > python-src/asfer.ClusterPairwiseLCS.txt
138.8 sudo python TranslateClusterPairwiseLCS.py | grep "Textually" 2>&1 >> MinedRulesFromDatasets.earthquakes

---
(FEATURE - DONE) 139. BigData Analytics subsystem (related to point 64,65 on software analytics)

139.1 (DONE) As mentioned in commit notes(13February2015) below, new multipurpose Java Hadoop MapReduce
code has been added to a bigdata_analytics/ directory. At present it computes the frequencies of astronomical entities in the MinedRulesFromDatasets textfile obtained from clustering+LCS.

139.2 VIRGO Linux Kernel has following design choices to interface with the machine learning code
in AsFer :
	139.2.1 (DONE) the kernel module does an upcall to userspace asfer code - already this facility exists in VIRGO linux drivers
	139.2.2 (INITIAL VIRGO COMMITS - DONE) the analytics subsystem creates a policy config file /etc/virgo_kernel_analytics.conf having key-value pairs for each config variable by mining the datasets with classification+clustering+any-

hadoop-based-analytics. This is read by a VIRGO Linux kernel module - kernel_analytics - with VFS calls and sets(and exports symbols) the runtime config variables that indirectly determine the kernel behaviour. These exported analytics variables can be read by other interested kernel modules in VIRGO Linux, USB-md and KingCobra (and obviously mainline kernel itself). An example Apache Spark python script which mines the most frequent IP address from Uncomplicated Fire Wall and some device driver info from logs in /var/log/kern.log and /var/log/udev which can be set as a config key-value in /etc/virgo_kernel_analytics.conf has been added to AsFer repository. Dynamic setting of kernel analytics key-value pairs has been implemented with Boost::Python C++ and CPython C Extensions which obviates /etc/virgo_kernel_analytics.conf reload.

139.2.3 (INITIAL ASFER COMMITS - DONE) kernel module implements the machine learning algorithm in C (C++ in kernel is not preferable - http://harmful.cat-v.org/software/c++/linus) which doesn't reuse existing code and hence less attractive. Despite this caveat, a PoC Boost::Python C++ invocation of VIRGO memory system calls works - VIRGO linux has been made an extension Python C++ and C module which is invoked from Python userspace. Commits for this are described in 217. This is a tradeoff between pure C kernelspace and pure python/C/C++ userspace.

Diagrams depicting the above options have been uploaded at: http://sourceforge.net/p/asfer/code/HEAD/tree/asfer-docs/AsFerVIRGOLinuxKernelInterfaceDesignChoices.jpg

---
(FEATURE - DONE) Commits as on 4 February 2015

C++ implementation of Knuth-Morris-Pratt String Match Algorithm added to repository.

(FEATURE - DONE) Commits as on 9 February 2015

Python+R script for DFT analysis of multimedia data has been added to repository.

(FEATURE - DONE) Commits as on 13 February 2015

Initial commits for BigData Analytics (for mined astro datasets) using Hadoop 2.6.0 MapReduce:

- new folder bigdata_analytics has been added
- hadoop_mapreduce is subfolder of above
- A Hadoop Java MapReduce implementation to compute frequencies of astronomical entities in MinedRulesFromDatasets.earthquakes (obtained from clustering+LCS earlier in python-src) has been added with compiled bytecode and jar(MinedRulesFromDatasets_MapReducer.java)
- This jar was executed on a Single Node Hadoop Cluster as per the documentation at:
  - http://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
  - http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html
- The HDFS filesystem data is in hdfs_data - /user/root output with frequencies are in userroot_output and namenode and datanode logs are in

hadooproot_logs(part_r_0000 has the frequencies)
- hdfs_data also has commandlines history and the console output logs for above.

(FEATURE - DONE) Commits as on 17 February 2015

Python implementations for LogLog and HyperLogLog cardinality estimation for streamed multiset data have been added to repository.

(FEATURE - DONE) Commits as on 18 February 2015

Python implementations for CountMinSketch(frequencies and heavy-hitters) and BloomFilter(membership) for Streaming Data have been added to repository.

(FEATURE - DONE) Commits as on 19,20 February 2015

Python clients for Apache Hive and HBase have been added to repository and invoked from
Stream Abstract Generator script as 2-phase streamer for Streaming_<algorithm> scripts.
Hive,Pig,HBase HDFS and script data added to repository in java-src/bigdata_analytics.

(FEATURE - DONE) 140. Schematic for Apache Cassandra/Hive/HBase <=> AsFer Streaming_<algorithm> scripts interface
---

($) BigData Source(e.g MovieLens) => Populated in Apache Cassandra, Apache HiveQL CLI(CREATE TABLE..., LOAD DATA LOCAL INPATH...),HBase shell/python client(create 'stream_data','cf'...) (or) Apache PigLatin

($) Apache Hive or HBase SQL/NoSQL or Cassandra table => Read by Apache Hive or HBase or Cassandra python client => StreamAbstractGenerator => Iterable,Generator => Streaming_<algorithm> scripts

---
(FEATURE - DONE) Commits as on 25 February 2015

Added the Hive Storage client code for Streaming Abstract Generator __iter__ overridden function. With this Streaming_<algorithm> scripts can instantiate this class which mimicks the streaming through iterable with three storages - HBase, File and Hive.

(DONE) 141. Classpaths required for Pig-to-Hive interface - for pig grunt shell in -x local mode
---
1. SLF4J jars
2. DataNucleus JDO, core and RDBMS jars (has to be version compatible)
in pig/lib and hive/lib directories:
datanucleus-api-jdo-3.2.6.jar
datanucleus-core-3.2.10.jar
datanucleus-rdbms-3.2.9.jar
3. Derby client and server jars
4. Hive Shims jars
hive-shims-0.11.0.jar
5. All Hadoop core jars in /usr/local/hadoop/share/hadoop:
common hdfs httpfs kms mapreduce tools yarn
6. HADOOP_CONF_DIR($HADOOP_HOME/etc/hadoop) is also required in classpath.

(DONE) 142. Classpaths required for Pig-to-HBase interface

In addition to the jars above, following cloudera trace jars are required:
htrace-core-2.01.jar and htrace-1.45.jar

hadoop2_core_classpath.sh shell script in bigdata_analytics exports all the jars
in (141) and (142).

(FEATURE - DONE) Commits as on 26 February 2015

Pig-HBase stream_data table creation and population related Pig scripts, HDFS
data and screenshots have been added to repository.

(FEATURE - DONE) Commits as on 27 February 2015

Cassandra Python Client has been added to repository and
Streaming_AbstractGenerator.py has been
updated to invoke Cassandra in addition to Hive,HBase and File storage.
Cassandra data have been added
in bigdata_analytics/

---
143. (FEATURE - DONE) Storage Abstraction in AsFer - Architecture Diagram
---
Architecture diagram for Hive/Cassandra/HBase/File NoSQL and other storage
abstraction implemented in python has been uploaded at:
http://sourceforge.net/p/asfer/code/HEAD/tree/asfer-docs/BigDataStorageAbstracti
onInAsFer.jpg

---
144. (FEATURE - DONE) Apache Spark and VIRGO Linux Kernel Analytics and Commits
as on 6 March 2015
---
1. Prerequisite: Spark built with Hadoop 2.6.0 with maven commandline:
mvn -Pyarn -Phadoop-2.4 -Dhadoop.version=2.6.0 -DskipTests package

2. Spark Python RDD MapReduce Transformation script for parsing the most
frequent source IP address
from Uncomplicated FireWall logs in /var/log/kern.log has been added to
repository. This parsed
IP address can be set as a config in VIRGO kernel_analytics module
(/etc/virgo_kernel_analytics.conf)

(FEATURE - DONE) Commits as on 10 March 2015

Spark python script updated for parsing /var/log/udev and logs in
python-src/testlogs.
(spark-submit Commandline: bin/spark-submit /media/shrinivaasanka/0fc4d8a2-1c74-
42b8-8099-9ef78d8c8ea22/home/kashrinivaasan/KrishnaiResearch_OpenSource/asfer-
code/python-src/SparkKernelLogMapReduceParser.py 2>
/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-9ef78d8c8ea22/home/kashrinivaasan/
KrishnaiResearch_OpenSource/asfer-code/python-src/testlogs/
Spark_Logs.kernlogUFWandHUAWEIparser.10March2015)

---
145. (FEATURE - DONE) Commits as on 26 March 2015

New python script to fetch stock quotes by ticker symbol that can be used for
Streaming_<algorithm> scripts has been added.

---
146. (FEATURE - DONE) Commits as on 2 April 2015

New Python script to get Twitter tweets stream data for a search query has been added.

---
147. (FEATURE - DONE) Related to Item 3 - Sequence Mining Implementation - Commits as on 3 April 2015

Python class that implements Apriori GSP algorithm for mining frequent subsequences in ordered sequences
dataset has been added to repository. Though exponential, together with Longest Common Subsequence, Apriori GSP gives a near accurate subsequences - befitting trend of the dataset. For example the logs added to repository print the candidate support after 5 iterations (which can be modified) for asfer.enchoros dataset thereby eliciting a pattern in astronomical objects.

148. An example Class Association Rule Learnt from Sequence Mining:

For length 6 most frequent subsequences in earthquake astronomical data from 1900 are: ('conjoinedplanets',frequency)

[('0', 313), ('4', 313), ('8', 313), ('3', 313), ('7', 313), ('2', 313), ('6', 313), ('1', 313), ('5', 313), ('9', 313), ('a', 313), ('14', 121), ('46', 66), ('56', 51), ('13', 51), ('34', 42), ('45', 36), ('9a', 33), ('79', 31), ('24', 30), ('16', 30), ('69', 29), ('7a', 29), ('146', 29), ('36', 26), ('12', 25), ('57', 24), ('68', 23), ('6a', 22), ('49', 22), ('47', 22), ('28', 21), ('58', 21), ('8a', 21), ('78', 21), ('134', 20), ('25', 20), ('35', 20), ('23', 18), ('29', 18), ('48', 18), ('27', 17), ('3a', 17), ('4a', 17), ('59', 16), ('37', 16), ('67', 15), ('5a', 14), ('17', 14), ('38', 13), ('346', 12), ('26', 11), ('124', 11), ('39', 11), ('15', 11), ('145', 11), ('14a', 10), ('456', 9), ('348', 8), ('2a', 8), ('247', 7), ('345', 7), ('1a', 7), ('469', 7), ('147', 7), ('356', 7), ('378', 6), ('156', 6), ('69a', 5), ('249', 5), ('179', 5), ('167', 5), ('358', 5), ('37a', 4), ('347', 4), ('279', 4), ('46a', 4), ('79a', 4), ('568', 4), ('169', 4), ('57a', 4), ('18', 4), ('1456', 4), ('679', 4), ('149', 4), ('359', 4), ('137', 3), ('135', 3), ('1469', 3), ('13a', 3), ('368', 3), ('1345', 3), ('123', 3), ('125', 3), ('268', 3), ('1346', 3), ('24a', 3), ('1347', 3), ('56a', 3), ('1256', 3), ('1348', 3), ('59a', 3), ('457', 3), ('256', 3), ('1468', 3), ('468', 3), ('467', 3), ('1356', 3), ('168', 3), ('1378', 3), ('19', 3), ('234', 3), ('68a', 3), ('1247', 3), ('148', 3), ('136', 2), ('67a', 2), ('29a', 2), ('349', 2), ('167a', 2), ('38a', 2), ('123a', 2), ('78a', 2), ('245', 2), ('247a', 2), ('248', 2), ('4568', 2), ('1248', 2), ('258', 2), ('1247a', 2), ('359a', 2), ('158', 2), ('36a', 2), ('47a', 2), ('23a', 2), ('12a', 1), ('26a', 1), ('1349', 1), ('2349', 1), ('139', 1), ('1467', 1), ('349a', 1), ('678', 1), ('49a', 1), ('126', 1), ('127', 1), ('34a', 1), ('379', 1), ('3568', 1), ('23469', 1), ('2349a', 1), ('179a', 1), ('124a', 1), ('3479', 1), ('369', 1), ('27a', 1), ('39a', 1), ('458', 1), ('459', 1), ('578', 1), ('16a', 1), ('259', 1), ('257', 1), ('1678', 1), ('368a', 1), ('17a', 1), ('134a', 1), ('1458', 1), ('159', 1), ('3469', 1), ('2345', 1), ('2346', 1), ('13479', 1), ('479', 1), ('169a', 1), ('469a', 1)]

---
indices for planets:
---
 * 0 - for unoccupied
 * 1 - Sun
 * 2 - Moon
 * 3 - Mars
 * 4 - Mercury
 * 5 - Jupiter
 * 6 - Venus

 * 7 - Saturn
 * 8 - Rahu
 * 9 - Ketu

Some inferences can be made from above:

By choosing the creamy layer of items with support > 30 (out of 313 historic events) - ~10%:
[('14', 121), ('46', 66), ('56', 51), ('13', 51), ('34', 42), ('45', 36), ('9a', 33), ('79', 31), ('24', 30), ('16', 30)]

Above implies that:

Earthquakes occur most likely when, following happen - in descending order of frequencies:
- Sun+Mercury (very common)
- Mercury+Venus
- Jupiter+Venus
- Sun+Mars
- Mars+Mercury
- Mercury+Jupiter
- Ketu is in Ascendant
- Saturn+Ketu
- Moon+Mercury
- Sun+Venus

Machine Learnt pattern above strikingly coincides with some combinations in Brihat Samhita (Role of Mars, Nodes, and 2 heavy planets, Mercury-Venus duo's role in weather vagaries) and also shows some new astronomical patterns (Sun+Mars and Jupiter+Venus as major contributors). Above technique is purely astronomical and scientific with no assumptions on astrology.

Some recent major intensity earthquakes having above sequence mined astronomical conjunctions :

Sendai Earthquake - 11 March 2011 14:46 - Sun+Mars in Aquarius, Mercury+Jupiter in Pisces
Nepal Earthquake - 25 April 2015 11:56 - Sun+Mars+Mercury in Aries
Chile Earthquake - 16 September 2015 22:55 - Sun+Mars+Jupiter in Leo
All three have Sun+Mars coincidentally.

(FEATURE - DONE) Commits as on 5 April 2015

Textual translation of Class Association Rules added to SequenceMining.py with logs.

(FEATURE - DONE) Commits as on 13 April 2015

Python implementation of :
   - Part-of-Speech tagging using Maximum Entropy Equation of Conditional Random Fields(CRF) and
   - Viterbi path computation in HMM-CRF for Named Entity Recognition has been added to repository.

(FEATURE - DONE) Commits as on 15 April 2015

Named Entity Recognition Python script updated with:
  - More PoS tags
  - Expanded HMM Viterbi probabilities matrix

- Feature function with conditional probabilities on previously labelled word

---
(FEATURE - DONE) Commits as on 17 April 2015

Twitter Streaming python script updated with GetStreamFilter() generator object for streaming tweets data.

(FEATURE - DONE) 149. Commits as on 10 May 2015

A Graph Search NetworkX+MatPlotLib Visualizer for WordNet has been implemented in python and has been added to repository. This is an initial code and more algorithms to mine relationships within a text data would be added using WordNet as ontology - with some similarities to WordNet definition subgraph obtained in http://sourceforge.net/p/asfer/code/HEAD/tree/python-src/InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit.py. But in this script visualization is more important.

(FEATURE - DONE) Commits as on 12 May 2015

WordNet Visualizer script has been updated to take and render the WordNet subgraph computed by Recursive Gloss Overlap algorithm in:
- http://arxiv.org/abs/1006.4458
-
http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf
- https://sites.google.com/site/kuja27/PresentationTAC2010.pdf?attredirects=0

(FEATURE - DONE) 150. Commits as on 14 May 2015

Updated WordNet Visualizer with:
- bidirectional edges
- graph datastructure changed to dictionary mapping a node to a list of nodes (multigraph)
- With these the connectivity has increased manifold as shown by gloss overlap
- the graph nodes are simply words or first element of lemma names of the synset
- more networkx drawing layout options have been added

Added code for:
- removing self-loops in the WordNet subgraph
- for computing core number of each node (maximum k such that node is part of k-core decomposition)

In the input example, nodes "India" and "China" have core numbers 15 and 13 respectively which readily classify the document to belong to
class "India and China". Thus a new unsupervised classifier is obtained based on node degrees.

(FEATURE - DONE) Commits as on 16 May 2015

- added sorting the core numbers descending and printing the top 10% core numbers which are prospective classes the document could belong to.
This is a new unsupervised text classification algorithm and is based on recursive gloss overlap algorithm in http://arxiv.org/abs/1006.4458 and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf. A major feature of this algorithm is that it is more precise in finding the class of a document as intrinsic merit is computed by mapping a text to subgraph of wordnet, than conventional text classification and clustering based on distance metrics and training data.

(FEATURE - DONE) 151. Commits as on 18 May 2015

An interesting research was done on the wordnet subgraph obtained by Recursive Gloss Overlap algorithm:
- PageRank computation for the WordNet subgraph was added from NetworkX library. This is probably one of the first applications of PageRank to
a graph other than web link graph.
- The objective was to find the most popular word in the subgraph obtained by the Random Walk Markov Model till it stabilized (PageRank)
- The resultant word with maximum pagerank remarkably coincides with the most probable class of the document and is almost similar in
ranking to core numbers of nodes ranked descending.
- Some random blog text was used.
- Above was mentioned as a theoretical statement in 5.12 of http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf

---
(FEATURE - DONE) Commits as on 19 May 2015

A primitive text generation from the definition graph has been implemented by:
- computing the k-core of the graph above certain degree k so that the core is dense
- each edge is mapped to a relation - at present "has" , "is in" are the only 2 relations (more could be added based on
hypernyms)
- Each edge is output as "X <relation> Y"
- Above gives a nucleus text extracted from the original document
- It implements the theory mentioned in:
https://sites.google.com/site/kuja27/DocumentSummarization_using_SpectralGraphTheory_RGOGraph_2014.pdf?attredirects=0&d=1

(FEATURE - DONE) Commits as on 20 May 2015

Hypernyms/Hyponyms based sentence construction added to WordNet Visualizer. The core-number and pagerank based classifier was tested with more inputs. The accuracy and relevance of the word node with topmost core number and pagerank for RGO graph looks to be better than the traditional supervised and unsupervised classifiers/clusterers. The name of the class is inferred automatically without any training inputs or distance metrics based only on density of k-core subgraphs.

(FEATURE - DONE) Commits as on 21 May 2015

Updated the Visualizer to print the WordNet closure of hypernyms and hyponyms to generate a blown-up huge sentence. The closure() operator
uncovers new vertices and edges in the definition graph (strict supergraph of RGO graph) and it is a mathematically generated sentence
(similar to first order logic and functional programming compositionality) and mimicks an ideal human-thinking process (psychological process of evocation which is complete closure in human brain on a word utterance).

(FEATURE - DONE) 151. Commits as on 22,23,24 May 2015

Added code for computing recursive composition of lambda functions over the RGO graph relations(edges):
- For this the Depth First Search Tree of the graph is computed with NetworkX
- The lambda function is created for each edge
- Composition is done by concatenating a recursive parenthesisation string of lambda expressions:
      for each DFS edge:

Composed_at_tplus1 = Composed_at_t(lambda <edge>)
- DFS is chosen as it mimicks a function invocation stack
- Lambda composition closure on the Recursive Gloss Overlap graph has been rewritten to create a huge lambda composition string with parentheses. Also two new lambda function relations have been introduced - "subinstance of" and "superinstance of". These are at present most likely do
not exist in WordNet parlance, because the Recursive Gloss OVerlap algorithm grows a graph recursively from WordNet Synset definition tokenization.
- Thus a tree is built which is made into a graph by pruning duplicate edges and word vertices. WordNet graph does not have a depth. RGO algorithm adds one more dimension to WordNet by defining above relations. These are different from hyper/hypo-nymns. During recursive gloss overlap, if Y is in Synset definition of X, Y is " subinstance of" X and X is "superinstance of" Y.
- Thus RGO in a way adds new hierarchical recursive relationships to WordNet based on Synset definition.
- The lambda composition obtained above is a mathematical or lambda calculus representation of a text document - Natural Language reduced to Lambda calculus compositionality operator.

(FEATURE - DONE) Commits as on 26 May 2015

Added initial code for Social Network Analyzer for Twitter following to create a NetworkX graph and render it with MatPlotLib.

(FEATURE - DONE) Commits as on 3 June 2015

Bonacich Power Centrality computation added to Twitter Social Network Analyzer using PageRank computation.

(FEATURE - DONE) Commits as on 4 June 2015

- Eigen Vector Centrality added to Twitter Social Network Analyzer.
- File storage read replaced with Streaming_AbstractGenerator in Streaming_HyperLogLogCounter.py and Streaming_LogLogCounter.py

---
(FEATURE - DONE) 152. Commits as on 7 June 2015

New Sentiment Analyzer script has been added to repository:
- This script adds to WordNet Visualizer with Sentiment analysis using SentiWordNet from NLTK corpus
- added a Simple Sentiment Analysis function that tokenizes the text and sums up positivity and negativity score
- A non-trivial Sentiment Analysis based on Recursive Gloss Overlap graph - selects top vertices with high core numbers
and elicits the positivity and negativity of those vertex words.
- Above is intuitive as cores of the graph are nuclei centering the document and the sentiments of the vertices of those cores
are important which decide the sentiment of the whole document.
- Analogy: Document is akin to an atom and the Recursive Gloss Overlap does a nuclear fission to extricate the inner strucure of a document
(subatomic particles are the vertices and forces are edges)
- Instead of core numbers, page_rank can also be applied (It is intriguing to see that classes obtained from core_numbers
and page_rank coincide to large extent) and it is not unusual to classify a document in more than one class (as it is more realistic).

Above script can be used for any text including social media and can be invoked as a utility from SocialNetworkAnalysis_<brand> scripts.
Microblogging tweets are more crisp and "emotionally precise" i.e. extempore - convey instantaneous sentiment (without much of a preparation)

(FEATURE - DONE) Commits as on 8 June 2015

Commits for:
 - fixing errors due to NLTK API changes in lemma_names(), definition() ...
[variables made into function calls]
 - Unicode errors
 - Above were probably either due to Ubuntu upgrade to 15.04 which might have
added some unicode dependencies and/or recent changes to NLTK 3.0
   (SentimentAnalyzer.py already has been updated to reflect above)

(FEATURE - DONE) Commits as on 10 June 2015

Sentiment Analysis for twitter followers' tweets texts added to
SocialNetworkAnalysis_Twitter.py which invokes SentimentAnalyzer.py

---
(FEATURE - DONE) Commits as on 13 June 2015

Sentiment Analysis for tweet stream texts added to
SocialNetworksAnalysis_Twitter.py which invokes SentimentAnalyzer.py

---
---
153. (FEATURE - THEORY - Minimum Implementation DONE) Sentiment Analysis as
lambda function composition of the Recursive Gloss Overlap WordNet subgraph
---

SentiWordNet based scoring of the tweets and texts have some false positives and
negatives. But the lambda composition of the
Recursive Gloss Overlap graph is done by depth-first-search - set of composition
trees which overlap. If each edge is replaced by
a function of sentiment scores of two vertex words, then lambda composition
performed over the graph is a better representation of sentiment
of the document. This is at present a conjecture only. An implementation of this
has been added to SentimentAnalyzer.py by doing a Belief
Propagation of a sentiment potential in the Recursive Gloss Overlap graph
considering it as a Bayesian graphical model.

Commits as on 15 June 2015

NeuronRain - (AsFer+USBmd+VIRGO+KingCobra) - version 15.6.15 release tagged
Most of features and bugfixes are in AsFer and VIRGO

---
154. (FEATURE - DONE) Belief Propagation and RGO - Commits as on 20 June 2015

SentimentAnalyzer is updated with a belief propagation algorithm (Pearl) based
Sentiment scoring by
considering Recursive Gloss Overlap Definition Graph as graphical model and
sentiwordnet score for
edge vertices as the belief potential propagated in the bayesian network (RGO
graph). Since the
sentiment is a collective belief extracted from a text rather than on isolated
words and the
sentiwordnet scores are probabilities if normalized, Sentiment Analysis is
reduced to Belief Propagation problem.
The previous Belief Propagation function is invoked from

SocialNetworkAnalysis_<> code to Sentiment Analyze tweet stream.

---

155. Updates to https://sites.google.com/site/kuja27/DocumentSummarization_using_SpectralGraphTheory_RGOGraph_2014.pdf?attredirects=0&d=1

Algorithms and features based on Recursive Gloss Overlap graph:
155.1 RGO visualizer
155.2 RGO based Sentiment Analyzer - Belief Propagation in RGO as a graphical model
155.3 Graph Search in text - map a text to wordnet relations by RGO graph growth - prints the graph edges which are relations hidden in
a text
155.4 New unsupervised text classifier based in RGO core numbers and PageRank
155.5 Social Network Analyzer (Tweets analysis by RGO graph growth)
155.5 Lambda expression construction from the RGO graph
155.6 Sentence construction by composition of edges

---

156. (FEATURE - DONE) Commits as on 4 July 2015 - RGO sentiment analyzer on cynical tweets

---

Added query for a "elections" to stream tweets related to it. Few sample cynical tweets were sentiment-analyzed with
RGO Belief Propagation SentimentAnalyzer that gives negative sentiment scores as expected whereas trivial SentiWordNet score summation
gives a positive score wrongly.

---

157. (THEORY) Recursive Gloss Overlap graphical model as a Deep Learning algorithm that is an alternative to Perceptrons - for text data

---

Point 155 mentions the diverse applications of Recursive Gloss Overlap algorithm each of which touch upon multiple facets of Machine Learning (or) Deep Learning. Instead of a multi-layered perceptron and a backpropagation on it which have the standard notation (W . X + bias) and have to be built, the wordnet RGO subgraph presents itself readily as a deep learning model without any additional need for perceptrons with additional advantage that complete graph theory results apply to it(core numbers, pageranks, connectivity etc.,), making it a Graph-theoretical learning instead of statistical (for text data).

---

158. (THEORY) Graph Discovery (or) Graph Guessing and EventNet (related to EventNet points 70-79)

EventNet mentioned previously is an infinite cause-effect graph of event vertices with partakers for each event
(kind of a PetriNet yet different). A special case of interest is when only few subgraphs of a giant EventNet is available and it is
often necessary to discover or guess the complete graph of causality with partakers. A familiar example is a crime scene investigation
where:
 - it is necessary to recreate the complete set of events and their causalities with partakers of a crime (reverse-engineering)
 - but only few clues or none are available - which are akin to very sparse subgraphs of the crime scene EventNet
 - Guessing (100-x)% of the graph from x% clue subgraphs can be conjectured to

be an NP problem - because non-deterministic polynomially an EventNet path can be guessed. As a counting problem this is #P-complete (set of all paths among known subgraphs of an unknown supergraph). But there is only one accepting path (could be in Unambigous Logspace if the number of states is in logspace)

Another example: set of blind men try to make out an object by touch.

---

159. (THEORY) Pattern Grammar and Topological Homeomorphism of Writing Deformations

---

Grammar for patterns like texts, pictures similar to RE,CFG etc., for pattern recognition:

- example text grammar: <a> := <o> <operator> <)>
- example text grammar: <d> := <o> <operator> <l>
- example text grammar: <v> := <\> <operator> </>

Previous grammar is extensible to any topological shapes. For example, handwriting of 2 individuals are homeomorphic deformations.

References:

159.1 Topology and Graphics - https://books.google.co.in/books?id=vCc8DQAAQBAJ&pg=PA263&lpg=PA263&dq=homeomorphism+and+handwriting&source=bl&ots=L_9kNgTcmF&sig=l-PfRL_jjcuF0L-2dJ5rukrC4CM&hl=en&sa=X&ved=0ahUKEwji8bDJ683QAhWBQY8KHQ3qAuEQ6AEIHzAA#v=onepage&q=homeomorphism%20and%20handwriting&f=false

---

160. (THEORY) Cognitive Element

An element of a list or group is cognitive with respect to an operator if it "knows" about all or subset of other elements of list or group where "knowing" is subject to definition of operation as below:

- In list 2,3,5,7,17 - 17 knows about 2,3,5,7 in the sense that 17 = 2+3+5+7 with + operator
- In list "addon","add","on" - "addon" knows about "add" and "on" with concatenation operator
- In list "2,3,6,8,9,10,..." - 6 knows about 2 and 3, 10 knows about 2 and 5 with factorization as operator.

This might have an equivalent notion in algebra. Motivation for this is to abstract cognition as group operator. Essentially, this reduces to an equivalence relation graph where elements are related by a cognitive operator edge.

---

161. (THEORY) Philosophical intuition for P(Good) summation - Continuation of (129) and other previous related points.

The apparent paradox in Perfect Voting can be reconciled as below to some extent:

- Voting is done in parallel in real-world elections and not serially
- Votes are counted in parallel - iterated integer addition in NC1 (http://www.ccs.neu.edu/home/viola/classes/gems-08/lectures/le8.pdf)
- Thus if RHS is just Circuit-Value Problem and not a Circuit-SAT then LHS getting equated to RHS is not unusual - both can be NC or P-Complete
- Separation of RHS from LHS happens only when the Circuit-SAT is required in

RHS.
- P(Good) convergence in perfect voting implies that Voter SAT is not necessary if there is perfection (proving the obvious). This is
true even though the PRG is imperfect that operates to choose on a perfect set.
- Parallelism implies NC circuits
- Above applies to infinite majority also (Mark Fey)
- RHS in worst-case can be EXP-Complete if the Majority+Voter SAT oracle circuit is of unrestricted depth and lot of variables are common across all voters. Proving EXP-completeness is ignored as it is evident from definition of DC circuits. Thus LHS has a P or NC algorithm to RHS EXP-Complete problem (something seemingly preposterous unless perfection is prohibited and voters do not have common variables in their boolean functions)

---
162. (THEORY) Majority Voting Circuit with Boolean Function Oracles
---
If each voter has a boolean function to decide which has fanout > 1 gates instead of a formula, RHS Majority Voting becomes generic.
Decision tree, Certificate, and other complexity measures automatically come into reckoning. Thus RHS circuit is an infinite (non-uniform) majority circuit with boolean function circuit DAGs or oracles.

References:

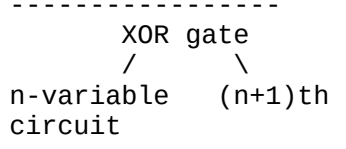
162.1 http://www.cs.cmu.edu/~odonnell/papers/barbados-aobf-lecture-notes.pdf
162.1 http://www.math.u-szeged.hu/~hajnal/research/papers/dec_surv.gz

---
163. (THEORY) Parity and Constant Depth - intuition (not a formal proof, might have errors)

Inductive Hypothesis:

For depth d and n variables parity has n^k sized circuit.

For n+1 variables:



Each XOR gate is of atleast depth 2 or 3 with formula - (x /\ not y) V (not x /\ y). Thus for each additional variable added depth gets added by atleast 2 contradicting the constant depth d in hypothesis above though size is polynomial.

---
164. (FEATURE - DONE) Commits as on 16 September 2015
---
cpp-src/cloud_move - Implementation of Move Semantics for Cloud objects:

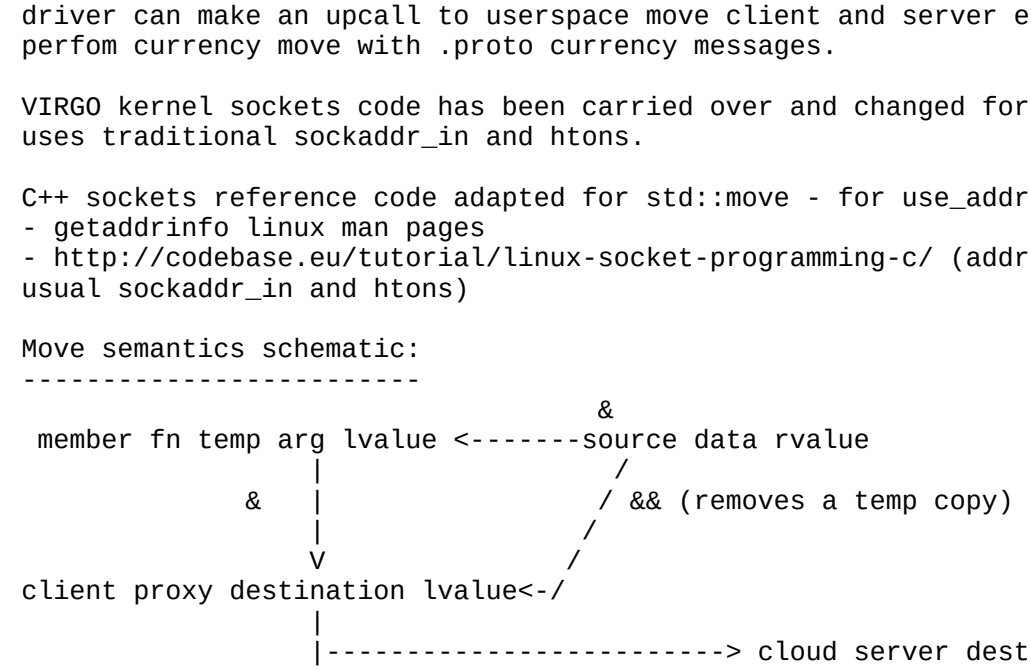
This expands on the usual move semantics in C++ and implements a Perfect Forwarding of objects over cloud. A move client invokes the T&& rvalue reference move constructor to move(instead of copying) a local memory content to a remote machine in userspace. Functionality similar to this in kernelspace is required for transactional currency move in KingCobra - http://sourceforge.net/p/kcobra/code-svn/HEAD/tree/KingCobraDesignNotes.txt and

https://github.com/shrinivaasanka/kingcobra-github-code/blob/master/KingCobraDesignNotes.txt. Though kernel with C++ code is not advised ,kingcobra driver can make an upcall to userspace move client and server executables and perfom currency move with .proto currency messages.

VIRGO kernel sockets code has been carried over and changed for userspace that uses traditional sockaddr_in and htons.

C++ sockets reference code adapted for std::move - for use_addrinfo clause:
- getaddrinfo linux man pages
- http://codebase.eu/tutorial/linux-socket-programming-c/ (addrinfo instead of usual sockaddr_in and htons)

Move semantics schematic:



lvalue reference& does additional copy which is removed by rvalue reference&& to get the rvalue directly. Move client proxies the remote object and connects to Move server and the object is written over socket.

---

165. (FEATURE - DONE) Commits as on 18 September 2015

Cloud Perfect Forwarding - Google Protocol Buffer Currency :

Currency object has been implemented with Google Protocol Buffers - in cloud_move/protocol_buffers/ src_dir and out_dir directories. Currency has been defined in src_dir/currency.proto file. Choice of Protocol Buffer over other formats is due to:
- lack of JSON format language specific compilers
- XML is too complicated
- Protocol Buffers also have object serialization-to-text member functions in generated C++ classes.

Protocol Buffer compilation after change to currency object:
protoc -I=src_dir/ --cpp_out=out_dir/ src_dir/currency.proto

---

166. (THEORY) Debug Analytics,Software Analytics,Automated Debugging - 24 November 2019 - related to 65, 581

Debugging software is painful process with repetitive labour - For example kernel and device driver development has the following lifecycle:
1. Writing the kernel patch code for some deep kernel panics.
2. Kernel incremental or full build.
3. Test the patch.
(1),(2) and (3) are sometimes frustratingly repetitive. If the debugging is represented as a state machine automaton, the cycles in automaton are the time consuming phases. Thus Debug Analytics can be defined as the problem to find the most efficient debug state machine automaton without cycles or with least number of cycles and cycles of least length.

Delta debugging algorithm bisects the code changes between two successive commits recursively and isolates the code snippet causing the regression in unit testing. Example: If revision r+1 has the patch set [p1,p2,p3,p4,p5] over revision r, delta debugging creates delta subsets [p1,p2],[p3],[p4,p5] and applies the deltas incrementally over r as paths in a binary search tree by union() and minus() primitives based on unit test pass or fail - for instance the path:

r+[p1,p2] - pass, r+[p1,p2] union [p3] - pass , r + [p1,p2,p3] union [p4,p5] - fail

isolates [p4,p5] delta as point of failure. minus() primitive is applied to [p1,p2,p3,p4,p5] and [p4,p5] are segregated to fail state in bisection. Program state reachability graphs are obtained by a fixed point of variables as vertices and edges amongst them are deduced from pointer analysis. These program states between two consecutive revision states create a stream of Program state graphs which is a Streaming Common Subgraph Mining problem. Correlating the common subgraphs in the program state graph stream and corresponding unit test pass-fail binary stream for sequence of patch revisions yields insights into the regression causing state subgraph:

... Graph(r)-Graph(r+delta) ...
... Pass-Fail ...

Debug Analytics goes further beyond isolating the buggy code and asks for minimum state debug automaton if any which resolves the bug.

References:

166.1 Learning Finite State Machines - https://www.cs.upc.edu/~bballe/other/phdthesis.pdf
166.2 Delta Debugging - Why Programs Fail - https://www.st.cs.uni-saarland.de/dd/ - Bisection in Git/SVN/Hg
166.3 Beautiful Code - [Oram-Wilson] - Chapter 28 - Beautiful Debugging by Andreas Zeller - DDD and Eclipse plugins for Delta Debugging algorithm - Code example 28.1 - computing common subgraphs between program state graphs - Fig 28.2 - Program State of GNU C Compiler - Example of regression between gdb 4.16 and gdb 4.17 for DDD

---

167. (THEORY) Prestige based ranking and Condorcet Elections

Web Search Engine Rankings by prestige measures are n-tuples of rankings (for n candidate web pages where n could be few billions) which are not condorcet rankings. Arrow's Theorem for 3-candidate condorcet election allows a winner if and only if there is dictatorship. If there are m search engines each creating n-tuples of condorcet rankings (NAE tuples), then conjecturally there shouldn't be a voting rule or a meta search engine that circumvents circular rankings and produces a reconciled ranking of all NAE tuples from m search engines.

---

168. (FEATURE - DONE) Commits as on 20,21,22 October 2015

Initial code for LinkedIn crawl-scrape. Uses Python-linkedin from https://github.com/ozgur/python-linkedin with
some additional parsing for url rewriting and wait for authurl input:
- prints a linkedin url which is manually accessed through curl or browser
- created redirect_url with code is supplied to raw_input "authurl:"
- parses the authcode and logs-in to linkedin to retrieve profile data; get_connections() creates a forbidden error

Logs for the above has been added to testlogs

---

169. (FEATURE - DONE) Commits as on 28 October 2015

Deep Learning Convolution Perceptron Python Implementation.

---
170. (FEATURE - DONE) Commits as on 29 October 2015, 1 November 2015

Deep Learning BackPropagation Multilayered Perceptron Python Implementation.

---
171. Commits as on 3 November 2015

DeepLearning BackPropagation implementation revamped with some hardcoded data removal.

---
172. (FEATURE - DONE) Hurricane Datasets Sequence Mining - Commits as on 4 November 2015

Miscellaneous code changes:

- Weight updates in each iteration are printed in DeepLearning_BackPropagation.py
- Changed maitreya_textclient path for Maitreya's Dreams 7.0 text client; Updated to read HURDAT2 NOAA Hurricane Dataset (years 1851-2012) in MaitreyaToEncHoro.py
- Maximum sequence length set to 7 for mining HURDAT2 asfer.enchoros.seqmining
- New files asfer.enchoros.ascrelative.hurricanes, asfer.enchoros.zodiacal.hurricanes have been added which are created by MaitreyaToEncHoro.py
- an example chartsummary for Maitreya's Dreams 7.0 has been updated
- 2 logs for SequenceMining for HURDAT2 encoded datasets and Text Class Association Rules learnt by SequenceMining.py Apriori GSP have been added to testlogs/
- Increased number of iterations in BackPropagation to 100000; logs for this with weights printed are added in testlogs/; shows a beautiful convergence and error tapering (~10^-12)

==========
Sorted Candidate support for all subsequence lengths - gives an approximate pattern in dataset:
[('0', 1333), ('4', 1333), ('8', 1333), ('3', 1333), ('7', 1333), ('2', 1333), ('6', 1333), ('1', 1333), ('5', 1333), ('9', 1333), ('a', 1333), ('14', 613), ('46', 315), ('67', 261), ('78', 190), ('59', 189), ('49', 186), ('34', 180), ('57', 165), ('45', 161), ('13', 159), ('56', 158), ('58', 147), ('12', 134), ('23', 128), ('146', 116), ('36', 115), ('8a', 113), ('25', 108), ('9a', 106), ('346', 103), ('69', 99), ('26', 98), ('24', 95), ('134', 90), ('16', 83), ('6a', 81), ('35', 77), ('38', 75), ('149', 75), ('68', 73), ('467', 73), ('28', 71), ('2a', 71), ('569', 70), ('29', 66), ('367', 66), ('4a', 64), ('37', 59), ('567', 59), ('3a', 57), ('457', 56), ('27', 55), ('7a', 55), ('47', 51), ('145', 50), ('3467', 49), ('5a', 48), ('124', 44), ('79', 41), ('14a', 38), ('1346', 37), ('459', 36), ('17', 32), ('126', 29), ('246', 29), ('1459', 28), ('46a', 27), ('59a', 25), ('1a', 25), ('15', 25), ('156', 25), ('39', 24), ('345', 23), ('13467', 22), ('1345', 22), ('3457', 22), ('19', 22), ('13457', 22), ('1457', 21), ('234', 21), ('136', 20), ('1367', 20), ('2346', 18), ('3567', 18), ('356', 18), ('78a', 17), ('258', 16), ('259', 16), ('169', 16), ('135', 14), ('146a', 14), ('278', 14), ('67a', 13), ('49a', 13), ('469', 13), ('13567', 12), ('679', 12), ('178', 12), ('1356', 12), ('237', 12), ('3679', 12), ('4569', 11), ('678', 11), ('268', 11), ('456', 11), ('57a', 11), ('129', 10), ('28a', 10), ('238', 10), ('68a', 10), ('147', 10), ('23a', 10), ('459a', 10), ('267', 10), ('26a', 9), ('23467', 9), ('247', 9), ('16a', 9), ('256', 9), ('2569', 9), ('137', 8), ('249', 8), ('1246', 8), ('257', 8), ('36a', 8), ('168', 8), ('1567', 8), ('12a', 7), ('346a', 7), ('1459a', 7), ('245', 7), ('2459', 7), ('2367', 7), ('236', 7), ('1268', 6), ('567a', 6), ('25a', 6), ('1247', 6), ('79a', 5), ('47a', 5), ('149a', 4), ('29a', 4), ('1469', 4),

('126a', 4), ('19a', 4), ('2469', 4), ('37a', 3), ('679a', 3), ('24a', 3), ('367a', 3), ('268a', 3), ('1367a', 3), ('58a', 3), ('147a', 3), ('3679a', 3), ('357', 3), ('123', 2), ('69a', 2), ('246a', 2), ('56a', 2), ('1268a', 2), ('124a', 2), ('1234', 2), ('1249', 2), ('4569a', 2), ('1567a', 2), ('2459a', 2), ('1357', 2), ('168a', 2), ('35a', 2), ('569a', 2), ('1346a', 2), ('1246a', 2), ('349', 1), ('34a', 1), ('137a', 1), ('467a', 1), ('38a', 1), ('2345', 1), ('237a', 1), ('267a', 1), ('27a', 1), ('39a', 1), ('247a', 1), ('123467', 1), ('1247a', 1), ('134a', 1), ('13467a', 1), ('12345', 1), ('12346', 1), ('239', 1), ('3467a', 1)]
size of dataset = 1333

==========
Sun , Mercury ,
==========
Class Association Rule 12 mined from dataset: with frequencies: 23.6309077269 percentage
==========
Mercury , Venus ,
==========
Class Association Rule 13 mined from dataset: with frequencies: 19.5798949737 percentage
==========
Venus , Saturn ,
==========
Class Association Rule 14 mined from dataset: with frequencies: 14.2535633908 percentage
==========
Saturn , Rahu ,
==========
Class Association Rule 15 mined from dataset: with frequencies: 14.1785446362 percentage
==========
Jupiter , Ketu ,
==========
Class Association Rule 16 mined from dataset: with frequencies: 13.9534883721 percentage
==========
Mercury , Ketu ,
==========
Class Association Rule 17 mined from dataset: with frequencies: 13.503375844 percentage
==========
Mars , Mercury ,
==========
Class Association Rule 18 mined from dataset: with frequencies: 12.3780945236 percentage
==========
Jupiter , Saturn ,
==========
Class Association Rule 19 mined from dataset: with frequencies: 12.0780195049 percentage
==========
Mercury , Jupiter ,
==========
Class Association Rule 20 mined from dataset: with frequencies: 11.9279819955 percentage
==========
Sun , Mars ,
==========
Class Association Rule 21 mined from dataset: with frequencies: 11.8529632408 percentage
==========
Jupiter , Venus ,
==========

Class Association Rule 22 mined from dataset: with frequencies: 11.0277569392 percentage
==========
Jupiter , Rahu ,
==========
Class Association Rule 23 mined from dataset: with frequencies: 10.0525131283 percentage
==========
Sun , Moon ,
==========
Class Association Rule 24 mined from dataset: with frequencies: 9.60240060015 percentage
==========
Moon , Mars ,
==========
Class Association Rule 25 mined from dataset: with frequencies: 8.70217554389 percentage
==========
Sun , Mercury , Venus ,
==========
Class Association Rule 26 mined from dataset: with frequencies: 8.6271567892 percentage
==========
Mars , Venus ,
==========
Class Association Rule 27 mined from dataset: with frequencies: 8.47711927982 percentage
==========
Rahu , Ascendant ,
==========
Class Association Rule 28 mined from dataset: with frequencies: 8.10202550638 percentage
==========
Moon , Jupiter ,
==========
Class Association Rule 29 mined from dataset: with frequencies: 7.951987997 percentage
==========
Ketu , Ascendant ,
==========
Class Association Rule 30 mined from dataset: with frequencies: 7.72693173293 percentage
==========
Mars , Mercury , Venus ,
==========

Above are excerpts from the logs added to testlogs/ for Sequence mined from HURDAT2 dataset for hurricanes from 1851 to 2012. Similar to the sequence mining done for Earthquake datasets, some of the mined sequences above are strikingly similar to astronomical conjunctions already mentioned in few astrological classics (E.g Sun-Mercury-Venus - the legendary Sun flanked by Mercury and Venus, Mercury-Venus) and many others look quite new (e.g prominence of Venus-Saturn). Though these correlations can be dismissed as coincidental, these patterns are output automatically through a standard machine learning algorithm like Sequence Mining on astronomical datasets without any non-scientific assumptions whatsoever. Scientific corroboration of the above might require knowhow beyond purview of computer science - e.g Oceanography, Gravitational Effects on Tectonic Geology etc.,

==========
==========
173. (FEATURE - DONE) BigData Storage Backend Abstraction subsystem for AsFer - Commits as on 5 November 2015 :

```
--------------------------------
- These are initial minimal commits for abstracting storage of ingested bigdata
for AsFer Machine Learning code
- A dependency injection implementation of database-provider-specific objects is
added to repository.
- Presently MySQLdb based MySQL backend has been implemented with decoupled
configuration which are injected into Abstract Backend class to build an object
graph. Python injector library based on Google Guice Dependency Injection
Framework (https://pythonhosted.org/injector/) is used for this.  Logs for a
sample MySQL query has been added to testlogs/

Schematic Dependency Injected Object Graph:
----------------------------------------
                              Injector
============================================================

|

|

V
            MySQL_Configuration ---injects config ----> MySQL_DBBackend ----
injects connection ----> Abstract_DBBackend ---> exec_query()
                                                                            /
\

|
            xxx_Configuration ---injects config ----> xxx_DBBackend ---- injects
connection -----------|

------------------------------------------------------------------------------
----------------------------------------
174. (FEATURE - DONE) AsFer C++ - Python Integration - Embedding Python in AsFer
C++ - Commits as on 11 November 2015
------------------------------------------------------------------------------
----------------------------------------
Python Embedding in C++ implementation has been added and invoked from asfer
main entrypoint with a boolean flag. This at present is not based on
boost::python and directly uses CPython API. Logs for this has been added in
cpp-src/testlogs. With this all AsFer machine learning algorithms can be invoked
from asfer.cpp main() through commandline as: $./asfer <python-script>.

Schematic Diagram:
-----------------
        AsFer C++ -----> C Python Embedding ------> AsFer Python  -------->
Machine Learning algorithms on Spark RDD datasets
        /\
V
         |
|
         --------------------<--------- userspace upcall ---------------VIRGO
kernel analytics and other kernel modules

------------------------------------------------------------------------------
----------------------------------------------
175. (FEATURE - DONE) Config File Support for AsFer C++ and Python Machine
Learning Algorithms - Commits as on 12 November 2015
------------------------------------------------------------------------------
----------------------------------------------
Config File support for asfer has been added. In asfer.cpp main(),
read_asfer_config() is invoked which reads config in asfer.conf and executes the
enabled AsFer algorithms in C++ and Embedded Python. Config key-values are
stored in a map.
```

---
176. (THEORY) Isomorphism of two Document Definition Graphs
---
http://arxiv.org/abs/1006.4458 and
http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings
.pdf describe the Recursive Gloss Overlap Algorithm to construct a Definition
Graph from a text document with WordNet or any other Ontology. If Definition
Graphs extracted from two documents are Isomorphic, then it can be deduced that
there is an intrinsic structural similarity between the two texts ignoring
grammatical variations, though vertices differ in their labelling.
[SubExponentialTIME algorithm for GI - Lazlo Babai -
http://people.cs.uchicago.edu/~laci/update.html]

---
177. Commits as on 13 November 2015

    - Corrections to Convolution Map Neuron computation with per-coordinate
weight added
    - Removed hardcoded convolution stride and pooling width in the class
    - testlogs following input bitmap features have been added with the output
of 5 neurons in final layer:
        - Without Zero inscribed
        - With Zero inscribed in bitmap as 1s
        - With Bigger Zero inscribed in bitmap as 1s
        - With Biggest Zero inscribed in bitmap as 1s
      The variation of final neural outputs can be gleaned as the size of the
pattern increases from none to biggest.
      The threshold has to be set to neural output without patterns. Anything
above it shows a pattern.
    - Added a config variable for invoking cloud perfect forwarding binaries in
KingCobra in asfer.conf
    - config_map updated in asfer.cpp for enableCloudPerfectForwarding config
variable

---
178. (THEORY) Approximate Machine Translation with Recursive Gloss Overlap
Definition Graph

For natural language X (=English) an RGO graph can be constructed (based on
algorithms in http://arxiv.org/abs/1006.4458 and
http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings
.pdf). This definition graph is a subgraph of WordNet-English. Each vertex is a
word from English dictionary. New RGO graph for language Y is constructed from
RGO(X) by mapping each vertex in RGO(X) to corresponding word in language Y.
RGO(X) and RGO(Y) are isomorphic graphs. Sentences in language Y can be
constructed with lamda composition closure on the RGO(Y) graph implemented
already. This is an approximation for machine translation between two natural
languages. I
n the converse direction, text graphs of 2 texts in 2 different natural
languages can be verified for Graph Isomo
rphism. Isomorphic text graphs in two natural languages prima facie indicate the
texts are similarly organized an
d have same meaning.

---
179. (FEATURE - DONE) Web Spider Implementation with Scrapy Framework - Commits
as on 16 November 2015

Initial commits for a Web Spider - Crawl and Scrape - done with Scrapy framework. Presently it crawls Google News for a query and scrapes the search results to output a list of lines or a digest of the news link texts. This output file can be used for Sentiment Analysis with Recursive Gloss Overlap already implemented. Complete Scrapy project hierarchy is being added to the repository with spider in WebSpider.py and
crawled items in items.py

---
180. (FEATURE - DONE) Sentiment Analyzer Implementation for Spidered Texts - Commits as on 16 November 2015

Sentiment Analyzer implementation for Crawled-Scraped Google News Search result texts with testlogs and screenshots of the RGO graph has
been added. This has been made a specialized analyzer for spidered texts different from twitter analyzer.

---
181. Commits as on 17 November 2015

Requirements.txt for package dependencies have been added to asfer-docs/.
WebSpider Crawl-Scraped URLs descriptions are inserted into MySQL table (asfer_webspider) by importing backend/Abstract_DBBackend.py MySQL implementation. Logs for this Scrapy crawl has been added to webspider/testlogs.
WebSpider.py parse() has been updated with MySQL_DBBackend execute_query()s.
__init__.py have been added for directory-as-package imports.

---
182. Commits as on 19 November 2015

- Added more stop words in spidered text and commented text generation in SocialNetworkAnalysis_WebSpider.py
- logs and screenshots for this has been added to testlogs/
- crawl urls in WebSpider.py has been updated
- Added more edges in definition graph by enumerating all the lemma names of a keyword in previous iteration. This makes the graph dense
and probability of a node having more k-core number increases and classification is more accurate.
- logs and screenshots for updated web spidered news texts of two topics "Theoretical Computer Science" and "Chennai" have been added

---
183. Commits as on 20 November 2015

- Updated Spidered text for Sentiment Analysis
- Changed the Sentiment Analysis scoring belief potential propagation algorithm:
  - Two ways of scoring have been added - one is based on DFS tree of k-core subgraph of the larger wordnet subgraph and the other is based on top core numbered vertices of the wordnet subgraph
  - Sentiment thus is computed only for the core of the graph which is more relevant to the crux or class of the document and less pertinent
vertices are ignored.
  - the fraction score is multiplied by a heuristic factor to circumvent floating points

---
184. Commits as on 23 November 2015

- Updated SentimentAnalyzer.py and SocialNetworkAnalysis_Twitter.py scripts with core number and k-core DFS belief potential propagation
similar to SocialNetworkAnalysis_WebSpider.py.
- logs and screenshots for twitter followers graph and tweet RGO graph Sentiment Analysis have been added.
- Example tweet is correctly automatically classified into "Education" class based on top core number of the vertices and top PageRank. Again this is an intriguing instance where Top core numbered vertices coincide with Top PageRank vertices which is probably self-evident from the fact that PageRank is a Markov Model Converging Random Walk and PageRank matrix multiplication is influenced by high weightage vertices (with lot of incoming links)
Sentiment Analysis of the tweet:
- K-Core DFS belief_propagated_posscore: 244.140625
  K-Core DFS belief_propagated_negscore: 1.0
- Core Number belief_propagated_posscore: 419095.158577
  Core Number belief_propagated_negscore: 22888.1835938
- Above Sentiment Analysis rates the tweet with positive tonality.

---
185. Commits as on 25 November 2015

- New WordNetPath.py file has been added to compute the path in a WordNet graph between two words
- WordNetSearchAndVisualizer.py text generation code has been updated for importing and invoking path_between() function in WordNetPath.py
by which set of sentences are created from the RGO WordNet subgraph. This path is obtained from common hypernyms for two word vertices for each of the edges in RGO WordNet subgraph. RGO graph is a WordNet+ graph as it adds a new relation "is in definition of" to the existing WordNet
that enhances WordNet relations. Each edge in RGO graph is made a hyperedge due to this hypernym path finding.
- logs and screenshots have been added for above

The text generated for the hypernym paths in WordNet subgraph in testlogs/ is quite primitive and sentences are connected with " is related to " phrase. More natural-looking sentences can be created with randomly chosen phrase connectives and random sentence lengths.

---
186. Commits as on 4 December 2015

All the Streaming_<>.py Streaming Algorithm implementations have been updated with:
- hashlib ripemd160 hash MD algorithm for hash functions and return hexdigest()
- USBWWAN byte stream data from USBmd print_buffer() logs has been added as a Data Storage and Data Source
- logs for the above have been added to testlogs/
- Streaming Abstract Generator has been updated with USB stream data iterable and parametrized for data source and storage
- Some corrections to the scripts

---
187. Commits as on 7 December 2015

- USB stream file storage name updated in Streaming_AbstractGenerator
- Corrections to CountMinSketch - hashing updated to include rows (now the element frquencies are estimated almost exactly)
- logs for above updated to CountMinSketch

Added Cardinality estimation with LogLog and HyperLogLog counters for USB stream datasets
- HyperLogLog estimation: ~110 elements
- LogLog estimation: ~140 elements

---
188. Commits as on 8 December 2015

- Updated the Streaming LogLogCounter and HyperLogLogCounter scripts to accept StreamData.txt dataset from Abstract Generator and added a Spark MapReducer script for Streaming Data sets for comparison of exact data with Streaming_<algorithm>.py estimations.
- Added logs for the Counters and Spark MapReducer script on the StreamData.txt
- LogLog estimation: ~133
- HyperLogLog estimation: ~106
- Exact cardinality: 104

---
189. (FEATURE - DONE) Commits as on 9 December 2015

- New python implementation for CountMeanMinSketch Streaming Data Frequency Estimation has been added which is an improved version of CountMinSketch that computes median of average of estimator rows in sketch
- Corrections to CountMinSketch hashing algorithms and Sketch width-depth as per the error bounds has been made
- Spark MapReducer prints the number of elements in the stream data set
- Logs for the above have been added to testlogs

---
190. (FEATURE - DONE) Commits as on 11 December 2015

Dependency Injection code commits for MongoDB backend - With this MongoDB is also a storage backend for AsFer algorithms similar to MySQL:
- Abstract_DBBackend.py has been updated for both MySQL and MongoDB injections
- MongoDB configuration and backend connect/query code has been added. Backend is either populated by Robomongo or pymongo reading from the Streaming Abstract Generator iterable framework.
- With this AsFer supports both SQL(MySQL) and NoSQL(file,hive,hbase,cassandra backends in Streaming Abstract Generator).
- log with a simple NoSQL table with StreamingData.txt and USBWWAN data has been added to testlogs/.
- MongoDB configuration has a database(asfer-database) and a collection(asfer-collection).
- MongoDB_DBBackend @provides pymongo.collection.Collection which is @inject-ed to Abstract_DBBackend
- Abstract_DBBackend changes have been reflected in Scrapy Web Spider - backend added as argument in execute_query()
- Abstract_DBBackend.py has a subtle problem:
    - Multiple @inject(s) are not supported in Python Injector
    - only the innermost @inject works and outer @inject throws a __init__ argument errors in webspider/
    - Conditional @inject depending on backend is required but at present

switching the order of @inject(s) circumvents this
- most recent scrapy crawl logs for this have been added to webspider/testlogs

---
191. (THEORY) An update and additions to runtime analysis of Recursive Gloss Overlap Algorithm in TAC 2010
(http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf)

Analysis of runtime in the link above is too cryptic. An updated analysis of steps are given here:
- Nodes at topmost level are keywords in document. For each subsequent recursion step following analysis is per keyword.
- Nodes at level i-1 are computed (base case) : x^(i-1) where x is average size of gloss definition
- Naive pairwise comparison of overlap is done at level i-1 which makes it x^2(i-1)
- Tree isomorphism algorithms can be optionally applied to reduce number of nodes getting recomputed. There are
polytime algorithms for subtree isomorphisms
(www.cs.bgu.ac.il/~dekelts/publications/subtree.pdf)
- Nodes at level i-1 are reduced by Overlap(i-1) : x^(i-1) - Overlap(i-1)
- Maximum number Nodes at level i - are from gloss expansion of those at i-1 : (x^(i-1) - Overlap(i-1))*x
- Thus total time at level i is:
Time_for_pairwise_comparison_to_find_gloss_verlap +
Time_for_removing_isomorphic_nodes
T(i) = sigma([x^(i-1) - Overlap(i-1)]^2 + subtree_isomorphism(i)))
- Above is naively upperbounded to O(x^(2d)) [as subtree isomorphism is polynomial in supertree and subtree vertices and is only an optimization step that can be ignored]. For W keywords in the document time bound is O(W*x^(2d)).
- If number of vertices in RGO multipartite graph (ignoring isomorphism) constructed above is V=O(x^(d)), runtime is O(W*V^2) which is
far less than O(E*V^2) mentioned in TAC 2010 link because number of keywords in toplevel are less than number of edges created during recursion which depend on x and exponentially on d. For example after first recursion completion, number of edges are W*x.
- On a related note runtime for intrinsic merit ranking of the RGO wordnet subgraph can not be equated per-se to ranking from prestige-based search engines as RGO is objective graph-theoretic ranking (does not require massive bot-crawling, indexing and link graph construction) and PageRank is subjective prestige based ranking(depends on crawling, indexing and time needed for link graph construction). Standard publicly available PageRank iteratively computes random walk matrix multiplication for link graphs for billions of nodes on WWW and this runtime has to be apportioned per-node. Time and Space for crawling and indexing have also to be accounted for per-node. Naive bound for PageRank per node is
O(time_for_iterative_matrix_multiplication_till_convergence/number_of_nodes). Matrix multiplication is O(n^omega) where omega~2.3. Thus pagerank bound per node is O(((n^omega)*iterations)/n) assuming serial PageRank computation. Parallel Distributed Versions of PageRank depend on network size and scalable.
- Parallel Recursive Gloss Overlap graph construction on a cloud could reduce the runtime to O(W*V^2/c) where c is size of the cloud.

---
192. (FEATURE - DONE) Commits as on 14 December 2015

- New Interview Algorithm script with NetworkX+Matplotlib rendering and takes as input a randomly crawled HTML webpage(uses beautiful soup to
rip off script and style tags and write only text in the HTML page) has been

added
- Above script also computes the graph theoretic connectivity of the RGO wordnet subgraph based on Menger's theorem - prints number of nodes
required to disconnect the graph or equivalently number of node indpendent paths
- logs and screenshots for the above have been added
- WebSpider.py has been updated to crawl based on a crawling target parameter - Either a streaming website(twitter, streamed news etc.,) or a
HTML webpage

---
193. (FEATURE - DONE) RGO graph complexity measures for intrinsic merit of a text document - Commits as on 15 December 2015

- Interview Algorithm Crawl-Visual script has been updated with a DOT graph file writing which creates a .dot file for the RGO graph constructed
- Also variety of connectivity and graph complexity measures have been added
- Importantly a new Tree Width computing script has been added. NetworkX does not have API for tree width, hence a naive script that iterates
through all subgraphs and finds intersecting subgraphs to connect them and form a junction tree, has been written. This is a costly measure compared to intrinsic merit which depends only on graph edges,vertices and depth of recursive gloss overlap. Tree decomposition of graph is NP-hard.

---
194. Commits as on 16 December 2015

- TreeWidth implementation has been corrected to take as input set of edges of the RGO graph
- TreeWidth for set of subgraphs less than an input parameter size is computed as TreeWidth computation is exponential in graph size
and there are MemoryErrors in python for huge set of all subgraphs. For example even a small graph with 10 nodes gives rise to 1024
possible subgraphs
- Spidered text has been updated to create a small RGO graph
- Logs and screenshots have been added
- Each subgraph is hashed to create a unique string for each subgraph
- TreeWidth is printed by finding maximum set in junction tree

---
195. (THEORY) WordNet, Evocation, Neural networks, Recursive Gloss Overlap and Circuit Complexity
---
WordNet and Evocation WordNet are machine learning models based on findings from psychological experiments. Are WordNet and Neural networks
related? Evocation WordNet which is network of words based on how evocative a word is of another word readily translates to a neural network.
This is because a machine learning abstraction of neuron - perceptron - takes weights, inputs and biases and if the linear program of these breach a threshold, output is activated. Evocative WordNet can be formulated as a neuron with input word as one end of an edge and the output word as the other end - a word evokes another word. Each edge of an Evocation WordNet or even a WordNet is a single input neuron and WordNet is one giant multilayered perceptron neural network. Since Threshold circuits or TC circuit complexity class are theoretical equivalents of a machine learning neuron model (threshold gate outputs 1 if more than k of inputs are 1 and thus an activation function of a neuron), WordNet is hence a gigantic non-uniform TC circuit of Threshold gates. Further TCk is in NC(k+1). This implies that all problems depending on WordNet are inherently parallelizable in theory and Recursive Gloss Overlap algorithm that depends on

WordNet subgraph construction in
http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings
.pdf and updates to it in (191) above are non-uniformly parallelizable.

---
196. (FEATURE - DONE) Mined Rule Search in Astronomical Data - Commits as on 17 December 2015

- New Mined Class Association Rule Search script has been added. This script searches the astronomical data with parsed Maitreya Text client
data with date, time and longitude-latitude queries.
- Where this is useful is after mining the astronomical data with SequenceMining, there is a necessity to search when a particular mined rule occurs. This is an experimental, non-conventional automated weather prediction (but not necessarily unscientific as it requires expertise beyond computer science to establish veracity).
- Logs for this has been added in testlogs/ and chartsummary.rulesearch

---
197. Commits as on 18 December 2015

- Interview Algorithm crawl-visual script changed for treewidth invocation
- Spidered text changed
- Rule Search script corrected to use datetime objects
- Rule Search script corrected to use timedelta for incrementing date and time
- logs and screenshots for above
- Junction Tree for RGO graph - logs and screenshots

---
198. (THEORY) Star Complexity of Graphs - Complement Function circuit and Unsupervised Classification with RGO graph
---
Star complexity of a graph defined in [Stasys Jukna] -
http://www.thi.informatik.uni-frankfurt.de/~jukna/ftp/graph-compl.pdf
 is the minimum number of union and intersection operations of star subgraphs required to create the larger graph which is strikingly relevant to the necessity of quantitative intrinsic merit computation in Recursive Gloss Overlap algorithm
[http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf]. Each star subgraph of the definition graph is an unsupervised automatically found class a text belongs to. Apart from this star complexity of a boolean circuit graph for Complement Function
http://arxiv.org/pdf/1106.4102v1.pdf and in particular for Prime complement special case should have direct relationship conjecturally to pattern in primes because boolean formula for the graph depends on prime bits.

---
199. (FEATURE - DONE) Tornado Webserver, REST API and GUI for NeuronRain - Commits as on 22 December 2015

Commits for NeuronRain WebServer-RESTfulAPI-GUI based on tornado in
python-src/webserver_rest_ui/:
- NeuronRain entrypoint based on tornado that reads a template and implements GET and POST methods
- templates/ contains renderable html templates
- testlogs/ has neuronrain GUI logs

- RESTful API entrypoint <host:33333>/neuronrain

---
200. (FEATURE - DONE) NeuronRain as SaaS and PaaS - for commits above in (199)

RESTful and python tornado based Graphical User Interface entrypoint that reads from various html templates and passes on
incoming concurrent requests to NeuronRain subsystems - AsFer, VIRGO, KingCobra, USBmd and Acadpdrafts - has bee added. Presently implements
simplest possible POST form without too much rendering (might require flask, twisted, jinja2 etc.,) for AsFer algorithms execution.
This exposes a RESTful API for commandline clients like cURL. For example a cURL POST is done to NeuronRain as:
cURL POST: curl -H "Content-Type: text/plain" -X POST -d '{"component":"AsFer","script":"<script_name>","arguments":"<args>"}' http://localhost:33333/neuronrain where REST url is <host:port>/neuronrain.
Otherwise REST clients such as Advanced RESTful Client browser app can be used.
With this NeuronRain is Software-As-A-Service (SaaS) Platform deployable on VIRGO linux kernel cloud, cloud OSes and containers like Docker.
More so, it is Platform-As-A-Service (PaaS) when run on a VIRGO cloud.

---
201. (FEATURE - DONE) Apache Spark MapReduce Parallel Computation of Interview Algorithm Recursive Gloss Overlap Graph Construction
 - Commits as on 24 December 2015

- 2 python files for Spark MapReduce of Recursive Gloss Overlap graph construction has been added to repository
- These two implement Interview Algorithm Recursive Gloss Overlap graph construction and Map-Reduce functions that parallellize
each recursion step computing the gloss tokens from previous recursion level. This is the most important part of the algorithm which
consumes lot of time in serial version along with overlap computation. Apache Spark has been recently gaining importance over its plain Hadoop counterpart and is many times faster than Hadoop in benchmarks.
- In this implementation, Resilient Distributed Dataset is the set of tokens in each recursion level and is parallelly computable in a
huge Spark cluster.
- For example if Spark cluster has 10000 nodes, and each level of recursion produces X number of gloss tokens for graph vertices, time
complexity in Spark is O(X/10000) where as serial version is O(X) with negligible network overhead. Spark has in-memory datasets which
minimizes network latency because of disk access.
- There were lot of implementation issues to make the parallellism. Map and Reduce functions use namedtuples to return data. With more than one
field in namedtuple, there are internal pickling issues in Py4J - Python4Java which PySpark internally invokes to get streamed socket bytes from
Java side of the object wrapped as an iterable. This prevents returning multiple values - for example Synsets - in Map-Reduce functions.
- So only tokens at a level are map-reduced and returned while the prevlevelsynsets (required for adding edges across vertices) are "pickled"
with a properietary asfer_pickle_load() and asfer_pickle_dump() functions that circumvents the python and java vagaries in pickling.
- With proprietary pickling and Map-Reduce functions, Recursive Gloss Overlap graph has been constructed in parallel. Screenshots for this has been added in testlogs.
- Proprietary pickling is done to a text file which is also added to repository. This file is truncated at the end of each recursion.
- Subtlety here is that maximum number of tokens at a recursion level t = number_of_tokens_at_level_(t-1) * maximum_size_of_gloss_per_word(s)

which grows as series = number_of_words*(1 + s + s^2 + s^3 + ... + s^t_max). Size of a document - number of words - can be upperbounded by a constant (d) due to finiteness of average text documents in realworld ignoring special cases of huge webpages with PDF/other.
- If size of the Spark cluster is O(f(d)*s^t_max), each recursion step is of time O(d*s^t_max/f(d)*s^t_max)=O(d/f(d)) and total time for all levels is O(d*t_max/f(d)) which is ranking time per text document. This bound neglects optimization from overlaps and isomorphic nodes removal and is worst case upperbound. For ranking n documents this time bound becomes O(n*d*t_max/f(d)). For constant t_max this bound is O(n*d/f(d)) - and thus linearly scalable.
- Maximum size of gloss per word (s) is also an upper-boundable constant. With constant d and t_max , size of cluster O(f(d)*s^t_max) is constant too independent of number of documents.
- Example: For set of 1000 word documents with f(d)=log(d), max gloss size 5 and recursion depth 2, size of cluster is O(log(1000)*25)~250 nodes with runtime for ranking n documents = O(n*1000*t_max/log(1000)=O(n*100*t_max) which is O(n) for constant t_max.
- Above is just an estimate of approximate speedup achievable in Spark cluster. Ideally runtime in Spark cloud should be on the lines of analysis in (191) - O(n*d*s^2(t_max)/f(d)*s^t_max) = O(n*d*s^t_max/f(d)).
- Thus runtime upperbound in worst case is O(n*d*s^t_max/f(d)) for cluster size O(f(d)*s^t_max).
- If cluster autoscales based on number of documents also, size is a function of n and hence previous size bound is changed to O(g(n)*f(d)*s^t_max) and corresponding time bound = O(n*d*s^2(t_max)/(f(d)*s^t_max*g(n))) = O(n*d*s^t_max/(f(d)*g(n)))

---

202. (THEORY) Recursive Gloss Overlap, Cognitive and PsychoLinguistics and Language Comprehension

---

Recursive Gloss Overlap algorithm constructs a graph from text documents. Presently WordNet is probably the only solution available to get relations across words in a document. But the algorithm does not assume WordNet alone. Any future available algorithms to create relational graphs from texts should be able to take the place of WordNet. Evocation WordNet (http://wordnet.cs.princeton.edu/downloads/evocation.zip) is better than WordNet as it closely resembles a neural network model of a word evocative of the other word and has stronger psychological motivation. There have been efforts to combine FrameNet, VerbNet and WordNet into one graph. Merit of a document is independent of grammar and language in which it is written. For example a text in English and French with grammatical errors delving on the same subject are equivalently treated by this algorithm as language is just a pointer to latent information buried in a document. Process of Language Comprehension is a field of study in Cognitive and Psychological Linguistics. Problem with prestige based subjective rankings is that same document might get varied reviews from multiple sources and consensus with majority voting is required. This is somewhat contrary to commonsense because it assumes majority decision is correct 100% (this is exactly the problem analyzed by P(Good) binomial summation and majority voting circuits elsewhere in this document). In complexity parlance prestige rankings are in BP* classes. Objective rankings are also in BP* classes because of dependency on the framework like WordNet to extract relationships without errors, but less costlier than prestige rankings - major cost saving being lack of dependence on WWW link graph crawling to rank a document.

Intuition for Recursive Gloss Overlap for weighing natural language texts is from computational linguistics, Eye-Movement tracking and Circuit of the Mind [Leslie Valiant]. Human process of comprehending language, recursion as an inherent trait of human faculty and evolution of language from primates to human is described in http://www.sciencemag.org/content/298/5598/1569, www.ncbi.nlm.nih.gov/pubmed/12446899 and http://ling.umd.edu/~colin/courses/honr218l_2007/honr218l_presentation7.ppt by

[Hauser, NoamChomsky and Fitch] with an example part-of-speech recursive tree of a sentence. For example, a human reader's eye usually scans each sentence in a text document word by word left-to-right, concatenating the meanings of the words read so far. Usually such a reader assumes grammatical correctness (any grammatical anamoly raises his eyebrows) and only accrues the keyword meanings and tries to connect them through a meaningful path between words by thinking deep into meaning of each word. This is exactly simulated in Recursive Gloss Overlap graph where gloss overlaps connect the words recursively. Fast readers have reasonably high eye-movement, sometimes randomly. The varied degree of this ability to form an accurate connected visualization of a text probably differentiates people's intellectual wherewithal to draw inferences. From theory of computation perspective, recursive gloss overlap constructs a disambiguated Context Sensitive graph from Natural language text a superset of context free grammars. But natural languages are suspected to be subset of larger classes of context sensitive languages accepted by Linear Bounded Automata. Thus reverse engineering a text from the recursive gloss overlap graph may yield a language larger than natural languages. Information loss due to text-to-graph translation should only be grammatical ideally (e.g Parts of Speech, connectives etc.,) because for comparing two documents for information quality, grammar shouldn't be a factor unless there are fringe cases where missing grammar might change the meaning of a document. Corrections for pre-existing grammatical errors are not responsibilities of this algorithm. This fringe case should be already taken care of by preprocessing ingestion phase that proof-reads the document for primitive grammatical correctness though not extensive and part-of-speech. Recursive Gloss Overlap graph constructed with Semantic Net frameworks like WordNet, Evocation WordNet, VerbNet, FrameNet, ConceptNet, SentiWordNet etc., can be fortified by mingling Part-of-Speech trees for sentences in a document with already constructed graph. This adds grammar information to the text graph.

Psycholinguistics have the notion of event related potentials - when brain reacts excessively to anamolous words in neural impulses. Ideal intrinsic merit rankings should also account for such ERP data to distinguish unusual sentences, but datasets for ERPs are not widely accessible except SentiWordNet. Hence Recursive Gloss Overlap is the closest possible for comparative study of multiple documents that ignores parts-of-speech, subjective assessments and focuses only on intrinsic information content and relatedness.

A thought experiment of intrinsic merit versus prestige ranking:
Performance of an academic personality is measured first by accolades,awards,grades etc., which form the societal opinion - prestige (citations). That is prestige is created from intrinsic merit. But measuring merit from prestige is anachronistic because merit precedes prestige. Ideally prestige and intrinsic merit should coincide when the algorithms are equally error-free. In case of error, prestige and merit are two intersecting worlds where documents without merit might have prestige and vice-versa. Size of the set-difference is measure of error.

References:

202.1 Circuits of the Mind - [Leslie Valiant] - http://dl.acm.org/citation.cfm?id=199266
202.2 Mind Grows Circuits - Lambda calculus and circuit modelling of mind - [Rina Panigrahy, Li Zhang] - http://arxiv.org/pdf/1203.0088v2.pdf
202.3 Psycholinguistics Electrified - EEG and SQUID Event Related Electric Potentials (ERP) peaking for anamolous words - N400 experiment - http://kutaslab.ucsd.edu/people/kutas/pdfs/1994.HP.83.pdf
202.4 Word Associations and Evocations - http://hci.cse.ust.hk/projects/evocation/index.html
202.5 Combining WordNet, VerbNet, FrameNet - http://web.eecs.umich.edu/~mihalcea/papers/shi.cicling05.pdf
202.6 Computational Psycholinguistics - PoS Parsers - http://www.coli.uni-saarland.de/~crocker/courses/comp_psych/comp_psych.html
202.7 Brain Data for Psycholinguistics - http://personality.altervista.org/docs/

14yg-al_brainsent@jlcl.pdf
202.8 ConceptNet 5 - http://conceptnet5.media.mit.edu/
202.9 Sanskrit WordNet - http://www.cfilt.iitb.ac.in/wordnet/webswn/
202.10 IndoWordNet - http://www.cfilt.iitb.ac.in/indowordnet/index.jsp
202.11 Brain Connectivity and Multiclass Hopfield Network - Associative memory - http://www.umiacs.umd.edu/~joseph/Wangetal_Neuroinformatics2015.pdf
202.12 Text Readability Measures, Coherence, Cohesion - http://www.readability.biz/Coherence.html
202.13 MultiWordNet for European Languages - http://multiwordnet.fbk.eu/english/home.php
202.14 Coherence and Text readability indices (Coh-Metrix, FleschKincaid, Brain Overload etc.,) - http://lingured.info/clw2010/downloads/clw2010-talk_09.pdf - Coherence measures how connected the document is and thus closely related to Intrinsic Merit obtained by WordNet subgraph for a text.
202.15 Readability and WordNet - http://www.aclweb.org/anthology/O08-1
202.16. Semantic Networks (Frames, Slots and Facets) and WordNet - http://onlinelibrary.wiley.com/doi/10.1207/s15516709cog2901_3/epdf - describes various graph connectivity measures viz., Clustering coefficient for the probability that neighbours of a node are neighbours themselves , Power-Law distribution of degree of a random node. WordNet is modern knowledge representation framework inspired by Semantic Networks which is a graph of Frames with Slots and Facets for edges amongst Frames.
202.17 Text Network Analysis - Extracting a graph relation from natural language text - http://noduslabs.com/publications/Pathways-Meaning-Text-Network-Analysis.pdf (2011) - Parses keywords from texts and connects the keyword nodes by relation edges with weights - an alternative to WordNet where connections are based on co-occurrence/proximities of words within a certain window. Two words are connected conceptually if they co-occur (similar to Latent Semantic Indexing by SVD).
202.18 Text Network Analysis - Networks from Texts - http://vlado.fmf.uni-lj.si/pub/networks/doc/seminar/lisbon01.pdf
202.19 Six Degrees - Science in Connected Age - [Duncan J.Watts] - Pages 141-143 - Groundbreaking result by [Jon Kleinberg] - https://www.kth.se/social/files/5533a99bf2765470d3d8227d/kleinberg-smallworld.pdf which studies Milgram's Small World Phenomenon by probability of finding paths between two nodes on a lattice of nodes: Probability of an edge between 2 nodes is inversely proportional to the clustering coefficient (r). The delivery time (or) path between 2 nodes is optimum when r=2 and has an inverted bell-curve approximately. When r is less than 2 and greater than 2 delivery time is high i.e finding the path between two nodes is difficult. In the context of judging merit by definition graph of a document, document is "meaningful" if it is "relatively easier" to find a path between two nodes(Resnik,Jiang-Conrath concept similarity and distance measures). This assumes the definition graph is modelled as a random graph. Present RGO definition graph algorithm does not create a random, weighted graph - this requires probabilistic weighted ontology. Statically, Kleinberg's result implies that document definition graphs with r=2 are more "meaningful, readable and visualizable" and can be easily understood where r is proportional to word-word edge distance measures (Resnik,Jiang-Conrath).
202.20 Align,Disambiguate,Walk distance measures - http://wwwusers.di.uniroma1.it/~navigli/pubs/ACL_2013_Pilehvar_Jurgens_Navigli.pdf
202.21 Coh Metrix - Analysis of Text on Cohesion and Language - https://www.ncbi.nlm.nih.gov/pubmed/15354684 - quantifies text for meaningfulness, relatedness and readability
202.22 Homophily in Networks - [Duncan J.watts] - Pages 152-153 - https://arxiv.org/pdf/cond-mat/0205383.pdf - This result implies in a social network where each vertex is a tuple of more than one dimension, and similarity between any two tuples s and j is defined as distance to an adjacent node j of s which is close to t, and any message from sender s reaches receiver t in minimum length of intermediaries - optimal when number of dimensions are 2 or 3 - Kleinberg condition is special case of this.
202.23 Six Degrees - [Duncan J.Watts] - Triadic closure of Anatole Rapoport - Clustering Coefficient - Page 58

---
203. Apache Spark MapReduce Parallel Computation of Interview Algorithm Recursive Gloss Overlap Graph Construction
- Commits as on 25 December 2015

- Added more parallelism to python-src/InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.py. New map-reduce functions
for computing parents (backedges in the recursion) have been added - mapFunction_Parents(), reduceFunction_Parents(), SparkMapReduce_Parents() in
python-src/InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit_SparkMapReducer.py
- These functions take the previous level tokens as inputs instead if synsets

---
204. Commits as on 28 December 2015

- Updated MapReduce functions in Spark Recursive Gloss Overlap implementation

---
205. Commits as on 29 December 2015

- Updated python-src/InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.py for choosing parallel spark mapreduce or
serial parents backedges computation based on a boolean function.
- This is because Recursive tokens mapreduce computation (Spark_MapReduce) is faster than serial version.
But serial parents computation is faster than parallel parents computation ironically(Spark_MapReduce_Parents).
This happens on single node Spark cluster. So, parents computation has been made configurable(serial or parallel)
- Logs and Screenshots for various texts experimented have been added to testlogs/
- python-src/InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit_SparkMapReducer.py mapreduce uses best_matching_synset() for
disambiguation now which was commented earlier.
- MapFunction_Parents() has a pickling problem in taking a tuple of synset objects as input arg due to which synsets have to be recomputed that
 causes the slowdown mentioned above compared to serial parents() version.
MapFunction_Parents() has been rewritten to take previous level gloss tokens in lieu of synsets to circumvent pickling issues.
- Slowdown could be resolved on a huge cluster.
- Thus this is a mix of serial+parallel implementation.
- Logs,DOT file and Screenshots for mapreduced parents() computation

---
206. Commits as on 30 December 2015

Some optimizations and redundant code elimination in python-src/InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.py

---
207. Commits as on 31 December 2015

- Spark Recursive Gloss Overlap Intrinsic Merit code has been optimized and some minutiae bugs have been resolved.
- Map function for parents computation in Spark cluster has been updated to act upon only the previous recursion level glosses
by proprietary pickling of the keyword into a file storage and loading it, and not as a function argument
- Problems with previous level gloss tokens were almost similar to MapReduce functions of the recursion
- New pickling file for parents computation in Spark has been added
- logs and screenshots have been added to testlogs
- With this, Spark Intrinsic Merit computation is highly parallellized apt for large clouds
- Also a parents_tokens() function that uses gloss tokens instead of synsets has been added
- New pickling dump() and load() functions have been added for keyword storage
- pickling is synchronized with python threading lock acquire() and release(). Shouldn't be necessary because of
Concurrent Read Exclusive Write (CREW) of keyword, but for safer side to prevent Spark-internal races.

---
208. Commits as on 1,2,3,4,5,6,7 January 2016

- Added sections 53.14, 53.15 for HLMN and PARITY 3-SAT in 53
- updated spidered text
- best_matching_synset() enabled in mapreduce backedges computation which accurately disambiguates the graph. Spark single node cluster is
significantly slower than serial version of parents() computation with only python calls probably due to costly Python4Java
back-and-forth stream socket reads.
- logs, DOT file and screenshots for single node Spark cluster have been added to testlogs/

---
209. Commits as on 8 January 2016

- In InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit_SparkMapReducer.py increased local threads to 2 (for dual core cpu(s)) for
all SparkContexts instantiated
- Added screenshot and logs for 2 local threads SparkContext mapreduce
- Reference: Berkeley EdX Spark OpenCourse - https://courses.edx.org/c4x/BerkeleyX/CS100.1x/asset/Week2Lec4.pdf

---
210. Commits as on 10 January 2016

NeuronRain Enterprise (GitHub) version 2016.1.10 released.

---
211. Commits as on 11 January 2016

Added section 53.16 for an apparent contradiction between polysize and
superpolynomial size among P/poly, Percolation and
special case of HLMN theorem.

---
212. (FEATURE - DONE) Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) - Commits as on 12 January 2016

Python rpy2 wrapper implementation for :
        - Principal Component Analysis(PCA)
        - Singular Value Decomposition(SVD)
which invoke R PCA and SVD functions and plot into 2 separate pdf files has been added to repository.
Logs for an example have been added to testlogs/

---
213. (FEATURE - DONE) Kullback-Leibler Divergence - Commits as on 17 January 2016

Kullback-Leibler Divergence implementation:

Approximate distance between 2 probability distributions with logs
in terms of weighted average distance represented as bits

---
214. (FEATURE - DONE) Basic Statistics - Norms and Median - Commits as on 19 January 2016

Implementation for basic statistics - L1, L2 norms, median etc.,

---
215. Commits as on 20 January 2016

- Updated AsFer Design Document for Psycholinguistics of Reading a Text document and Recursive Gloss Overlap
- Added Standard Deviation and Chi-Squared Test R functions to python-src/Norms_and_Basic_Statistics.py

---
216. (THEORY) Recursive Lambda Function Growth Algorithm - Psycholinguistic Functional Programming simulation of Human Reader Eye Movement Tracking and its special setting application in Recursive Gloss Overlap
---
Example sentence:
California Gas Leak Exposes Growing Natural Gas Risks.

A left-right scan of the human reading groups the sentence into set of phrases and connectives and grows top-down gloss overlap graphs:
p1 - California Gas Leak
p2 - Exposes
p3 - Growing Natural Gas Risks

For each phrase left-right, meaning is created by recursive gloss overlap disambiguated graph construction top-down (graph is constructed left-right, top-

down):
p1 - graph g1
p2 - graph g2
p3 - graph g3

Prefix graph construction and functional programming:
Above sentence has three prefix phrases and graphs- p1, p1-p2, p1-p2-p3 in order (and g1, g1-g2, g1-g2-g3). As reader scans the sentence, meaning is built over time period by lambda function composition. Function f1 is applied to g1,g2 - f1(g1,g2) - f1(g1(California Gas Leak), g2(Exposes)) - which returns a new graph for prefix p1-p2. Function f2 is applied to f1(g1,g2) and g3 - f2(f1(g1,g2),g3(Growing Natural Gas Risks)) - which returns a new graph for sentence p1-p2-p3. This is recursively continued and above example can be generalized to arbitrarily long sentences.

This formulation does not depend on just semantic graphs - graphs g1,g2 and g3 could be functional programming subroutines too, which makes f2(f1(g1,g2),g3) as one complete function composition tree. In previous example, for each phrase a function is defined and finally they are composed:
p1 - function g1 - california_gas_leak()
p2 - function g2 - exposes()
p3 - function g3 - growing_natural_gas_risks()
f3 = f2(f1(g1,g2),g3)

How functions f1,f2,g1,g2,g3 are internally implemented is subjective. These functions are dynamically created and evaluated on the fly. Grouping sentences into phrases and connectives has to be a preprocessing step that uses PoS NER tagger - even a naive parsing for language connectives should suffice. Return values of these functions are functions themselves - this is standard "First Class Object" concept in higher order lambda calculus - https://en.wikipedia.org/wiki/First-class_citizen . These functions themselves might invoke smaller subroutines internally which build meaning and return to previous level. Recursive Gloss Overlap implements a special case of this where f1,f2 are gloss overlap functions and g1,g2,g3 are wordnet subgraphs. Its Spark implementation does a "Parallel-Read, Top-Down Graph construction" as against left-right read. In a more generic alternative, these functions could be deep learning neural networks, LISP subroutines etc., that compute "meaning" of the corresponding phrase. If the reading is not restricted to left-right, the composition tree should ideally look like a balanced binary tree per sentence. This is a kind of "Mind-Grows-Functions" formalism similar to the Mind-Grows-Circuit paradigm in [Rina Panigrahy, Li Zhang] - http://arxiv.org/pdf/1203.0088v2.pdf. It is questionable as to what does "Meaning" mean - is it a graph, neural electric impulse in brain etc., and the question itself is self-referencing godel sentence: What is meant by meaning? - which may not have answers in an axiomatic system because word "meaning" has to be defined by something more atomic than "meaning" itself.

As an ideal human sense of "meaning" stored in brain is vague and requires a "Consciousness and Freewill" theory, only a computation theoretic definition of meaning (i.e a circuit graph) is axiomatically assumed. Thus above lambda function composition is theoretically equivalent to boolean function composition (hence circuit composition). With this the above recursive lambda function growth algorithm bridges two apparently disconnected worlds - complexity theory and machine learning practice - and hence a computational learning theory algorithm only difference being it learns a generic higher-order lambda function (special case is recursive gloss overlap) from text alphabet strings rather than a boolean function from binary strings over {0,1} alphabet. Essentially every lambda function should have a circuit by Church-Turing thesis and its variants - Church-Turing thesis which states that any human computable function is turing-computable is still an axiom without proof on which computer science has been founded. Because of this equivalence of lambda functions and Turing machines, the lambda function learning algorithm essentially covers all possible complexity classes which include Recursively Enumerable languages. Theoretically, concepts of noise sensitivity, influence which measure the

probability of output change for flipped input should apply to this lambda function learning algorithm for text documents - For example typo or error in grammar that affects the meaning.

Learning theory perspective of the previous - From Linial-Mansour-Nisan theorem , class of boolean functions of n variables with depth-d can be learnt in O(n^O(logn^d)) with 1/poly(n) error. Consequentially, Above Lambda Function Growth for depth d can be encoded as a circuit (a TC circuit in wordnet special case) that has n inputs where n are number of keywords in text (each word is boolean encoded to a binary string of constant length) and learnt in O(n^O(logn^d)) with 1/poly(n) error. This learning theory bound has striking resemblance to parallel Recursive Gloss Overlap bound of O(n1*d1*s^t_max/(g(n1)*f(d1))) for n1 documents with d1 keywords each on a cluster. Equating to LMN bound gives rise to interesting implication that average number of gloss per keyword = log(number_of_keywords) which is too high gloss size per word(other variables equated as n=d1,t_max=d). Therefore recursive gloss overlap could be faster than LMN learning algorithm.

References:

216.1 Eye Movement Tracking - https://en.wikipedia.org/wiki/Psycholinguistics#Eye-movements

216.2 Eye Movements in Text Comprehension - Fixations, Saccades, Regressions - http://www.jove.com/video/50780/using-eye-movements-to-evaluate-cognitive-processes-involved-text

216.3 Symbol Grounding Problem and learning from dictionary definitions - http://aclweb.org/anthology//W/W08/W08-2003.pdf - "...In the path from a word, to the definition of that word, to the definition of the words in the definition of that word, and so on, through what sort of a structure are we navigating (Ravasz & Barabasi, 2003; Steyvers & Tenenbaum, 2005)? Meaning is compositional: ..." - Symbol grounding is the problems of infinite regress while affixing meaning to words. As defined by Frege, word is a referent and meaning is its definition and the two are different. This paper lays theoretical foundations for Graphical Ontologies like WordNet etc., Process of meaning grasping is defined in terms of finding a Grounding set of vertices of an ontology which is equal to a Feedback Vertex Set of the graph. Feedback Vertex Set is subset of vertices of an ontology graph removal of which causes the directed graph to be cycle-less. This is an NP-complete problem. Definition graph construction in Recursive Gloss Overlap creates a subgraph of an ontology projected onto the text document.

216.4 TextGraphs - graph representation of texts - https://sites.google.com/site/textgraphs2017/

216.5 Symbol Grounding Problem - http://users.ecs.soton.ac.uk/harnad/Papers/Harnad/harnad90.sgproblem.html - " ... (1) Suppose the name "horse" is grounded by iconic and categorical representations, learned from experience, that reliably discriminate and identify horses on the basis of their sensory projections. (2) Suppose "stripes" is similarly grounded. Now consider that the following category can be constituted out of these elementary categories by a symbolic description of category membership alone: (3) "Zebra" = "horse" & "stripes"[17] ... Once one has the grounded set of elementary symbols provided by a taxonomy of names (and the iconic and categorical representations that give content to the names and allow them to pick out the objects they identify), the rest of the symbol strings of a natural language can be generated by symbol composition alone, [18] ..."

216.6 Understanding Natural Language - [Terry Winograd] - https://dspace.mit.edu/handle/1721.1/7095#files-area - Chapter 3 - Inference - Meaning of sentences are represented as relations between objects. Recursive Lambda Function Growth described previously has a lambda function for each relation. Relations/Lambda Functions are mostly verbs/adverbs/adjectives and arguments of lambda functions are objects/nouns.

216.7 Grounded Cognition - [Barsalou] - http://matt.colorado.edu/teaching/highcog/readings/b8.pdf - How Language is Grounded - "...Phrasal structures embed recursively..." - Neuroimageing

evidences for Grounded cognition

---
217. Commits as on 21 January 2016

- Corrected Chi-squared test input args
- logs added to testlogs/

---
218. Commits as on 27 January 2016

Updated Sections 14 and 216.

---
219. Commits as on 29 January 2016

- Uncommented both commandlines in cpp-src/asferpythonembedding.sh
---
220. (FEATURE - DONE) Python-C++-VIRGOKernel and Python-C-VIRGOKernel boost::python and cpython implementations:

- It is a known idiom that Linux Kernel and C++ are not compatible.
- In this commit an important feature to invoke VIRGO Linux Kernel from userspace python libraries via two alternatives have been added.
- In one alternative, C++ boost::python extensions have been added to encapsulate access to VIRGO memory system calls - virgo_malloc(), virgo_set(), virgo_get(), virgo_free(). Initial testing reveals that C++ and Kernel are not too incompatible and all the VIRGO memory system calls work well though initially there were some errors because of config issues.
- In the other alternative, C Python extensions have been added that replicate boost::python extensions above in C - C Python with Linux kernel works exceedingly well compared to boost::python.
- This functionality is required when there is a need to set kernel analytics configuration variables learnt by AsFer Machine Learning Code dynamically without re-reading /etc/virgo_kernel_analytics.conf.
- This completes a major integration step of NeuronRain suite - request travel roundtrip to-and-fro top level machine-learning C++/python code and rock-bottom C linux kernel - bull tamed ;-).
- This kind of python access to device drivers is available for Graphics Drivers already on linux (GPIO - for accessing device states)
- logs for both C++ and C paths have been added in cpp_boost_python_extensions/ and cpython_extensions.
- top level python scripts to access VIRGO kernel system calls have been added in both directories:
        CPython - python cpython_extensions/asferpythonextensions.py
        C++ Boost::Python - python cpp_boost_python_extensions/asferpythonextensions.py
- .so, .o files with build commandlines(asferpythonextensions.build.out) for "python setup.py build" have been added in build lib and temp directories.
- main implementations for C++ and C are in cpp_boost_python_extensions/asferpythonextensions.cpp and cpython_extensions/asferpythonextensions.c

- Schematic Diagram:

```
        AsFer Python -----> Boost::Python C++ Extension ------> VIRGO memory
system calls --------> VIRGO Linux Kernel Memory Drivers
        /\
V
         |
|
         ---------------------------------------------
<---------------------------------------------
        AsFer Python -----> CPython Extensions ------> VIRGO memory system calls
--------> VIRGO Linux Kernel Memory Drivers
         /\
V
         |
|
         ---------------------------------------------
<---------------------------------------------
```

---

221. Commits as on 2 February 2016

- Uncommented PyArg_ParseTuple() to read in the key-value passed from Python layer
- Unified key-value in Python layer and within CPython code with : delimiter
- added Py_BuildValue() to return vuid - VIRGO Unique ID - to python and commented virgo_free() so that a parallel code
can connect to kmem cache and do a virgo_get on this vuid - this is a precise scenario where Read-Copy-Update fits in so that
multiple versions can co-exist at a time

---

222. (THEORY) Recursive Lambda Function Growth Algorithm - 216 elaborated with examples

---

Boolean function learning and PAC learning are special cases of this Lambda function learning algorithm because any
intermediate lambda function can be a boolean function too.

Algorithms steps for English Natural Language Procssing - Parallel read:

- Approximate midpoint of the sentence is known apriori
- Language specific connectives are known apriori (e.g is, was, were, when, who etc.,)
- Sentence is recursively split into phrases and connectives and converted into lambda functions with balanced number of nodes in left and right subtrees
- Root of each subtree is unique name of the function

Example Sentence1:

PH is infinite relative to random oracle.

Above sentence is translated into a lambda function composition tree as:
relative(is(PH, infinite), to(random,oracle))) which is a tree of depth 3

In this example every word and grammatical connective is a lambda function that takes some arguments and returns a partial "purport" or "meaning" of that segment of sentence. These partial computations are aggregated bottom-up and culminate in root.

Example Sentence2:

Farming and Farmers are critical to success of a country like India.

Above sentence is translated into a lambda function composition tree as:
Critical(are(and(Farming,Farmers)), a(success(to,of), like(country, India)))

This composition is slightly different from Part-of-Speech trees and inorder traversal of the tree yields original sentence.
As functions are evaluated over time period, at any time point there is a partially evaluated composition prefix tree which operates on
rest of the sentence to be translated yet - these partial trees have placeholder templates that can be filledup by a future lambda function
from rest of the sentence.

Each of these lambda functions can be any of the following but not limited to:
- Boolean functions
- Generic mathematical functions
- Neural networks (which include all Semantic graphs like WordNet etc.,)
- Belief propagated potentials assigned by a dictionary of English language and computed bottom-up
- Experimental theory of Cognitive Psycholinguistics - Visuals of corresponding words - this is the closest simulation of process of cognition and comprehension because when a sentence is read by a human left-right, visuals corresponding to each word based on previous experience are evoked and collated in brain to create a "movie" meaning of the sentence. For example, following sentence:

Mobile phones operate through towers in each cellular area that transmit signals.

with its per-word lambda function composition tree :

in(operate(mobile phones, through(tower)), that(each(cellular area), transmit(signals)))

evokes from left-to-right visuals of Mobile phones, towers, a bounded area in quick succession based on user's personal "experience" which are merged to form a visual motion-pictured meaning of the sentence. Here "experience" is defined as the accumulated past stored in brain which differs from one person to the other. This evocation model based on an infinite hypergraph of stacked thoughts as vertices and edges has been described earlier in sections 35-49 and 54-59. This hypergraph of thoughts grows over time forming "experiences". For some words visual storage might be missing, blurred or may not exist at all. An example for this non-visualizable entity in above sentence is "transmit" and "signal" which is not a tangible. Thus the lambda function composition of these visuals are superimposed collations of individual lambda functions that return visuals specific to a word, to create a sum total animated version. Important to note is that these lambda functions are specific to each reader which depend on pre-built "experience" hypergraph of thoughts which differs for each reader. This explains the phenomenon where the process of learning and grasping varies among people. Hypergraph model also explains why recent events belonging to a particular category are evoked more easily than those in distant past - because nodes in stack can have associated potential which fades from top to down and evocation can be modelled as a hidden markov model process on a stack node in the thought hypergraph top-down as the markov chain. Rephasing, thoughts stored as multiplanar hypergraphs with stack vertices act as a context to reader-specific lambda functions. If stack nodes of thought hypergraph are not markov models - node at depth t need not imply node at depth t-1 - topmost node is the context disambiguating visual returned by the lambda function for that word. This unifies storage and computation and hence a plausible cerebral computational model - striking parallel in non-human turing computation is the virtual address space or tape for each process serving as context.

The language of Sanskrit with Panini's Grammar fits the above lambda calculus framework well because in Sanskrit case endings and sentence meaning are independent of word orderings. For example following sentence in Sanskrit:

Sambhala Grama Mukhyasya VishnuYashas Mahatmana: ... (Chieftain of Sambhala Village, Vishnuyashas the great ...)

can be rearranged and shuffled without altering the meaning, a precise requisite for lambda function composition tree representation of a sentence, which is tantamount to re-ordering of parameters of a lambda function.

As an alternative to WordNet:

Each lambda function carries dictionary meaning of corresponding word with placeholders for arguments.
For example, for word "Critical" corresponding lambda function is defined as:
Critical(x1,x2)
If dictionary meanings of critical are defined with placeholders:
1) Disapproval of something -
2) - is Important to something -
there is a need for disambiguation and 2) has to be chosen based either on number of arguments or disambiguation using lesk algorithm or more sophisticated one. If Second meaning fits context then, lambda function Critical(x1,x2) returns:
x1 is important to something x2
Thus WordNet can be replaced by something more straightforward. These steps can be recursed top-down to expand the meaning - in above example "important" might have to be lookedup in dictionary. This simulates basic human process of language learning and comprehension.

This algorithm accepts natural languages that are between Context Free Languages and Context Sensitive Languages.

References:
222.1 Charles Wikner - Introductory Sanskrit - http://sanskritdocuments.org/learning_tutorial_wikner/
222.2 Michael Coulson - Teach yourself - Sanskrit - http://www.amazon.com/Complete-Sanskrit-Teach-Yourself-Language/dp/0071752668)

---
223. Commits as on 4 February 2016

- Updated AsFer Design Document with more references for Discrete Hyperbolic Factorization in NC - PRAM-NC equivalence
---
(FEATURE - DONE) C++ - Input Dataset Files nomenclature change for NeuronRain Enterprise
---
- 3 new input files cpp-src/asfer.enterprise.encstr, cpp-src/asfer.enterprise.encstr.clustered, cpp-src/asfer.enterprise.encstr.kNN have been added which contain set of generic binary strings for KMeans, KNN clustering and Longest Common Subsequence of a clustered set .
- Code changes for above new input files have been done in asfer.cpp and new class(asferencodestr.cpp and asferencodestr.h) has been added
for generic string dataset processing
- Changes for generic string datasets have been done for kNN and KMeans clustering
- logs for clustering and longest common subsequence have been added
---
(FEATURE - DONE) Python - Input Dataset Files nomenclature change for NeuronRain Enterprise
---

- 1 new input file python-src/asfer.enterprise.encstr.seqmining has been added for Sequence Mining of generic string dataset -
presently contains set of generic binary strings .
- Code changes for above new input files have been done in SequenceMining.py
- logs for SequenceMining of binary strings with maximum subsequence length of 15 have been added

---
224. Commits as on 4 February 2016

(FEATURE - DONE) Crucial commits for Performance improvements and Cython build of Spark Interview algorithm implementation
---
- New setup.py has been added to do a Cythonized build of PySpark Interview algorithm MapReduce script
- Commandline for cython build: python setup.py build_ext --inplace
- This compiles python spark mapreduce script into a .c file that does CPython bindings and creates .o and .so files
- FreqDist has been commented which was slowing down the bootstrap
- MapReduce function has been updated for NULL object checks
- Thanks to Cython, Performance of Spark MapReduce has performance shootup by factor of almost 100x - Spark GUI job execution time
shown for this interview execution is ~10 seconds excluding GUI graph rendering which requires few minutes. This was earlier taking few hours.
- With Cython ,essentially Spark-Python becomes as fast as C implementation.
- DOT file, Output log, Spark logs and Screenshot has been added
- This completes important benchmark and performance improvement for Recursive Gloss Overlap Graph construction on local Spark single node
cluster.
- Above Cythonizes just a small percentage of Python-Spark code. If complete python execution path is Cythonized and JVM GC and Heap tunables for Spark are configured, this could increase throughput significantly further. Also Spark's shuffling parameters tuning could reduce the communication cost.

---
225. Commits as on 5 February 2016

Further optimizations to Spark-Cython Interview algorithm implementation:

- webspidered text for RGO graph construction updated
- PySpark-Cython build commandlines added as a text file log
- Commented unused python functions - shingling and jaccard coefficient
- threading.Lock() Spark thread pickling synchronizer variable made global in MapReducer
- renamed yesterday's logs and screenshot to correct datetime 4 February 2016
- Added a new special graph node ["None"] for special cases when no parents are found in backedges computation in MapReducer. This makes a
default root in the RGO graph as shown in matplotlib networkx graph plot
- Spark logs show time per RDD job - an example MapReduce job takes 335 milliseconds
- Tried Cythonizing InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.py but there is a serious cython compiler error in
Matplotlib-PyQt. Hence those changes have been backed out.
- DOT file and Cython C file have been updated

---
226. Commits as on 8 February 2016

- PySpark-Cythonized Interview Algorithm implementation generated graph has been pruned to remove edges invoving "None" vertices which were
added during Spark MapReduce based on a config variable.
- Logs and Screenshots for the above have been added to testlogs/
- Some benchmark results parsed from the Spark logs with commandline - "for i in `grep Finished
InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.2localthreads.CythonOptimized.8February2016.out |grep TaskSetManager| awk '{print $14}'`; do sum=$(expr $sum + $i); done":
      - Total time in milliseconds - 99201
      - Number of jobs - 113
      - Average time per mapreduce job in milliseconds - 877
Thus a merit computation of sample document with few lines (33 keywords) requires ~100 seconds in local Spark cluster. This excludes NetworkX and other unoptimized code overhead.

---
227. (THEORY) Graph Edit Distance in Interview Algorithm, PySpark-Cython Interview Algorithm benchmarks and Merit in Recursive Lambda Function Growth Algorithm
---
Interview Algorithm to assess merit, throughout this document, refers to 3 algorithms - Citation graph maxflow(which includes standard radius,eccentricity and diameter measures of the graph), Recursive Lambda Function growth and Graph edit distance (or Q&A) based interview between a pair of documents. Recent result about impossibility of better algorithms for edit distance (Backurs-Indyk - http://arxiv.org/abs/1412.0348 - edit distance cannot be computed in subquadratic time unless Strong Exponential Time Hypothesis is false) finds its applications in Graph edit distance for Interview Algorithm Questions&Answering where one document's Recursive Gloss Overlap graph is compared for distance from another graph's RGO WordNet subgraph (http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf). Graph edit distance is a special case of edit distance when graph edges are encoded as strings. A better algorithm to find edit distance between 2 documents' wordnet subgraphs ,therefore implies SETH is false.

Previous benchmarks for PySpark-Cython parallel interview algorithm implementation could scale significantly on a cluster with ~100 Spark nodes with parallel runtime of ~800 milliseconds. If all Java Spark code is JVM tuned and python modules are Cythonized fully, even this could be bettered bringing it to order of few milliseconds.

Computing Intrinsic Merit of a text with Recursive Lambda Function growth could involve multitude of complexity notions:
      - Depth of Composition tree for a text
      - Size of resultant lambda function composition e.g size of WordNet subgraphs, neural nets, complexity of context disambiguation with thought hypergraph psychological model enunciated in 222.
      - In essence, this reduces to a lowerbound of lambda function composition - boolean circuit lowerbounds are special cases of lambda function composition lowerbounds. Since thought hypergraph disambiguator is reader dependent, each reader might get different merit value for same document which explains the real-life phenomenon of "subjectivity". To put it simply, Objectivity evolves into Subjectivity with "experience and thought" context. Evaluating merit based on thought context is non-trivial because disambiguating a word visually as mentioned in 222 requires traversal of complete depth of thought hypergraph - it is like billions of stacks interconnected across making it tremendously multiplanar - a thought versioning system. This explains another realword

phenomenon of experiential learning: An experienced person "views" a text differently from a relatively less experienced person.

---
228. (THEORY) Thought Versioning - ThoughtNet and EventNet - Psychophilosophical Digression - Might have
some parallels in Philosophical Logic - related to points (70-79)

(*) This postulates an experimental model for a "Logical Brain". It makes no assumptions about neural synapses, firing etc., though a graph can be mapped to a neural network with some more reductions. The computation side is taken care of by Recursive Lambda Function Growth (which might invoke neural learning etc.,) while the storage is simulated with a giant ThoughtNet - together this is a turing-computable model for natural language processing which is subjective to each individual. This model makes an assumption of pre-existing consciousness or an equivalent to account for emotions and sentiments.

(*) ThoughtNet is built over time from inception of life - books a person reads, events in life, interactions etc., - are lambda-evaluated and stored. This is were the computation and storage distinction seems to vanish - It is not a machine and tape but rather a machinetape (analogy: space and time coalescing to spacetime). Evaluated lambda functions themselves are stored in ThoughtNet not just thoughts data - storage contains computation and computation looksup storage.

(*) Thought Versioning System - tentatively named ThoughtNet - mentioned in 227 is a generalization of Evocation WordNet as a multiplanar graph with stacked thoughts as vertices. Each stack corresponds to a class an experience or thought is binned - visually or non-visually.

(*) An approximate parallel to ThoughtNet versioning is the Overlay FileSystems in Unixen. Overlay filesystems store data one over the other in layers which play "obscurantist" and prevent the bottom truth from being out. This follows a Copy-up-on-write (CUOW) - a data in bottom layer is copied up and modified and stays overlaid. File access systems calls "see" only "topmost" layer. In ThoughtNet this roughly translates to Thought edges of particular class-stack node placed one over the other and grown over period of time. ThoughtNet thus presents an alternative to Overlay storage where a layer with maximum argmax() potential can be chosen.

(*) EventNet when unified with ThoughtNet makes a cosmic storage repository, all pervasive. Thoughts are stored with accompanied potentials - which could be electric potentials - some stronger thoughts might have heavy potentials compared to insignificant thoughts. How this potentials arise in first place is open to question. Rather than numeric and sign representation of sentiments, this ThoughtNet model proposes "coloring" of the emotions - akin to edge coloring of graphs. ThoughtNet is a giant pulp of accrued knowledge in the form of observed events stored as hyperedges. Here a distinction has to be made between Wisdom and Knowledge - Knowledge measures "To know" while Wisdom measures "To know how to know". Common knowledge in Logic ("We know that you know that We know ... ad infinitum") thus still does not explain wisdom (Does wisdom gain from knowledge and vicecersa?). ThoughtNet is only a knowledge representation and Wisdom remains open to interpretation - probably recursive lambda function growth fills-in this gap - it learns how to learn.

(*) For example, an event with pleasant happening might be stored with high potential of positive polarity as against a sad event with a negative polarity potential. Repetitive events with high potentials reinforce - potentials get added up. This extends SentiWordNet's concept of postivity, negativity and objectivity scoring for words. It is not a necessity that Hidden Markov Models are required for finding a right evocative (mentioned in 56). Trivial argmax(all potentials per stack class) is sufficient to find the most relevant evocative

hyperedge. This explains the phenomenon where an extremely happy or sad memory of a class lingers for long duration despite being old whereas other memories of same class fade away though relatively recent. Emotions affecting potentials of thought hyperedges is explained further below.

(*) EventNet is a cosmic causality infinite graph with partaker nodes whereas ThoughtNet is per partaker node. This evocation model is experimental only because a void exists in this simulation which can be filledup only by consciousness and egoself. Example: A sentence "Something does not exist" implies it existed in past or might exist in future and thus self-refuting. Otherwise mention of "something" should have never happened.

(*) EventNet is a parallelly created Causality graph - event vertices occur in parallel across universe and causation edges are established in parallel. This is like a monte-carlo Randomized NC circuit that grows a random graph in parallel.

(*) Previous evocation model with HMM or maximum potential need not be error-free - it can be derailed since human understanding at times can be derailed - because humans are inherently vulnerable to misunderstand ambiguous texts. Hence above evocation might return a set of edges than a unique edge which has to be further resolved by additional information for disambiguation. For example, rather than individual words a sentence as a block might require disambiguation: In the Cognitive Psycholinguistics model which stores thoughts as visuals mentioned in 222, there is a chance of derailment because of evocation returning multiple edges when human emotions are involved (Love, Anger, Sarcasm, Pathos etc.,). ThoughNet based disambiguation simulates subjective human reasoning more qualitatively specific to each individual and accurately than usual statistical sentiment inferences which are mere numbers and do not consider individual specific perception of a text. This also explains how an "image"/"perception" is engineered over a time period which depends on thoughts stored.

Thought experiment for this defective disambiguation is as below

---

(*) Example Sentence 1: "Love at first sight is the best". Considering two persons reading this sentence with drastically different ThoughtNet contexts - one person has multiple good followed by bad experiences stored as ThoughtNet hypergraph edges with stronger postive and negative potentials and the argmax() disambiguated evocative edges returned could be therefore both quite positive and negative inducing him to either like or abhor this sentence based on whichever wins the mind conflict (i.e whichever potential positive or negative is overpowering), while the other person has a pure romantic history with good experiences alone disambiguated evocation in which case is uniquely positive. This is real-world simulation of "confused" thought process when emotions rule. In other words emotions determine the polarity and extent of potentials tagged to each thought hyperedge and more the emotional conflict, larger the chance of difficulty in unique disambiguation and derailment. It is not a problem with ThoughtNet model,but it is fundamental humane problem of emotions which spill over to ThoughtNet when accounted for. If human reasoning is ambiguous so is this model. This danger is even more pronounced when lambda compositions of visuals are involved than just sentences. As an example, a test human subject is repeatedly shown visuals related to love reinforcing the potentials of hyperedges belonging to a class of "Love" and subject is bootstrapped for future realtime love. This artificially creates an illusory backdrop of "trained data" ThoughtNet context - a kind of "Prison Experiment for Behavioural Modification Under Duress" - the distinction between reality and imagination gets blurred - e.g Stanford Prison Experiment. Machine learning programs cannot quantify emotions through statistics (in the absence of data about how emotions are stored in brain, does emotion originate from consciousness etc.,) and hence creating a visualized meaning through recursive lambda composition of natural language texts have a void to fill. If emotions are Turing-computable, the lambda function composition makes sense. ThoughtNet model with potentials, thus, simulates human grasp of a text with provisions for emotions. No existing machine learning model at present is known to reckon emotions and thought

contexts.

(*) Example Sentence 2: "You are too good for any contest". This sentence has sarcastic tone for a human reader. But can sarcasm be quantified and learned? Usual statistical machine learning algorithms doing sentiment analysis of this sentence might just determine positivity, negativity or objectivity ignoring other fine-grained polarities like "rave, ugly, pejorative, sarcastic etc.,". Trivial sentiment analysis rates this as positive while for a human it is humiliating. This is an example when lambda composition of visuals fares better compared to textual words. Visually this lambda-composes to a collated thought-enacted movie of one person talking to another having a discourteous, ridiculing facial expression. Lambda functions for this visual compositions could invoke face-feature-vector-recognition algorithms which identify facial expressions. In social media texting smileys(emoticons) convey emotions (e.g tongue-in-cheek). Thus disambiguating the above text sentence accurately depends on a pre-existing visually stored ThoughtNet context of similar sarcastic nature - a person with prior visual ThoughtNet experience hyperedge similar to sarcastic sentence above is more capable of deciphering sarcasm than person without it. This prior edge is "training data" for the model and is stored with an edge-coloring of configurable choosing. Marking sentiments of Thought edges with edge coloring is more qualitative and coarse grained than numbering - each of the emotions Love, Hate, Sorrow, Divinity etc can be assigned a color. As ThoughtNet is grown from beginning of life, stacked up thoughts are colored where the sentiment coloring is learnt from environment. For example, first time a person encounters the above sentence he may not be familiar with sarcasm, but he might learn it is sarcastic from an acquaintance and this learning is stored with a "sarcastic" sentiment coloring in multiplanar thoughtnet hypergraph. Intensity of coloring determines the strength of thought stored. Next instances of evocations ,for example "too good" and "contest" which is the crucial segment, return the above edge if intensely colored (in a complex setting, the lambda composition tree is returned ,not just the sentence).

(*) Example Poetry 3 - lambda evaluation with shuffled connectives, is shown below with nested parenthesization - Each parenthesis scope evaluates a lambda function and composed to a tree - [Walrus-Carpenter - Alice in Wonderland]:
(("The time has come,") (the Walrus said),
("To talk of many things:)
(Of shoes--and ships--and sealing-wax--)
(Of cabbages--and kings--)
And (why (the sea is boiling hot--))
And (whether (pigs have wings.")))
This lambda composition itself can be classified in classes viz., "time", "poetry", "sea" etc., (in special case of gloss overlap this is easy to classify by computing high core numbers of the graph). Above evaluation itself is stored in ThoughtNet as hyperedge in its entirety as the "meta-learning", and not just "knowledge",and an edge coloring for sentiment can be assigned. During future evocation, say utterance of word "time", the edge returned is the most intensely colored edge.

(*) Perfect design of Psycholinguistic text comprehension must involve ego-centric theory specific to each reader as mentioned previously. Combining EventNet and ThoughtNet throws another challenge: When did ThoughtNet begin? Who was the primordial seed of thought? EventNet on the other hand can be axiomatically assumed to have born during BigBang Hyperinflation and the resultant "ultimate free lunch". Invoking Anthropic principle - "I think,therefore I am", "I exist because I am supposed to exist" etc., - suggests the other way i.e EventNet began from ThoughtNet. Then, did consciousness create universe? Quoting an Indological text - Hymn of creation - Nasadiya Sukta - "Even non-existence did not exist in the beginning" which concurs with aforementioned paradoxes. This theorizes duality arose from singularity throwing spanner into spokes of linguistics which are based on duality of synonyms and antonyms. That is a glaringly discordant note in achieving a perfect artificial intelligence. Thus conscious robot may never be a reality.

(*) Above exposition is required to get to the bottom of difficulties in formalising "what is meant by meaning" and "intrinsic merit" in perfect sense. Graph theoretic intrinsic merit algorithms might require quantum mechanical concepts like Bose-Einstein condensate mentioned in 18.10 i.e In recursive gloss overlap graph context, word vertices are energy levels and edges amongst words are subatomic particles.

(*) If ThoughtNet begot EventNet by anthropic principle, there is a possibility language was born before something existed in reality. For example, word "Earth" existed before Earth was created. This presents another circular paradox - ThoughtNet implies EventNet implies ThoughtNet ... because events create thoughts and anthropic principle implies thought created events. This also implies thought created brain and its constituents which in turn think and "understand meaning" of a text - sort of Supreme consciousness splitting itself into individual consciousness.

(*) Another counterexample: A cretan paradox like sentence - "This sentence is a lie" - which is true if it is false and false if true - Lambda function composition algorithm is beyond the scope of such sentences. This sentence exhibits 2 levels of truths and falsehoods - Truth/Falsehood within a sentence and Truth/Falsehood outside the sentence - in former observer and observed are one and the same and in the latter observer stands aloof from observed

(*) If there are two levels of Truth/Falsehoods similar to stratified realities defined in https://sites.google.com/site/kuja27/UndecidabilityOfFewNonTrivialQuestions.pdf, above paradox "This sentence is a lie" ican be analyzed (kind of Godel sentence proving languages are incomplete) as below with TRUE/FALSE and true/false being two levels of this nest:

- This sentence is a lie - true - self-refuting and circular
- This sentence is a lie - TRUE - not self-refuting because TRUE transcends "true"
- This sentence is a lie - false - self-refuting and circular
- This sentence is a lie - FALSE - not self-refuting because FALSE transcends "false"

(*) (QUITE PRESUMPTIVE, THIS MIGHT HAVE A PARALLEL IN LOGIC BUT COULDN'T FIND IT) In other words, the sentence "(It is TRUE that (this sentence is a lie))" is different from "(It is true that this sentence is a lie)" - in the former the observer is independent of observed sentence (complete) where as in the latter observer and observed coalesce (incomplete) - TRUE in former has scope beyond the inner sentence while true in latter is scope-restricted to the inner sentence. If Truth and Falsehood are scoped with a universe of discourse, above paradox seems to get reconciled and looks like an extension of Incompleteness theorems in logic. Natural languages and even Formal Logic have only one level of boolean truth/falsehood and thus previous is mere a theoretical fancy.

(*) Above philosophical dissection of meaning has striking similarities to quantum mechanics - If language has substructure abiding by quantum mechanics [e.g Bose-Einstein model for recursive gloss overlap wordnet subgraph], the links across words which correspond to subatomic particles must obey Heisenberg uncertainty principle also i.e As a linguistic parallel, 100% accurate meaning may never be found by any model of computation. This can not be ruled out if network is a random graph with probabilities for edges.

(*) Levels of Truth are reminiscent of Tarski's Undefinability of Truth Theorem - https://plato.stanford.edu/entries/goedel-incompleteness/#TarTheUndTru - There are two levels of languages - Inner Object Language and Outer Meta Language - Truth of statements in inner object languages are determined in outer meta language. An example sentence in object language is the Cretan Paradox - "All cretans are liars". In previous example, "this sentence is a lie" is defined in object language (English) and is a self-referential Goedel sentence. Truth of this sentence has to be decided in metalanguage. Another example paradox sentence in object language is: "This is bigger than the biggest". Can truth of

this be determined in object language? This can be geometrically interpreted: If object language and metalanguage are vector spaces of dimensions k and k+l, any sentence in object language is a vector of dimension k (e.g word2vec representation extended to sentence2vec) and truth of it has to be decided in metalanguage space of dimensions k+l. In object language space there can not be an entity bigger than biggest - e.g set of concentric circles limited by outermost circle and truth of this sentence is "False". But when this sentence is projected to a k+l metalanguage vector space, a circle "bigger" in k+l dimension vectorspace can exist, than the "biggest" circle in k dimension vectorspace and truth is "True". Example of dimensions in a natural language are Parts-of-Speech.

---

229. (THEORY) Bose-Einstein Condensate model for Recursive Gloss Overlap based Classification

---

Bose-Einstein condensate fitness model by [Ginestra Bianconi] described in 18.10 for any complex network graph is:

Fitness of a vertex = 2^(-b*Energy) where b is a Bose-Einstein condensate function.

Here fitness of a vertex is defined as ability to attract links to itself - Star complexity. It is not known if ability of a word to attract links from other words is same as intrinsic merit of a document in recursive gloss overlap context - prima facie both look equal. As energy of a node decreases, it attracts increased number of node edges. In Recursive Gloss Overlap graph, node with maximum core number or PageRank must have lowest energy and hence the Fittest. Previous equation is by applying Einstein derivation of Satyendranath Bose's Planck's formula for Quantum mechanics where in a critical temperature above zero kelvin most of the particles "condense" in lowest energy level. This was proved in 1995-2001 by NIST lab (http://bec.nist.gov/).

Applying the above model to a relational graph obtained from text document implies graph theoretic representation of language has lurking quantum mechanical traits which should apply to all semantic graphs and ontologies viz.,WordNet, ConceptNet etc.,. For example, when a winner word vertex takes all links to other words, the winner word becomes a dictator and thus the only class a document can belong to. Thus Bose-Einstein condensate can be mapped to a Dictator boolean function. This should ideally be extensible to structural complexity of circuit graphs i.e gate within a neural net TC circuit version of WordNet with maximum fanin and fanout has lowest energy where neighbours flock. If energy implies information entropy in a document, low energy implies low entropy of a text i.e sigma(p log p) of a text as bag of words probability distribution, decreases as text document increasingly becomes less chaotic and vice-versa holds too. As a document becomes more chaotic with high entropy and energy, fitness per word vertex decreases as there are multiple energy centres within it. Hence low entropy is equivalent to presence of a high fitness word vertex in the graph.

Analogy: Recursive Gloss Overlap graph can be construed as a 3 dimensional rubbersheet with low energy words as troughs. Deepest troughs are words (lowest energy levels) that classify a document.

Reference:

229.1 Linked [Albert Laszlo Barabasi] - Chapter 8 - Einstein's Legacy
229.2 Bianconi-Barabasi Bose-Einstein Condensation in Networks, Simplicial Complexes, Intrinsic Fitness - http://www.eng.ox.ac.uk/sen/files/symposium2017/bianconi.pdf

---

230. (THEORY) Expander Graphs, Bose-Einstein Condensate Fitness model and Intrinsic Merit

---

Edge expander graphs are defined with Cheeger's constant for minimum high edge boundary per vertex subset:

minimum(|boundary(S)|/|S|), 1 < |S| < n/2

where boundary(S) of a subset S of vertices of G (V(G)) is:

boundary(S) = { (u,v) | u in S, v in V(G) \ S}

From Bose-Einstein condensate fitness model, ability of a vertex to attract links increases with decreasing energy of a vertex. This implies boundary(S) expands with presence of low energy vertices. Thus Bose-Einstein condensate model translates into an edge expander construction. Also high expansion implies high star complexity. Recursive Gloss Overlap graph with a low energy vertex is a high edge expander and easiliy classifiable to a topic with core numbers.

Measuring intrinsic merit (and therefore fitness if both are equivalent concepts) by edge expansion (Cheeger's constant) over and above usual connectivity measures formalises a stronger notion of "meaningfulness" of a text document. For example High Cheeger's constant of RGO graph implies high intrinsic merit of a text.

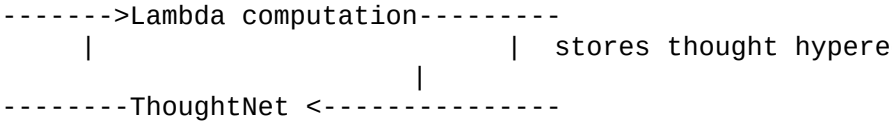
---

231. (THEORY) Sentiment analysis and ThoughtNet Continuum - Fine grained emotions and evocation - related to 228

(*) ThoughtNet need not be a discrete hypergraph but a continuum of edges and nodes. Analogy: Set of countably infinite natural numbers and continuum of uncountably infinite reals. It is difficult to visualize a continuum as a graph - no graph theoretic gadget exists for a continuum graph - Infinite graph is countable, Continuum graph is uncountable. By diagonalization, continuum could be as hard as halting problem.

(*) Each feature of a visual thought is a class stack node in ThoughtNet. Number of features per visual could be thousands. Visual is flattened into a hyperedge across class stack nodes corresponding to feature vector vertices.

(*) Example schematic:



(*) Colors of Thought hyperedges are determined by sentimentality of class stack nodes. The class stack nodes are colored too, not just edges. If an edge transcends various colored class stack nodes, resultant color of thought hyperedge is a function of colors of constituent class stack nodes:

Edge sentiment color of ThoughtNet Hyperedge = weight_function(node colors in the hyperedge).

(*) Node colors are priors. How these priors are decided for each class stack node is the mainstay of emotional inference. This requires assumption of Desire/ Prejudice/Bounded Rationality/Irrational Exuberance arising from Consciousness or equivalent. This irrationality creates a pre-conceived potential per class stack node, though no edges may be part of the class at all. Irrationality has to be assumed to be an independent variable and atomic because if it is not, what it depends on is a question. Though previous inquisition on Sentiments, Meaning et al border on Theology, it is inevitable to exclude the emotional root causes while working on a close-to-perfect mathematical simulation of human sentiments.

----
232. VIRGO commits for AsFer Software Analytics with SATURN Program Analysis - 29 February 2016, 1 March 2016

Software Analytics - SATURN Program Analysis added to VIRGO Linux kernel drivers
----

- SATURN (saturn.stanford.edu) Program Analysis and Verification software has been
integrated into VIRGO Kernel as a Verification+SoftwareAnalytics subsystem
- A sample driver that can invoke an exported function has been added in drivers
- saturn_program_analysis
- Detailed document for an example null pointer analysis usecase has been created in virgo-docs/VIRGO_SATURN_Program_Analysis_Integration.txt
-
linux-kernel-extensions/drivers/virgo/saturn_program_analysis/saturn_program_analysis_trees/error.txt is the error report from SATURN
- SATURN generated preproc and trees are in
linux-kernel-extensions/drivers/virgo/saturn_program_analysis/preproc and
linux-kernel-extensions/drivers/virgo/saturn_program_analysis/
saturn_program_analysis_trees/

----
233. (FEATURE-DONE) Commits as on 3 March 2016 - PERL WordNet::Similarity subroutine for pair of words
----
----
*) New perl-src folder with perl code for computing WordNet Distance with Ted Pedersen et al WordNet::Similarity CPAN module
*) Logs for some example word distances have been included in testlogs. Some weird behaviour for similar word pairs - distance is 1 instead of 0. Probably a clockwise distance rather than anticlockwise.

This leverages PERLs powerful CPAN support for text data processing. Also python can invoke PERL with PyPerl if necessary.

----
234. (FEATURE-DONE) PERL WordNet::Similarity build and python issues notes
----

- NLTK WordNet corpora despite being 3.0 doesnot work with WNHOME setting (errors in exception files and locations of lot of files are wrongly pointed to)
- Only Direct download from Princeton WordNet 3.0 works with WNHOME.
- Tcl/Tk 8.6 is prerequisite and there is a build error "missing result field in Tk_Inter" in /usr/include/tcl8.6/tcl.h. Remedy is to enable USE_INTERP_RESULT flag with macro: #define USE_INTERP_RESULT 1 in this header and then do:
      make Makefile.PL
      make
      make install
in WordNet::Similarity with WNHOME set to /usr/local/WordNet-3.0.
- PyPerl is no longer in active development - so only subprocess.call() is invoked to execute perl with shell=False.

----
235. (THEORY) Dream Sentiment Analysis with ThoughtNet - related to 18 and 228

Dream Analysis in http://cogprints.org/5030/1/NRC-48725.pdf (mentioned in 18.8)

scores dreams based on polarity with bag of words representation- dreams are known to be related to Limbic and Paralimbic systems in brain. Insofar as theories pertaining to ThoughtNet described in 18 and 228 are concerned, only conscious evocatives arising from events are described. Going a step further, can ThoughtNet account for interpretation of dreams? Dreams, psychologically, are known to be vagaries of suppressed thoughts manifesting in subsconscious state. ThoughtNet stores thoughts of experiences,visuals and events as hyperedges consciously. A dream model is proposed where in subsconscious/RapidEyeMovement state, Thought hyperedges momentarily disintegrate and re-arrange to form new edges manifesting as dream visuals. Once dream expires, these dream edges are dissolved and original ThoughtNet is restored. Before dissolution, dream might be stored if potentially strong. This explains why some dreams linger after wakeup. Caveat: this is just a theory of dream analysis based on some thought experimentation and has no scientific backing. This model makes an assumption that dream hyperedges are functions of preexisting ThoughtNet hyperedges which is not quite right because there are exceptional phenomena like extra-sensory perception where visions of future are observed by few individuals which could not have arisen from past experience and thoughts. ESP implies brain has faculty to foresee inherently in a superconsicous state which is subdued in normal conscious state. It is more accurate to say that dream hyperedges are functions of both past thought hyperedges and superconscious hidden variables which might explain ESP.

---
236. (THEORY) Regularity Lemma and Ramsey Number of Recursive Gloss Overlap graph and ThoughtNet
---
Throughout this document, ranking of documents which is so far a pure statistical and machine learning problem is viewed from Graph Theory and Lambda Function growth perspective - theory invades what was till now engineering. Both Regularity Lemma and Ramsey theory elucidate order in large graphs. Regularity Lemma has a notion of density which is defined as:

$$d(V1, V2) = |E(V1, V2)|/|V1||V2|$$

where V1 and V2 are subsets of vertices V(G) for a graph G and E(V1,V2) are edges among subsets V1 and V2. Density is a quantitative measure of graph complexity and intrinsic connectedness merit of a text document. Ramsey Number R(a,b) of a graph is the least number such that there always exists a graph of vertex order R(a,b) with a clique of size a or an independent set of size b. Typically associated with "Party problem" where it is required to find minimum number of people to be invited so that a people know each other or b people do not know each other, Ramsey number of ThoughtNet sentiment colorings - e.g bipartisan red-blue colorings for 2 thought hyperedge sentiment polarities - provides a theoretical bound on size of ThoughtNet as such so that order arises in a motley mix of emotionally tagged thoughts - a clique of similar sentimental edges emerges. Similar application of Ramsey number holds good for Recursive Gloss Overlap graph also - a document's graph can have cliques and independent sets with vertex colorings for word sentiments.

==========
237. (FEATURE-DONE) Commits as on 11 March 2016 - Deep Learning Convolution - Multi Feature Convolution Map Kernel Filters
==========
- Convolution Network supports Multiple Feature Maps (presently 3)
- New example bitmap for feature recognition of pattern "3" inscribed as 1s has been introduced
- Final neural network from Max pooling layer now is randomized with randint() based weights for
each of the 10 neurons
- Logs for this have been committed to testlogs
- Logs show a marked swing in Maxpooling map where the segments of pattern "3"

are pronounced.
- Final neural layer shows a variegated decision from each neuron for corresponding 3 convolution maps

---
238. (FEATURE-DONE) Commits as on 14 March 2016

- Some bugs resolved
- Added one more example with no pattern
- Convolution is computed for all 3 bitmap examples
- Final neuron layer now is a function of each point in all maxpooling layers
- The existence of pattern is identified by the final output of each of 10 neurons
- Patterns 0 and 3 have a greater neural value than no pattern. Gradation of neural value indicates intensity of pattern.
- Above is a very fundamental pattern recognition for 2 dimensional data. Sophisticated deconvolution is explained in http://arxiv.org/abs/1311.2901 which reverse engineers pooling layers with Unpooling.
- 3 logs for this commit have been included in testlogs/
- random weighting has been removed.

---
239. (THEORY) Deep Learning Convolution Network and a boolean function learning algorithm of different kind
---
Deep Learning Convolution Network for Image Pattern Recognition elicits features from the image considered as 2-dimensional array. In the
Deep Learning implementation in python-src/ image is represented as {0,1}* bitmap blob. Each row of this bit map can be construed as satisfying assignment to some boolean function to be learnt. Thus the 2-dimensional bitmap is set of assignments satisfying a boolean function - A boolean function learnt from this data - by some standard algorithms viz., PAC learning, Hastad-Linial-Mansour-Nisan fourier coefficient concentration bounds and low-degree polynomial approximation (learning phase averages the coefficients from all sample points and prediction phase uses this averaged fourier coefficient) - accepts the image as input. Deep Learning Convolution Network is thus equivalent to a boolean function learnt from an image - quite similar to HMLN low-degree learning, deep learning convolution also does weighted averaging in local receptive field feature kernel map layers, maxpooling layer and final neuron layers with binary output. Essentially convolution learns a TC circuit. It presupposes existence of set of satisfying assignments which by itself is a #P-complete problem. This connects two hitherto unrelated concepts - Image Recognition and Satisifiability - into a learning theory algorithm for 2-dimensional binary data. Same learning theory approach may not work for BackPropagation which depend on 4 standard Partial Differential Equations.

---
240. (FEATURE-DONE) Commits as on 15 March 2016

- DeepLearning BackPropagation implementation:
    - An example Software Analytics usecase for CPU and Memory usage has been included
    - Number of Backpropagation weight update iterations has been increased 10 fold to 3000000.
    - logs for some iterations have been included in testlogs/
- DeepLearning Convolution Network implementation:
    - Image patterns have been changed to 0, 8 and no-pattern.

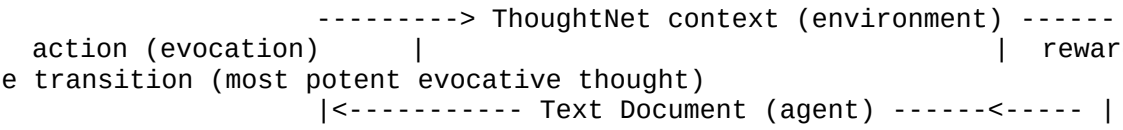
- Final neuron layer weights have been changed by squaring to scale down the output - this differentiates patterns better.
- logs for this have been included in testlogs/

---

241. (THEORY and IMPLEMENTATION) ThoughtNet and Reinforcement Learning

---

Reinforcement Learning is based on following schematic principle of interaction between environment and agent learning from environment. ThoughtNet described in 228 and previous is an environmental context that every human agent has access to in decision making. This makes human thought process and thought-driven-actions to be a special case of Reinforcement Learning. Reinforcement Learning is itself inspired by behavioural psychology.



Schematic above illustrates text document study as reinforcement learning. During left-right reading of text, evocation action is taken per-word which accesses ThoughtNet storage and returns a reward which is the strongest thought edge. State transition after each evocation is the partially evaluated lambda composition prefix mentioned in 222 - this prefix is updated after reading each word left-right. Human practical reading is most of the time lopsided - it is not exactly left-right but eye observes catchy keywords and swivels/oscillates around those pivots creating a balanced composition tree. Error in disambiguation can be simulated with a model similar to what is known as "Huygen's Gambler's Ruin Theorem" which avers that in any Gambling/Betting game, player always has a non-zero probability of becoming insolvent - this is through a Bernoulli trial sequence as below with fair coin toss (probability=0.5):

- 1) First coin flip - Pr(doubling1) = 0.5, Pr(bankrupt1) = 0.5
- 2) Second coin flip - Pr(bankrupt2) = Pr(bankrupt1)*0.5 = 0.25
- 3) Third coin flip - Pr(bankrupt3) = Pr(bankrupt2)*0.5 = 0.125
- ....

By union bound probability of gambler going bankrupt after infinite number of coin flips is 1. This convergence to 1 applies to unfair biased coin tosses also. This theorem has useful application in ThoughtNet reinforcement learning disambiguation. Reward is defined as probability of most potent evocative thought being retrieved from ThoughtNet (this is when return value is set of hyperedges and not unique - probability of reward is 1/size_of_set_returned) and state transition is akin to consecutive coin flips. This proves that probability of imperfect disambiguation is eventually 1 even though reward probability is quite close to 1.

A ThoughtNet Reinforcement Learning implementation has been committed as part of python-src/DeepLearning_ReinforcementLearningMonteCarlo.py

References:

241.1 Reinforcement Learning - [Richard Sutton] - http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html, https://www.dropbox.com/s/b3psxv2r0ccmf80/book2015oct.pdf?dl=0.

---

242. (FEATURE-BENCHMARK-DONE) Interview Algorithm PySpark-Cython implementation benchmark with and without Spark configurables

---

This benchmark is done with both:
InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.py

```
InterviewAlgorithmWithIntrinisicMerit_SparkMapReducer.py
both precompiled with Cython to create .c source files and .so libraries.

Number of keywords in newly crawled web page: 35

---------------------------
With spark-defaults.conf :
---------------------------
Execution 1:
------------
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-
github-code/python-src/InterviewAlgorithm/testlogs# for i in `grep Finished
InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.2localthreads.CythonOpt
imized.16March2016.out |grep TaskSetManager| awk '{print $14}'`
> do
> sum=$(expr $sum + $i)
> done

root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-
github-code/python-src/InterviewAlgorithm/testlogs# echo $sum
243058
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-
github-code/python-src/InterviewAlgorithm/testlogs# grep Finished
InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.2localthreads.CythonOpt
imized.16March2016.out |grep TaskSetManager| awk '{print $14}'|wc -l
156
Per task time in milliseconds: ~1558

------------
Execution 2:
------------
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-
github-code/python-src/InterviewAlgorithm/testlogs# for i in `grep Finished
InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.2localthreads.CythonOpt
imized2.16March2016.out |grep TaskSetManager| awk '{print $14}'`
> do
> sum=$(expr $sum + $i)
> done
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-
github-code/python-src/InterviewAlgorithm/testlogs# echo $sum
520074
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-
github-code/python-src/InterviewAlgorithm/testlogs# grep Finished
InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.2localthreads.CythonOpt
imized2.16March2016.out |grep TaskSetManager| awk '{print $14}'|wc -l
312
Per task time in milliseconds: ~1666

-----------------------------
Without spark-defaults.conf
-----------------------------
Execution 3:
------------
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-
github-code/python-src/InterviewAlgorithm/testlogs# echo $sum
69691
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
```

9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-github-code/python-src/InterviewAlgorithm/testlogs# grep Finished InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.2localthreads.CythonOptimized3.16March2016.out |grep TaskSetManager| awk '{print $14}'|wc -l
156
Per task time in milliseconds: ~446 which is twice faster than earlier benchmark done in February 2016 and almost 4 times faster than previous 2 executions with spark-defaults.conf enabled.

A spark-defaults.conf file has been committed to repository (Reference: http://spark.apache.org/docs/latest/configuration.html). With Spark configuration settings for memory and multicore, something happens with Spark performance which is counterintuitive despite document size being almost same.

---
243. Commits (1) as on 16,17 March 2016

PySpark-Cython Interview Algorithm for Merit - Benchmark

- 3 executions with and without spark-defaults.conf config settings have been benchmarked
- Cython setup.py has been updated to compile both InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.py and InterviewAlgorithmWithIntrinisicMerit_SparkMapReducer.py
to create .c and .so files
- Benchmark yields some intriguing results:
	- Executions with spark-defaults.conf enabled are 4 times slower than executions without spark-defaults.conf
	- Execution without spark config is twice faster than earlier benchmark
- Logs and screenshots for above have been committed to testlogs
- Text document has been recrawled and updated (number of keywords almost same as previous benchmark)
- Spark config file has been committed

---
244. Commits (2) as on 16,17 March 2016

More benchmarking of PySpark-Cython Intrinsic Merit computation
---
- Enabled Spark RDD cacheing with cache() - storage level MEMORY
- Recompiled .c and .so with Cython
- uncommented both lines in Cython setup.py
- logs and screenshots for above have been committed in testlogs/
- locking.acquire() and locking.release() have been commented
- With this per task duration has been brought down to ~402 milliseconds on single node cluster. Ideally on a multinode cluster
when tasks are perfectly distributable, this should be the runtime.

With spark-defaults.conf - multiple contexts disabled and reuse worker enabled
---
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-github-code/python-src/InterviewAlgorithm/testlogs# echo $sum
246796
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-github-code/python-src/InterviewAlgorithm/testlogs# grep Finished InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.2localthreads.CythonOptimized.17March2016.out |grep TaskSetManager| awk '{print $14}'|wc -l
156
Per task time milliseconds: ~1582

Without spark-defaults.conf - locking.acquire()/release() disabled and Spark RDD cacheing enabled
---
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-github-code/python-src/InterviewAlgorithm/testlogs# echo $sum
248900
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-github-code/python-src/InterviewAlgorithm/testlogs# grep Finished InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.2localthreads.CythonOptimized2.17March2016.out |grep TaskSetManager| awk '{print $14}'|wc -l
618
Per task time milliseconds: ~402 (fastest observed thus far per task; Obviously something is wrong with spark-defaults.conf enabled)

---
245. (FEATURE-BENCHMARK-DONE) Interview Algorithm PySpark-Cython implementation benchmark with Spark configurables
- Commits as on 21 March 2016

Following benchmark was done with a new spark-defaults.conf (committed as spark-defaults2.conf):

root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-github-code/python-src/InterviewAlgorithm/testlogs# echo $sum
965910
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/GitHub/asfer-github-code/python-src/InterviewAlgorithm/testlogs# grep Finished InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.2localthreads.CythonOptimized.21March2016.out |grep TaskSetManager| awk '{print $14}'|wc -l
618
Time per task in milliseconds: ~1562 (Again, with spark-defaults.conf with some additional settings, there is a blockade)

---
246. (THEORY) Learning a variant of Differential Privacy in Social networks
---
Nicknames/UserIds in social media are assumed to be unique to an individual. Authentication and authorization with Public-Key Infrastructure
is assumed to guarantee unique identification. In an exceptional case, if more than one person shares a userid intentionally or because of cybercrime to portray a false picture of other end, is it possible to distinguish and learn such spoof behaviour? This is a direct opposite of differential privacy [Cynthia Dwork - http://research.microsoft.com/pubs/64346/dwork.pdf] where 2 databases differing in one element are indistinguishable to queries by a randomized algorithm with coin tosses. But in previous example of shared id(s) in social network, distinguisher should be able to discern the cases when a remote id is genuine and spoofed/shared - probably suitable name for it is Differential Identification. This can theoretically happen despite all PKI security measures in place when id is intentionally shared i.e a coalition of people decide to hoodwink PKI by sharing credentials and the receiving end sees them as one. For example, if a chat id is compromised by a coalition, the chat transcript is no longer credible reflection of the other end, and receiving end may not be aware of this at all. Here chat transcript is a mix of both genuine and spoofed portrayal of other end. An example statistical Learning algorithm for

distinguishing when spoof happened analyzes the chat transcript conversations with genuine chat as priors if there is a prior available. Non-statistically, it is not known how a boolean function learner would look like for this example. A special case is when a coalition is unnecessary. The remote end can be a single individual garnering collective wisdom from multiple sources to present a deceptive portrayal of multitude.

---
247. (FEATURE-DONE) Commits as on 24 March 2016 - Code Optimizations - Cacheing and Loop invariant - Interview Algorithm
---

- New Cacheing datastructures have been included for storing already found previous level backedge vertices for each level and
already found gloss tokens. Presently these caches are python dictionaries only and in a distributed setting this can be replaced with an
object key-value stores like memcached.(InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit_SparkMapReducer.py)
- There was an inefficient loop invariant computation for prevelevesynsets_tokens which has been moved to outermost while. (InterviewAlgorithm/InterviewAlgorithmWithIntrinisicMerit_Crawl_Visual_Spark.py)
- spark-default2.conf has been updated with few more Spark config settings
- Spidered text has been recrawled
- logs and screenshots for this commits have been included in testlogs/
- Cython .c , .so files have been rebuilt
- Cacheing above in someway accomplishes the isomorphic node removal by way of Cache lookup

---
---
248. (FEATURE-BENCHMARK-DONE) Intrinsic Merit PySpark-Cython implementation - With and Without cacheing
- Commits as on 25 March 2016

1.Cache md5 hashkey has been changed to be the complete tuple (tokensofprevlevel,keyword) - the rationale is to have perfect uniqueness to find backvertices for a keyword and gloss from previous recursion step.
2.Following benchmark was done by disabling and enabling lookupCache and lookupCacheParents boolean flags.
3.Cache values are "Novalue" string literals by default initialized in lambda and the Cache updates are not append()s but
simple key-value assignments.
--
Number of keywords: 7
Without Cacheing: 9 minutes for all tasks (including NetworkX and graphics rendering times which is quite considerable)
With Cacheing: 9 minutes for all tasks (including NetworkX and graphics rendering times which is quite considerable)

Summation of individual task times in milliseconds and averaging sometimes gives misleadingly high duration per task though
total execution is faster. Because of this start-end time duration is measured. For above example Cacheing has no effect because
there are no repetitions cached to do lookup. Logs and screenshots with and without cacheing have been committed to testlogs/

---
249. (THEORY) An alternative USTCONN LogSpace algorithm for constructing Recursive Gloss Overlap Graph

---
Benchmarks above are selected best from multiple executions. Sometimes a mysterious slowdown was observed with both CPUs clocking 100% load which could be result of JVM Garbage Collection for Spark processes among others. These benchmarks are thus just for assessing feasibility of Spark clustering only and a perfect benchmark might require a cloud of few thousand nodes for a crawled webpage. At present there is no known standard algorithm to convert a natural language text to a relational graph without WordNet and internally looking up some *Net ontology to get path relations between two words could be indispensable theoretically speaking. This is the reverse process of Graph Traversals - Breadth First and Depth First - traversal is the input text and a graph has to be extracted from it. For n keywords, there are n^2 ordered pairs of word vertices possible for each of which WordNet paths have to be found. This is Undirected ST Connectivity in graphs for which there are quite a few algorithms known right from [Savitch] theorem upto [OmerReingold] ZigZag product based algorithm in logspace. This is more optimal compared to all pairs shortest paths on WordNet. This algorithm does not do deep learning recursion top-down to learn a graph from text. For all pairs this is better than O(n^3) which is upperbound for All Pairs Shortest Paths. These n^2 pairs of paths overlap and intersect to create a graph. Presently implemented Recursive Gloss Overlap learns deeper than all pairs USTCONN algorithm because it uses "superinstance" notion - whether x is in definition of y - something absent in WordNet jargon. This algorithm is quadratic in number of words where as Recursive Gloss Overlap is linear - O(n*d*s^tmax/cpus) - counterintuitively.

---
250. Commits as on 29 March 2016

- New subroutine for printing WordNet Synset Path between 2 words - this prints edges related by hypernyms and hyponyms
- an example log has been committed in testlogs/

From the example path in logs, distance is more of a metapath than a thesaurus-like path implemented in Recursive Gloss Overlap in NeuronRain. Due to this limitation of WordNet hyper-hyponyms the usual unsupervised classifier based on core numbers of the graph translated from text would not work as accurately with straightforward WordNet paths. This necessitates the superinstance relation which relates two words k and l: k is in definition of l.

---
251. (FEATURE-DONE) Commits (1) as on 30 March 2016 - MemCache Integration with Spark-Cython Intrinsic Merit Computation
---
- In this commit, native python dictionary based cache lookup for Spark_MapReduce() and Spark_MapReduce_Parents() is replaced with standard MemCache Distributed Object Cacheing get()/set() invocations.
- Logs and Screenshots for above have been committed to testlogs/
- Prerequisites are MemCached (ubuntu apt package) and Python-memcache (pip install) libraries
- a recrawled webpage has been merit-computed
- MemCached is better than native dict(s) because of standalone nature of Cache Listener which can be looked up from anywhere on cloud

---
252. (FEATURE-DONE) Commits (2) as on 30 March 2016 - Spark-Cython Intrinsic Merit computation with spark-defaults.conf enabled
---

- spark-defaults2.conf has been enabled with extra java options for performance (reference: http://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html)
- lookupCache and lookupCacheParents boolean flags have been done away with because memcache has been enabled by default for subgraph cacheing
- With this spark logs show more concurrency and less latency. Dynamic allocation was in conflict with number of executors and executors has been hence commented in spark-defaults2.conf
- text document has been updated to have ~10 keywords. This took almost 7 minutes which looks better than one with spark-defaults.conf disabled
- Screenshot and logs have been committed in testlogs/

---
253. (FEATURE-DONE) Commits as on 1 April 2016

Loopless Map function for Spark_MapReduce()

- new Map function MapFunction2() has been defined which doesn't loop through all keywords per level, but does the glossification per keyword which is aggregated by Reduce step. MapFunction() does for loop for all keywords per level.
- a dictionary cacheing with graphcachelocal has been done for lookup of already gloss-tokenized keywords. This couldn't be memcached due to a weird error that resets the graphcachelocal to a null which shouldn't have.
- This has a execution time of 6 minutes overall for all tasks compared to for-loop Map function in Spark_MapReduce().
- logs and screenshots for Loopless Map with and without graphcachelocal have been committed
- Rebuilt Cython .c, .o, .so, intrinsic merit log and DOT files

---
254. (THEORY) Hypercontractivity - Bonami-Beckner Theorem Inequality, Noise Operator and p-Norm of Boolean functions
---
Hypercontractivity implies that noise operator attenuates a boolean function and makes it close to constant function. Hypercontractivity is defined as:

      |Trhof|q <= |f|p where 0 <= rho <= sqrt((p-1)(q-1)) for some p-norm and q-norm

and in its expanded form as:

      sigma(rho^|S|*(1/2^n*sigma(|f(x)|^q*(-1)^(parity(xi))))^(1/q)*parity(S)) <= (1/2^n*sigma(|f(x)|^p)^(1/p)

Hypercontractivity upperbounds a noisy function's q-norm by same function's p-norm without noise. For 2-norm, by Parseval's theorem which gives 2-norm as sigma(f(S)^2), hypercontractivity can be directly applied to Denoisification in 53.11 and 53.12 as it connects noisy and noiseless boolean function norms.

Reference:

254.1 Bonami-Beckner Hypercontractivity - http://theoryofcomputing.org/articles/gs001/gs001.pdf

---
255. (THEORY) Hypercontractivity, KKL inequality,Social choice boolean functions - Denoisification 3

For 2-norm, Bonami-Beckner inequality becomes:

sigma[(rho^|S|*fourier_coeff(S)^2)^(2/2)] <= [1/2^n*(sigma(f(x)^p)^2/p)]

For boolean functions f:{0,1}^n -> {-1,0,1}, [Kahn-Kalai-Linial] inequality is special case of previous:

For rho=p-1 and p=1+delta, sigma(delta^|S|*fourier_coeff(S)^2) <= (Pr[f!=0])^2/(1+delta)

where p-norm is nothing but a norm on probability that f is non-zero (because Pr[f!=0] is a mean of all possible f(x) at 2^n points).

From 53.12 size of a noisy boolean function circuit has been bounded as:
Summation(rho^|S|*fourier_coeff(S)^2) * 2^((t^(1/d)/20)-1) <= Size
for |S| > t. From KKL inequality, 2-norm in LHS can be set to maximum of (Pr[f!=0])^2/(1+delta) in 53.12 and Size lowerbound is obtained for a denoisified circuit:

(Pr[f!=0])^2/(1+delta) * 2^((t^(1/d)/20)-1) <= Size

which is exponential when delta=1 and Pr[f!=0] is quite close to 1 i.e f is 1 or -1 for most of the inputs.

If f is 100% noise stable, from Plancherel theorem:

sigma(rho^|S|*f(S)^2) = 1, S in [n]

For |S| < t=n, 1-Summation(rho^|S|*fourier_coeff(S)^2) <= 2*(Size)*2^(-t^(1/d)/20)

But for rho=p-1 and p=1+delta, 1-Summation(delta^|S|*fourier_coeff(S)^2) > 1-(Pr[f!=0])^2/(1+delta) from KKL inequality.

For |S| < t=n, 1-(Pr[f!=0])^2/(1+delta) < 1-Summation(rho^|S|*fourier_coeff(S)^2) <= 2*(Size)*2^(-t^(1/d)/20)

=> 0.5*(1-(Pr[f!=0])^2/(1+delta))*2^(t^(1/d)/20) < 0.5*(1-Summation(rho^|S|*fourier_coeff(S)^2))*2^(t^(1/d)/20) <= Size

Again denoisification by letting rho and delta tend to 0 in above, places an exponential lowerbound, with constant upperbound on depth, on a noiseless circuit size. In above size bound Lt n^(1/n) = 1 for n tending to infinity (Real analysis result - http://www.jpetrie.net/2014/10/12/proof-that-the-limit-as-n-approaches-infinity-of-n1n-1-lim_n-to-infty-n1n-1/). Percolation in 100% noise stable regime can have NC/poly circuits while above stipulates the lowerbound for noiseless circuits to be exponential when depth is constant. It implies NC/Poly formulation of percolation has unrestricted depth which is obvious because 100% noise stability regime is attained when n tends to infinity. Randomized algorithm equivalent of Percolation implies Percolation is in RP or RNC with pseudorandom advice which vindicates derandomized NC representation for percolation. This leads to an important result if 100% stability implies lowerbound: LHS is an NC/Poly Percolation circuit in 100% noise stability regime of polynomial size. RHS is exponential, denoisified, 100% stable circuit of arbitrary hardness. From 53.9 and 53.16, RHS has an NC/Poly lowerbound e.g if RHS is PH-complete then PH in P/poly. From Toda's theorem, PH is in P^#P and there is a known result that P^#P in P/Poly implies P^#P = MA. Unifying:
PH in P^#P in P/Poly => P^#P=MA. (assumption: NC/Poly is in P/Poly because any logdepth, polysize bounded fanin circuit has polysize gates)
NC/log percolation circuit would imply collapse of polynomial hierarchy - RHS PH-complete is lowerbounded by LHS NC/log in P and trivially P=NP.

Percolation boolean functions have a notion of revealment - that is, there exists a randomized algorithm corresponding to each percolation boolean function which has coin toss advice to query the next bit and revealment is the maximum probability that a bit index i in [n] is queried in these sets of bits B in [n]- it reveals the secret in randomness. Thus LHS Percolation in Pr(Good) circuit can be simulated by a randomized algorithm with revealment. In such a case, advice strings are random coin tosses. This revealment has close resemblance to random restrictions - each restriction brings down a variable by revealing it finally leading to a constant function. Following theorem connects Fourier weights of percolation and Randomized revealments:

Sigma(fourier_coeff(S)^2) <= revealment*constant*2-norm(f)

There is a special case in which size of advice list can be brought down to logn

from n for Z(n) grid. Set of logn points are pseudorandomly chosen from n points. This logn sized subset when sorted would still point to left-right trend unless leftmost and rightmost are not in sample. This places Percolation in NC/log which could imply a PH collapse mentioned previously. This is an exponential decrease in size of advice.

Reference:

255.1 Randomized Algorithms and Percolation - http://math.univ-lyon1.fr/~garban/Slides/Newton.pdf

---
256. (THEORY) Majority Voting in Multipartisan Elections

Uptill now, P(Good) majority voting in this document describes only 2-candidate elections with a majority function voting circuit.
Noise stability of a voter's decision boolean function is assumed as a suitable measure of voter/voting error in RHS of P(Good) summation. Most generic RHS is when there is a multiway contest and error is bounded. P(Good) binomial summation still applies for multipartisan democracy because voter decision vector (set of binary decision bits of all voters - in {Good,Bad}*) remains same though number of candidates increase manifold. Differences are in:
- Individual Voter Judging Functions which are not boolean functions and
- Generic Majority Function that needs to find most prominent index from an assorted mix of votes (Heavy hitter) - might require Sorting Networks on Hashvalues.
- There are three variables - number of variables per voter, number of voters and number of candidates (which is 2 in boolean decision functions special case for fixed 2 candidate elections) and these numbers can be arbitrarily huge - in the order of Graham's number for example.

Voter judging functions have to choose an index within n candidates than simple 0-1. In a special case non-boolean functions can be simulated by a functional aggregation of boolean functions e.g Voted Candidate index in binary representation is nothing but a logN sized binary string of 0-1 output gates from multiple boolean functions for maximum of N candidates. Measuring how good is the choice made by a multiway voter decision function has no equivalent notion of stability and sensitivity in non-boolean functions. For 2 voters 0-1 boolean function is sufficient. Multiparty Generic Majority Function without tie in which candidates are hashkeys and votes are hashvalues can be shown to be NP-complete from Money Changing Problem and 0-1 Integer Linear Programming(https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions_2014.pdf). [Friedgut-Kalai-Nisan] theorem is quantitative extension of Arrow's Theorem and Gibbard-Satterthwaite Theorem which prove that there is a non-zero probability that elections with 3 candidates and more can be manipulated. This appears sufficient to prove that Goodness probability of a multiway contest in RHS of P(good) binomial summation can never converge to 100% implying that democracy is deficient. This non-constructively shows the impossibility without even knowing individual voter error and series summation. Interestingly, this divergence condition coincides with contradiction mentioned in 53.16.3.1 for PARITY3SAT superpolynomial size circuits. Assuming an NC/Poly or P/Poly LHS Percolation Boolean Function with 100% noise stability in P(Good) LHS, Multiway Election in RHS can be placed in hardest known complexity classes - Polynomial Hierarchy PH, EXPSPACE - *) for PH it is expoential DC-uniform circuit size required by RHS *) for adversarial simulation by Chess,Go etc., it is EXPTIME-hard required by RHS etc.,. This fits into scenario 53.9.5 when LHS is 100% stable circuit and RHS is unstable PH=DC circuit.

Reference:

256.1. Elections can be manipulated often - [Friedgut-Kalai-Nisan] - http://www.cs.huji.ac.il/~noam/apx-gs.pdf

256.2. Computational Hardness of Election Manipulation - https://www.illc.uva.nl/Research/Publications/Reports/MoL-2015-12.text.pdf

---
257. Commits as on 4 April 2016

- Updated AsFer Design Document - KKL inequality and Denoisification
- Spark-Cython intrinsic merit computation has been md5hash enabled again since there were key errors with large strings
- logs and screenshots have been committed to testlogs/
- rebuilt Cython .c, .so files
- Spidered text has been changed - This took 28 minutes for ~40+ keywords better than 6 minutes for ~10 keywords earlier. Again Spark in single node cluster dualcore has some bottleneck.

---
258. (FEATURE-BENCHMARK-DONE) Analysis of PySpark-Cython Intrinsic Merit Computation Benchmarks done so far
---
From 191 and 201 time complexity analysis of Recursive Gloss Overlap algorithm, for single node dualcore cluster following is the
runtime bound:

$$O(n*d*s^{tmax}/cpus)$$

where d is the number of keywords, n is the number of documents and s is the maximum gloss size per keyword. For last two benchmarks done with
MD5 hashkeys, Local Cacheing and Global Cacheing(MemCached) enabled number of words are 10 and 40. Corresponding runtimes are:
---
10 words
---
n=1
d=10
s=constant
tmax=2 (depth 2 recursion)
cpus=2 (dual core)
Runtime = 7 minutes = $k*10*(s)^2/2$
---
40 words
---
n=1
d=40
s=constant
tmax=2 (depth 2 recursion)
cpus=2 (dual core)
Runtime = 28 minutes = $k*40*(s)^2/2$
Ratio is 7/28 = $k*10*(s)^2/2/k*40*(s)^2/2$ = 0.25 a linear scaling in number of words which substantiates the theoretical bound. If cluster scales on number of words and documents (e.g logd and logn) this should have non-linear asymptotic scaling. Maximum number of words per document can be an assumption and set to a harcoded large number. If number of web documents versus number of keywords follow a Gaussian Normal distribution i.e a small fraction of documents have high number of words while the tails have less number of keywords, cluster prescales on a function of mean which is apriori known by a heuristic. Above excludes the crawling time per webpage - crawling does not build linkgraph.

---
259. (THEORY) Major update to Computational Geometric PRAM-NC algorithm for Discrete Hyperbolic Factorization
---

PRAM-NC Algorithm for Discrete Hyperbolic Factorization in 34 above:

34.1 LaTeX - http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.tex/download
34.2 PDF - http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.pdf/download

internally relies on [BerkmanSchieberVishkin] and other All Nearest Smaller Values PRAM algorithms for merging sorted tesselated hyperbolic arc segments which is then binary searched to find a factor. There are some older alternatives to merging sorted lists other than All Nearest Smaller Values which require more processors. Section 2.4 in [RichardKarp-VijayaRamachandran] on "Sorting, Merging and Selection" in PRAM where input is an array of O(N=n) elements, describes some standard PRAM algorithms, gist of which is mentioned below (https://www.eecs.berkeley.edu/Pubs/TechRpts/1988/5865.html):

259.1) There are 3 parallel computing models - Parallel Comparisons, Comparator Sorting Networks and PRAMs
259.2) Section 2.4.1 on merging two increasing sequences of length n and m, n <= m in Page 19 gives an algorithm of O(loglogN) steps on O(n+m) CREW PRAM processors. This constructs a merge tree bottom-up to merge two sorted lists. For merging more than 2 lists, requires logarithmic additional parallel steps.
259.3) Section 2.4.2 describes Batcher's and [AjtaiKomlosSzmeredi] Bitonic Sort algorithm which sorts n elements in O((logn)^2) time with n/2 processors.
259.4) Section 2.4.3 describes Cole's PRAM sorting algorithm which requires O(logn) steps and O(n) PRAM processors.

Advantage of these older algorithms is they are implementable e.g Bitonic sort - https://en.wikipedia.org/wiki/Bitonic_sorter, http://www.nist.gov/dads/HTML/bitonicSort.html. Replacing [BerkmanSchieberVishkin] in ANSV sorted tile merge for discrete hyperbolic factorization with older algorithms like bitonic parallel sorting networks would require more processors but still be in NC though not PRAM.

References:

259.1 Bitonic Sorting - https://cseweb.ucsd.edu/classes/wi13/cse160-a/Lectures/Lec14.pdf

---
260. Commits as on 10 April 2016

Complement Function script updated with a new function to print Ihara Identity zero imaginary parts mentioned in
Complement Function extended draft:
1. https://sourceforge.net/p/asfer/code/HEAD/tree/asfer-docs/AstroInferDesign.txt
2. https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

Logs for this included in python-src/testlogs/

---
261. Commits (1) as on 12 April 2016

(FEATURE-DONE) Implementation for NC Discrete Hyperbolic Factorization with Bitonic Sort alternative (in lieu of unavailable PRAM implementations):

Bitonic Merge Sort Implementation of:

- http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.tex/download
- http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.pdf/download

- Bitonic Sort is O((logn)^2) and theoretically in NC
- Bitonic Sort requires O(n^2logn) parallel comparators. Presently this parallelism is limited to parallel
exchange of variables in bitonic sequence for which python has builtin support (bitonic_compare()) - internally how it performs on
multicore archs is not known.
- a shell script has been added which does the following (cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound_Bitonic.sh):
    - NC Factorization implementation with Bitonic Sort is C+++Python (C++ and Python)
    - C++ side creates an unsorted mergedtiles array for complete tesselated hyperbolic arc
(cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound_Bitonic.cpp)
    - This mergedtiles is captured via output redirection and awk in a text file
cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound_Bitonic.mergedtiles
    - Python bitonic sort implementation which is a modified version of algorithm in https://en.wikipedia.org/wiki/Bitonic_sorter
requires the size of the array to be sorted in exponents of 2. Presently it is hardcoded as array of size 16384 - 2^16
(python-src/DiscreteHyperbolicFactorizationUpperbound_Bitonic.py). It outputs the sorted array. Vacant elements in array are zero-filled

- Binary and logs for C++ and Python side have been committed
- Parallel comparators on large scale require huge cloud of machines per above bound which makes it NC.

Commits (2) as on 12 April 2016

- Compare step in Bitonic Sort for merged tiles has been Spark MapReduced with this separate script -
../python-src/DiscreteHyperbolicFactorizationUpperbound_Bitonic_Spark.py. Can be used only in a large cluster. With this comparators are highly parallelizable over a cloud and thus in NC.

References:

261.1 Parallel Bitonic Sort - http://www.cs.utexas.edu/users/plaxton/c/337/05s/slides/ParallelRecursion-1.pdf

---
262. Commits as on 13 April 2016

Spark Bitonic Sort - Imported BiDirectional Map - bidict - for reverse lookup (for a value, lookup a key in addition to for a key,
lookup a value) which is required to avoid the looping through of all keys to match against a tile element.

---

263. Commits as on 27 April 2016

  Interim Commits for ongoing investigation of a race condition in Spark MapReduce for NC Factorization with Bitonic Sort :

- The sequential version of bitonic sort has been updated to do away with usage of boolean flag up by 2 compare functions for True and False
- Spark NC implementation of Bitonic Sort for Factorization still has some strange behaviour in sorting. In progress commits for this do following in Compare-And-Exchange phase:
        - Comparator code has been rewritten to just do comparison of parallelized RDD set of tuples where each tuple is of the form:
                (i, i+midpoint)
        - This comparison returns a set of boolean flags collect()ed in compare()
        - The variable exchange is done sequentially at each local node.
        - This is because Spark documentation advises against changing global state in worker nodes (though there are exceptions in accumulator)
        - There were some unusual JVM out of memory crashes, logs for which have been committed
        - some trivial null checks in complement.py
        - Bitonic Sort presents one of the most non-trivial cases for parallelism - parallel variable compare-and-exchange to be specific and present commits are a result of few weeks of tweaks and trial-errors.
        - Still the race condition is observed in merge which would require further commits.

---

264. Commits as on 28 April 2016

Interim commits 2 for ongoing investigation of race conditions in Spark Bitonic Sort:

        - bidict usage has been removed because it requires bidirectional uniqueness, there are better alternatives
        - The variable exchange code has been rewritten in a less complicated way and just mimicks what sequential version does in CompareAndExchange phase.
        - logs for Sequential version has been committed
        - number to factor has been set to an example low value
        - .coordinates and .mergedtiles files have been regenerated
        - Still the race condition remains despite being behaviourally similar to sequential version except the parallel comparator array returned by mapreduce
        - Parallel comparator preserves immutability in local worker nodes

---

265. (THEORY) UGC(Unique Games Conjecture) and Special Case of Percolation-Majority Voting Circuits

---

265.1 Unique Games Conjecture by [SubhashKhot] is a conjecture about hardness of

approximating certain constraint satisfaction problems. It states that complexity of obtaining polynomial time approximation schemes for specific class of constraint satisfaction NP-hard problems is NP-hard itself i.e polytime approximation is NP-hard. From section 255 above, if in a special case percolation with sampling of left-right points on percolation grids results in logarithmic advice, then percolation is in NC/log. Thus if RHS has 100% stable NP-hard voting function then NP is in NC/log implying NP is in P/log and P=NP. This disproves UGC trivially because all NP problems have P algorithms. But this disproof depends on strong assumptions that percolation has NC/log circuit and RHS of Majority voting is 100% noise stable. There are already counterexamples showing divergence of RHS in sections 53.16.3.1 and 256.
265.2 Voter Boolean Functions can be written as :
	265.2.1 Integer Linear Programs : Voter Judging Boolean Functions are translated into an Integer Linear Program, binary to be precise because value for variable and final vote is 0, 1 and -1(abstention). This readily makes any decision process by individual voter to be NP-hard.
	265.2.2 and Constraint Satisfaction Problems : Voter expects significant percentage of constraints involving variables to be satisfied while voting for a candidate which brings the majority voting problem into the UGC regime. Approximating the result of an Election mentioned in Section 14 is thus reducible to hardness of approximation of constraint satisfaction and could be NP-hard if UGC is true.
265.3 Thus RHS of Majority Voting Circuit becomes a functional composition behemoth of indefinite number of individual voter constraint and integer linear programs. There is a generic concept of sensitivity of a non-boolean function mentioned in [Williamson-Schmoys] which can be applied to sensitivity and stability of a voter integer linear program and constraint program.
265.4 Approximating outcome of a majority voting (e.g. real life pre/post poll surveys) depends thus on following factors:
	265.4.1 Correctness - Ratio preserving sample - How reflective the sample is to real voting pattern e.g a 60%-40% vote division in reality should be reflected by the sample.
	265.4.2 Hardness - Approximation of Constraint Satisfaction Problems and UGC.

References:

265.4 Unique Games Conjecture and Label Cover - https://en.wikipedia.org/wiki/Unique_games_conjecture
265.5 Design of Approximation Algorithms - Page 442 - http://www.designofapproxalgs.com/book.pdf

---
266. Commits 1 as on 29 April 2016

Interim commits 3 for Bitonic Sort Spark NC Factorization Implementation
---
- AsFer Design Document updated for already implemented NVIDIA CUDA Parallel C reference code for bitonic sort on hypercube
- Bitonic Sort C++ code updated for max size of sort array - reduced from 16384 to 256
- Bitonic Sort Python Spark implementation updated for returning a tuple containing index info also instead of plain boolean value
because Spark collect() after a comparator map() does not preserve order and index information was lost.
- if clauses in exchange phase have been appropriately changed
- Still there are random mysterious jumbled tuples which will be looked into in future commits
- This implementation becomes redundant because of already existing better NVIDIA CUDA Parallel C implementation and
is of academic interest only.

Final commits for Bitonic Sort Spark NC Factorization Implementation

---
- Indeed Bitonic Sort from previous commit itself works without any issues.
- Nothing new in this except reduced sort array of size just 64 for circumventing frequent Py4Java Out of Memory errors and for quick runtime
- Logs for this have been committed in testlogs/
- the debug line "merged sorted halves" is the final sorted tiles array
- .coordinates and .mergedtiles files have been updated

---
267. Commits as on 30 April 2016

- Printed return value of bitonic_sort() in sorted. Earlier it was wrongly printing globalmergedtiles a global state which is not changed anymore
- logs for bitonic sort of tesselated hyperbolic arc

---
268. Commits as on 1 May 2016

- The Map step now does both Compare and Exchange within local worker nodes for each closure and result is returned in tuples
- The driver collect()s and just assigns the variables based on comparator boolean field in tuple which are already exchanged within local worker

---
269. Commits as on 2 May 2016

Accumulator support for Bitonic Sort mapreduce globalmergedtiles
---
- Spark context has been parametrized and is passed around in the recursion so that single spark context keeps track of the accumulator variables
- globalmergedtiles_accum is the new accumulator variable which is created in driver and used for storing distributed mutable globalmergedtiles
- AccumulatorParam has been overridden for Vectors as globals
- Spark does not permit however, at present, mutable global state within worker tasks and only driver creates the accumulator variables which can only be incrementally extended within closure workers.
- muting or accessing accumulator.value within worker raises Exception: Can't access value of accumulator in worker tasks
- Logs for above have been committed to testlogs/
- Because of above limitation of Spark in global mutability, accumulator globalmergedtiles is updated only outside the Spark mapreduce closures.
- Support for global mutables, if Spark has it, would make compare-and-exchange phase closer in behaviour to an NVIDIA Parallel bitonic sort.
- Documentation on this necessary feature is scarce and some suggest use of datastores like Tachyon/Alluxio for distributed shared mutables which is non-trivial. This cloud parallel sort implementation achieves thus almost ~90% of NVIDIA CUDA parallel bitonic sort - CompareAndExchange is done in parallel but assigning CompareAndExchange results is done sequentially.

---
270. Commits as on 3 May 2016

New threading function assign_compareexchange_multithreaded() has been implemented which is invoked in lieu of sequential for loop assignment , by creating a thread for

each pair of (i,i+midpoint) compare-exchange assignment
of accumulator globalmergedtiles.This thread function assigns the Spark
Mapreduce result of Compare and Exchange.

Mulithreading is an alternative for global state mutability in the absence of
Spark support
and it does not require any third party in-memory cacheing products.

Coordinates are also shuffled first few elements of which correspond to number
to factor N - these are
the factors found finally. Multithreaded assignment maps to a multicore
parallelism. Logs for this have been
committed to testlogs with and without Coordinates Shuffling.

---
271. Commits 2 as on 3 May 2016

Number to factorize has been changed to 147. The mergedtiles array has been set
to size 256. The logs for this have been committed in testlogs/. The factors
are printed in last few lines of the logs in shuffled globalcoordinates
corresponding to 147 in sorted globalmergedtiles accumulator:

sorted= [147, 147, 147, 147, 147, 147, 147, 147, 147, 147, 147, 147, 147, 146,
146, 146, 146, 146, 146, 146, 146, 145, 145, 145, 145, 145, 145, 144, 144, 144, 144,
144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
144, 144, 144, 144, 144, 144, 144, 144, 144, 143, 143, 143, 143, 143, 142, 142,
142, 142, 141, 141, 141, 141, 141, 140, 140, 140, 140, 140, 140, 140, 140, 140,
140, 140, 140, 140, 140, 140, 140, 140, 139, 138, 138, 138, 138, 138, 138, 138,
138, 138, 137, 136, 136, 136, 136, 136, 136, 136, 136, 135, 135, 135, 135, 135,
135, 135, 135, 135, 135, 134, 134, 133, 133, 133, 132, 132, 132, 132, 132, 132, 132,
132, 132, 132, 132, 132, 131, 130, 130, 130, 130, 130, 130, 129, 129, 129, 129,
128, 128, 128, 128, 128, 128, 127, 126, 126, 126, 126, 126, 126, 126, 126, 126,
126, 125, 125, 125, 124, 124, 124, 124, 123, 123, 123, 122, 122, 121, 120, 120,
120, 120, 120, 120, 120, 120, 120, 119, 118, 118, 117, 117, 117, 116, 116, 116, 116,
115, 114, 114, 114, 114, 113, 112, 112, 111, 111, 111, 110, 110, 109, 108, 108,
108, 108, 107, 106, 106, 105, 104, 104, 103, 102, 102, 101, 100, 100, 99, 98,
98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79,
79, 78, 78, 77, 77, 76, 76, 75, 75, 74, 74, 73, 0, None]
globalcoordinates= [1, 21, 21, 49, 49, 7, 147, 147, 3, 3, 3, 3, 7, 73, 73, 147,
2, 2, 145, 2, 144, 5, 5, 29, 29, 5, 24, 16, 16, 9, 8, 24, 18, 18, 6, 6, 6, 143,
2, 2, 3, 4, 4, 4, 3, 3, 4, 8, 71, 12, 36, 47, 9, 36, 12, 142, 13, 13, 11, 11, 2,
70, 141, 2, 3, 3, 140, 3, 46, 34, 14, 7, 139, 2, 27, 14, 10, 10, 69, 19, 5, 7,
4, 4, 4, 5, 138, 45, 22, 68, 3, 137, 2, 6, 6, 3, 136, 2, 33, 16, 67, 135, 4, 4,
8, 3, 3, 5, 134, 5, 9, 14, 26, 44, 66, 133, 7, 132, 18, 65, 4, 3, 3, 4, 6, 43,
11, 131, 32, 21, 10, 130, 129, 5, 12, 25, 9, 64, 3, 3, 42, 128, 7, 127, 4, 31,
15, 63, 126, 8, 41, 13, 62, 6, 3, 5, 125, 17, 20, 24, 5, 124, 61, 30, 123, 4, 3,
122, 40, 60, 121, 120, 3, 4, 59, 39, 23, 29, 119, 4, 118, 117, 58, 38, 3, 116,
115, 3, 28, 57, 114, 113, 37, 3, 56, 112, 55, 111, 110, 3, 36, 109, 54, 108, 53,
107, 2, 35, 106, 52, 105, 104, 51, 103, 102, 101, 50, 100, 49, 99, 98, 48, 97,
96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 1, 78,
77, 1, 76, 1, 1, 75, 74, 1, 73, 1, 72, 146, None]
sorted globalmergedtiles accumulator version: = [147, 147, 147, 147, 147, 147,
147, 147, 147, 147, 147, 147, 147, 146, 146, 146, 146, 146, 146, 146, 145, 145,
145, 145, 145, 145, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
144, 143, 143, 143, 143, 143, 142, 142, 142, 142, 141, 141, 141, 141, 141, 140,
140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140,
139, 138, 138, 138, 138, 138, 138, 138, 138, 138, 137, 136, 136, 136, 136, 136,
136, 136, 136, 136, 135, 135, 135, 135, 135, 135, 135, 135, 135, 134, 134, 133, 133,
133, 132, 132, 132, 132, 132, 132, 132, 132, 132, 132, 132, 132, 131, 130, 130,
130, 130, 130, 130, 130, 129, 129, 129, 129, 128, 128, 128, 128, 128, 128, 127, 126,
126, 126, 126, 126, 126, 126, 126, 126, 126, 125, 125, 125, 124, 124, 124, 124,

123, 123, 123, 122, 122, 121, 120, 120, 120, 120, 120, 120, 120, 120, 119, 118, 118, 117, 117, 117, 116, 116, 116, 116, 115, 114, 114, 114, 114, 113, 112, 112, 111, 111, 111, 110, 110, 109, 108, 108, 108, 108, 107, 106, 106, 105, 104, 104, 103, 102, 102, 101, 100, 100, 99, 98, 98, 97, 96, 95, 94, 93, 92, 91, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 79, 78, 78, 77, 77, 76, 76, 75, 75, 74, 74, 73, 0, None]
16/05/03 18:33:06 INFO SparkUI: Stopped Spark web UI at http://192.168.1.101:4040
16/05/03 18:33:06 INFO DAGScheduler: Stopping DAGScheduler
16/05/03 18:33:06 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!

Factors of 147 are [1,21,49,7,147,3] from globalcoordinates and globalmergedtiles(corresponding to 147 elements) printed previously in logs.

---
272. (THEORY) Integer Partitions, Riemann Sums, Multipartisan Majority Voting - a special case
---
From https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions_2014.pdf, Integer Partitions when visualized as area under a curve in discrete sense, correspond to Riemann Sums Discrete Area Integral i.e Each part in the partition slices the area under a curve formed by topmost points of each part in partition. This curve can be perturbed to arbitrary shape without altering the area underneath it - in other words each perturbed curve corresponds to a partition of N where N is the area integral value for the curve. A striking analogy: a vessel containing water is perturbed and the surface of the liquid takes arbitrary curvature shapes but the volume of the liquid remains unchanged (assuming there is no spillover) - has some similarities to perturbation of spacetime Riemannian Manifolds due to gravity in General Relativity. This analogy simulates a 3-dimensional partition number (as against the usual 2-dimensional) of liquid of volume V. Let p3(V) denote the 3-dimensional partition of a liquid of volume V. It makes sense to ask what is the bound for p3(V). For example a snapshot of the curvature of liquid surface be the function f. A slight perturbation changes this function to f' preserving the volume. Hence f and f' are 2 different partitions of the V - the difference is these are continuous partitions and not the usual discrete integer partitions with distinct parts. Number of such perturbations are infinite - there are infinitely many f=f'=f''=f'''=f''''=f''''' .... = V and hence p3(V) is infinite (proof is by diagonalization: an arbitrary point of the curve on the continuum can be changed and this recursively continues indefinitely). As any perturbation can be linked to a (pseudo)random source, p3(V) is equivalent to number of pseudorandom permutations (randomness extracted from some entropy source). p3(V) can be extended to arbitrary dimensions n as pn(V). This generalized hash functions and also majority voting to continuous setting - though it is to be defined what it implies by voting in continuous population visavis discrete. In essence, Multipartisan Majority Voting = Hash Functions = Partitions in both discrete and continuous scenarios.

---
273. Commits as on 4 May 2016

Sequence Mining of Prime Numbers as binary strings

- First 10000 prime numbers have been written to a text file in binary notation in complement.py
- SequenceMining.py mines for most prominent sequences within prime binary

strings - a measure of patterns in prime distribution
- Logs for this have been committed to testlogs/

---
274. Commits as on 11 May 2016

Sequence Mining in Prime binary strings has been made sophisticated:
	- Prints each binary string sequence pattern in decimals upto maximum of 15-bit sequences mined from first 10000 primes.
	- This decimals are written to a file PatternInFirst10000Primes.txt
	- Approx.py R+rpy2 has been invoked to print an R function plotter of this decimal pattern in prime binary string sequences
	- The function plots the sequences in Y-axis and length of sequences in asscending orider in X-axis(with decreasing support)
	- As can be seen from pdf plot, the sequences show dips amongst peaks periodically.

---
275. (THEORY) Non-boolean Social Choice Functions Satisfiability, Sensitivity, Stability and Multipartisan Majority Voting - 256 continued
---
Let f be a voter decision function which takes in x1,x2,...xm as decision variables and outputs a vote for candidate 1 <= c <= n. This generalizes voter boolean judging functions for 2 candidates to arbitrarily large number of contestants. Assuming f to be continuous function and parameters to be continuous (as opposed to discrete binary string SAT assignments to variables in boolean setting), this generalizes Satisfiability problem to continuous, multivalued and non-boolean functions. This allows the candidate index to be continuous too. The function plot of this (candidate index versus decision string) would resemble any continuous mathematical function.
Stability which is: Pr[f(s) == f(t)] - Pr[f(s) != f(t)] for randomly correlated s=(x11,x12,...,x1m) and t=(x21,x22,x23,...,x2m) where correlation (measured by usual euclidean distance between s and t) could be uncomputable value in terms of mean but can be approximated as difference in ratio of length integral of function arc segments with high curvature and low curvature (where function has huge fluctuations and relatively flat). Sensitivity follows accordingly. This requires first derivative of the function to find out local maxima and minima.
From Gibbard-Satterthwaite theorem any social choice function for more than 3 candidates has a non-zero probability of being manipulated though negligible in practice. Probably this implies atleast indirectly that the stability of any non-boolean social choice function < 1 if number of candidates > 3 hinting at divergence of Pr(Good) Binomial series. This function plot should not be confused with function plot mentioned in 272 for votes partitioned across candidates which reduces to Riemann Sum. Unlike boolean sensitivity/stability, non-boolean counterparts can only be approximated and exact computation might never halt being undecidable. Non-boolean Social Choice Function generalizes satisfiability because an appropriate assignment to (x1,x2,x3,.....,xm) has to be found so that f(x1,x2,x3,.....,xm)=c where 1 <= c <= n.
This is precisely curve-line intersection problem - curve f(x1,x2,x3,...,xn) and line y=c - in computational geometry. Intersecting points (a1,a2,a3,...,az) are the satisfying assignments which choose candidate c. Contrasting this with boolean kSAT which is NP-complete, non-boolean kSAT has polynomial time computational geometric algorithms to find satisfying intersection points. It is intriguing to observe a sharp threshold phenomenon in computational hardness of satisfiability - an easy polytime non-boolean kSAT becomes NP-hard boolean kSAT.
Assuming that real-life voters have non-boolean decision functions only, the RHS

circuit of Pr(Good) majority voting is confined to polytime satisfiability realm alone. Non-boolean voting decision functions can be specialized for 2 candidates special case too - this allows fractional values to decision variables. This is plausible because Linear Programming (non-boolean) is polytime while 0-1 Integer Linear Programming (boolean) is NP-complete. Because of Non-boolean Social Choice functions (e.g Linear Programs with Constraints which each voter solves) being the most generic which allow fractional values to variables, any of the standard polytime Simplex algorithms - Dantzig Pivot Tableau and Karmarkar Interior Point Method - can be applied to find a satisfying assignment. Voter decides to vote for/againt when the solution to LP is above/below a threshold. This is a far from real,hypothetical,perfectly rational election setting. Real life elections have bounded rationality. It is an alternative to Computational Geometric intersection detection previously mentioned.

References:

275.1 Intersection detection - https://www.cs.umd.edu/~mount/Papers/crc04-intersect.pdf
275.2 Transversal Intersection of two curves - Newton iteration and Cramer's rule - https://en.wikipedia.org/wiki/Intersection_(Euclidean_geometry)

---
276. (FEATURE-DONE) Commits as on 30 May 2016 - Continued from 220

VIRGO CloudFS system calls have been added (invoked by unique number from syscall_32.tbl) for C++ Boost::Python interface to VIRGO Linux System Calls. Switch clause with a boolean flag has been introduced to select either VIRGO memory or filesystem calls.
kern.log and CloudFS textfile Logs for VIRGO memory and filesystem invocations from AsFer python have been committed to testlogs/

---
277. (FEATURE-DONE) Commits as on 31 May 2016 - Continued from 220 and 276

Python CAPI interface to NEURONRAIN VIRGO Linux System Calls has been updated to include File System open, read, write primitives also.
Rebuilt extension binaries, kern.logs and example appended text file have been committed to testlogs/. This is exactly similar to commits done for Boost::Python C++ interface. Switch clause has been added to select memory or filesystem VIRGO syscalls.

---
278. (FEATURE-DONE) Commits as on 7 June 2016

- getopts implementation for commandline args parsing has been introduced in Maitreya textclient rule search python script.
- an example logs for possible high precipitation longitude/latitudes in future dates - July 2016 - predicted by sequence mining learnt rules from past data has been added to testlogs/

```
root@shrinivaasanka-Inspiron-1545:/home/shrinivaasanka/Maitreya7_GitHub/martin-pe/maitreya7/releases/download/v7.1.1/maitreya-7.1.1/src/jyotish# python MaitreyaEncHoro_RuleSearch.py  --min_year=2016 --min_month=7 --min_days=1 --min_hours=10 --min_minutes=10 --min_seconds=10 --min_long=77 --min_lat=07 --max_year=2016 --max_month=7 --max_days=15 --max_hours=10 --max_minutes=10 --max_seconds=10 --max_long=78 --max_lat=10 |grep " a Class Association"
{  --date="2016-7-2 10:10:10 5"  --location=" x 77:0:0 7:0:0 " --planet-list } -
```

There is a Class Association Rule match [['Mercury', 'Venus']] in sign Gemini
{ --date="2016-7-3 10:10:10 5" --location=" x 77:0:0 7:0:0 " --planet-list } -
There is a Class Association Rule match [['Mercury', 'Venus']] in sign Gemini
{ --date="2016-7-4 10:10:10 5" --location=" x 77:0:0 7:0:0 " --planet-list } -
There is a Class Association Rule match [['Mercury', 'Venus']] in sign Gemini
{ --date="2016-7-5 10:10:10 5" --location=" x 77:0:0 7:0:0 " --planet-list } -
There is a Class Association Rule match [['Mercury', 'Venus']] in sign Gemini
{ --date="2016-7-6 10:10:10 5" --location=" x 77:0:0 7:0:0 " --planet-list } -
There is a Class Association Rule match [['Mercury', 'Venus']] in sign Gemini
{ --date="2016-7-11 10:10:10 5" --location=" x 77:0:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Cancer
{ --date="2016-7-13 10:10:10 5" --location=" x 77:0:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Cancer
{ --date="2016-7-14 10:10:10 5" --location=" x 77:0:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Cancer

---
279. (FEATURE-DONE) Commits - 23 June 2016 - Recurrent Neural Network Long Term Short Term Memory - Deep Learning - Implementation
---

A very minimal python implementation of LSTM RNN based on Schmidhuber-Hochreiter LSTM has been added to
already existing AsFer algorithms repertoire. LSTM RNN has gained traction in recent years in its variety of
applications in NLP, Speech Recognition, Image Pattern Recognition etc., The logs for this implementation show
a convergence of final output layer in 25th iteration itself (out of 10000). Learning the weights requires
other algorithms like Gradient Descent, Backpropagation (and there are already known limitations in weight learning with these)

---
280. (FEATURE-DONE) Commits - 23 June 2016 - Minimal Reinforcement Learning (Monte Carlo Action Policy Search) implementation
---

Reinforcement Learning has been implemented which changes state based on environmental observation and an appropriate policy is
chosen for observation by Monte Carlo Random Policy Search based on which rewards for each transitions are accumulated and output in the end.
Logs for 3 consecutive executions of Reinforcement Learning have been committed with differing total rewards gained.
Input observations are read from text file ReinforcementLearning.input.txt.

Reference: Richard Sutton - http://webdocs.cs.ualberta.ca/~sutton/papers/Sutton-PhD-thesis.pdf

---
---
281. (FEATURE-DONE) Commits - 24 June 2016 - ThoughtNet Reinforcement Learning implementation
---

ThoughtNet based Reinforcement Learning Evocation has been experimentally added as a clause in python-src/DeepLearning_ReinforcementLearningMonteCarlo.py. This is just to demonstrate how ThoughtNet hypergraph based evocation is supposed to work. python-src/ReinforcementLearning.input.txt has been updated to have a meaningful textual excerpt. Logs for this evocative model has been committed to

python-src/testlogs/DeepLearning_ReinforcementLearningMonteCarlo.ThoughtNet.out.24June2016

---

282. (FEATURE-DONE) Commits - 26 June 2016 - ThoughtNet File System Storage

---

ThoughtNet text files have been stored into a filesystem backend. It is read and eval()-ed into list of edges and hypergraphs.
Separate ThoughtNet directory has been created for these text files.

---

283. (THEORY) ThoughtNet growth and Evocation Reinforcement Learning algorithm (not feasible to implement exactly):

---

loop_forever
{
When an observation stimulus from environment arrives:
(*) Sensory instruments that simulate eye, ear, nose, etc., receive stimuli from environment (thoughts, music, noise, sound etc.,)
and convert to sentential form - this is of the order of billions - and appended to list in ThoughtNet_Edges.txt
(*) Sentences are classified into categories (for example, by maximum core numbers in definition wordnet subgraph) which is also order of billions and added as hypergraph edges in dict with in ThoughtNet_Hypergraph.txt - hyperedge because same document can be in more than 2 classes. This creates a list for each key in dict and the list has to be sorted based on evocation potential - reward for reinforcement learning. There is no need for monte carlo and dynamic programming if pre-sorted because sum of rewards is always maximum - only topmost evocative edge is chosen in each list of a key.
(*) observation is split to atomic units - tokenized - or classified and each unit is lookedup in ThoughtNet_Hypergraph.txt to get a list and hyperedge with maximum potential is returned as evocative with optional lambda evaluation which is the action side of reinforcement learning.
}

---

284. (FEATURE-DONE) Auto Regression Moving Average - ARMA - Time Series Analysis for Stock Ticker Stream

---

Commits - 27 June 2016

---

1. Time Series Analysis with AutoRegressionMovingAverage of Stock Quote streamed data has been implemented (R function not invoked). Logs which
show the actual data and projected quotes by ARMA for few iterations have been committed to testlogs. This ARMA projection has been plotted in R to a pdf file which is also committed.
2. ARMA code implements a very basic regression+moving averages. Equation used is same though not in usual ARMA format
(1-sigma())X(t) = (1+sigma())
3. Also committed is the R plot for SequenceMined Pattern in first 10000 prime numbers in binary format (DJIA approx and approxfun plots have been regenerated)

---

285. (FEATURE-DONE) Commits 1 - 28 June 2016 - Neo4j ThoughtNet Hypergraph Database Creation

---

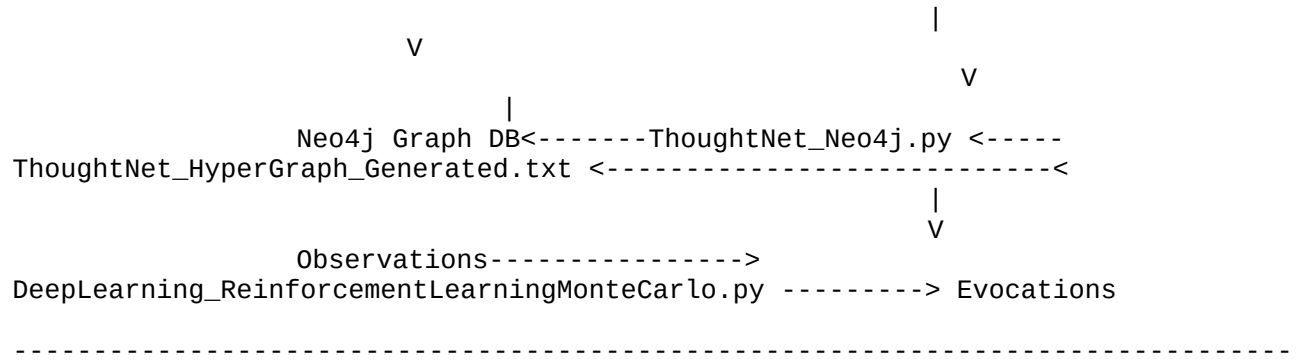
1.New file ThoughtNet_Neo4j.py has been added to repository which reads the ThoughtNet edges and hypergraph text files and uploads them into Neo4j Graph Database through py2neo client for Neo4j.
2.Neo4j being a NoSQL graph database assists in querying the thoughtnet and scalable.
3.ThoughtNet text files have been updated with few more edges and logs for how Neo4j graph database for ThoughtNet looks like have been committed to testlogs/

---
286. (FEATURE-DONE) Commits 2 - 28 June 2016 - ThoughtNet Neo4j Transactional graph creation
---
1.transactional begin() and commit() for graph node and edges creation has been included.
2.This requires disabling bolt and neokit disable-auth due to an auth failure config issue.
3.Logs and a screenshot for the ThoughtNet Hypergraph created in GUI (http://localhost:7474) have been committed to testlogs/

---
287. (FEATURE-DONE) Commits - 29 June 2016 - ThoughtNet and Reinforcement Deep Learning
---
Major commits for ThoughtNet Hypergraph Construction and Reinforcement Learning from it
---
1. python-src/DeepLearning_ReinforcementLearningMonteCarlo.py reads from automatically generated ThoughtNet Hypergraph text backend created by python-src/ThoughtNet/Create_ThoughtNet_Hypergraph.py in python-src/ThoughtNet/ThoughtNet_Hypergraph_Generated.txt
2. Separate Recursive Gloss Overlap CoreNumber and PageRank based Unsupervised Classifier has been implemented in python-src/RecursiveGlossOverlap_Classifier.py which takes any text as input arg and returns the classes it belongs to without any training data. This script is imported in python-src/ThoughtNet/Create_ThoughtNet_Hypergraph.py to automatically classify observations from environment and to build ThoughtNet based on them for evocations later.
3. python-src/ThoughtNet/ThoughtNet_Neo4j.py reads from automatically generated ThoughtNet in python-src/ThoughtNet/ThoughtNet_Hypergraph_Generated.txt
4. Logs for above have been committed to respective testlogs/ directories
5. Compared to human evaluated ThoughtNet Hypergraphs in python-src/ThoughtNet/ThoughtNet_Hypergraph.txt, generated python-src/ThoughtNet/ThoughtNet_Hypergraph_Generated.txt does quite a deep learning from WordNet mining and evocations based on this automated ThoughtNet are reasonably accurate.
6. python-src/ReinforcementLearning.input.txt and python-src/ThoughtNet/ThoughtNet_Edges.txt have been updated manually.

---
288. (FEATURE-DONE) Schematic Diagram for ThoughtNet Reinforcement Evocation - approximate implementation of 283

Environment -------(stimuli)------> Sensors -----> Create_ThoughtNet_Hypergraph.py ----> RecursiveGlossOverlap_Classifier.py ----|



---

289. (FEATURE-DONE) Commits - 30 June 2016 - Sentiment Scoring of ThoughtNet edges and Sorting per Class

---

1. Create_ThoughtNet_Hypergraph.py has been changed to do sentiment scoring which is a nett of positivity, negativity and objectivity per thought hyperedge and to sort the per-class key list of hyperedges descending based on sentiment scores computed.
2. python-src/ReinforcementLearning.input.txt which is the input observation stimulus and python-src/ThoughtNet/ThoughtNet_Edges.txt, the training data for construction of Hypergraph have been updated with additional text sentences.
3. Hypergraph created for this is committed at python-src/ThoughtNet/ThoughtNet_Hypergraph_Generated.txt with logs in python-src/ThoughtNet/testlogs/Create_ThoughtNet_Hypergraph.sentiment_sorted.out.30June2016. Example evocatives for python-src/ReinforcementLearning.input.txt input to python-src/DeepLearning_ReinforcementLearningMonteCarlo.py are logged in python-src/testlogs/DeepLearning_ReinforcementLearningMonteCarlo.ThoughtNet_sentiment_sorted.out.30June2016

This completes ThoughtNet Classifications based Evocative Inference implementation minimally.

---

290. (FEATURE-DONE) Commits - 1 July 2016 - Reinforcement Learning Bifurcation

---

Reinforcement Learning code has been bifurcated into two files than having in if-else clause, for facilitating future imports :-
one for MonteCarlo and the other for ThoughtNet Evocation. Logs for these two have been committed to testlogs.

---

291. (THEORY) Logical Time and ThoughNet (related to EventNet 70-79)

ThoughtNet Hypergraph is multiplanar and when seen from elevation it resembles billions of skyscraper category towers interconnected by Thought HyperEdges which are classified onto these categories. This gives an alternative route to define Time. Each tower represents chronologically stacked up events/occurrences/thoughts when unsorted i.e most recent thought is on top always and evocation based on chronology returns topmost thought. ThoughtNet evocation is an associative memory process returning most relevant thought in past for an observation in present. Two stack node towers with differing depths define two views of time i.e time is subjective to the category than objective across ThoughtNet. In a timeless universe, measuring duration between two events reduces to height of these unequal stacks which is not absolute - there are many logical "Time"(s). Each stack node has its own independent logical "Clock". This makes no assumption about relativistic theory of time and is independent of any physical law. From theory of computation standpoint, this is similar to a multi-

tape turing machine but with added feature in which tapes are interconnected. When the stacks are sorted for evocation potential based on some criterion (e.g sentiwordnet, EEG electric pulse etc.,) chronology information is lost to some extent but still the edge id retains it - most recent edge has largest edge numeric id. Yet, this assumes a distributed global unique id is achievable. When events occur in parallel, two edges geographically separate can have same edge id in the absence of special mechanisms for uniqueness. Vis-a-vis EventNet, ThoughtNet is not a causation graph. Relationship between EventNet and ThoughtNet is: Every partaker node in EventNet has a ThoughtNet. EventNet when topologically sorted has no absolute time - can have many orderings of events - similar to ThoughtNet - each thought edge can have varied logical timestamps when viewed from different category spectacles - in the absence of time. Following is the hierarchy: Each node in an EventNet is an event, and each event has partaker nodes which create graph by interaction among themselves, and each partaker node has an internal ThoughtNet. Thus there are 3 levels - EventNet is the biggest, Event is bigger and Partaker is big - EventNet is a graph of graph nodes of graph nodes. The tensor product of these nested graphs has 3 tiers - EventNet(Event(PartakerThoughtNet))) - denoted as EventNet (*) Event (*) PartakerThoughtNet. This tensor product captures both individual's view of event ordering and cosmic event ordering. ThoughtNet per node in an event is more of a projection of its observable universe - imaginary - while interaction among nodes in an event and causality between events are real. Measurement of duration between events in this tensor product can be done in various ways: 1) Distance between two event nodes causally connected 2) Global topological sorting of EventNet 3) From the ThoughtNet projection observed by each node in an event. This gives rise to multiple interpretations of time.

As a working example, following edges were evoked when word "economic" was uttered:

```
================================================================================
===========
Observation: economic
evocative thought (reward) returned(in descending order of evocation potential):
The HDI was created to emphasize that people and their capabilities should be
the ultimate criteria for assessing the development of a country, not economic
growth alone. The HDI can also be used to question national policy choices,
asking how two countries with the same level of GNI per capita can end up with
different human development outcomes. These contrasts can stimulate debate about
government policy priorities.
evocative thought (reward) returned(in descending order of evocation potential):
We need an SPI and we need to understand its value in our society because we
need to understand how we’re doing in terms of health and education and the
quality of our water
evocative thought (reward) returned(in descending order of evocation potential):
Social progress depends on the policy choices, investments, and implementation
capabilities of multiple stakeholders—government, civil society, and business.
================================================================================
===========
```

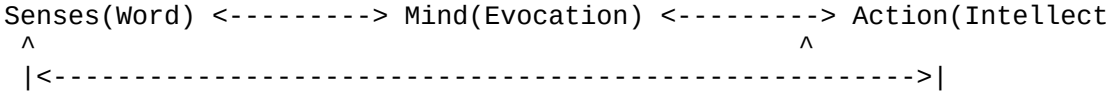
Only the top most evocative edge (sorted based on sentiment scores) has the word "economics" in it while the other two don't. But still the classification by definition graph core numbers inferred that these belong to "economic" class and pigeonholed them to stack "economic" - in other words the concept "economics" was deep-learnt by inner graph structure of the sentence without training data. But bottom two edges were most recent compared to the topmost and yet scored less on sentiment.

A practical ThoughtNet storage could be of billions of edges based on experiential learning of an individual over a period of lifetime and has to be suitably stored in a medium that mimicks brain. Ideally ThoughtNet has to be in some persistence bigdata storage though it still is just an approximation. Neo4j backend for ThoughtNet has been implemented in ThoughtNet/. ThoughtNet is kind of Evocation WordNet expanded for thoughts represented as sentences (because there is no better way to encode thoughts than in a natural language) and classified. Hypergraph encodes the edges as numbers - for example,

"transactions":[1] and "security":[1] implies that a sentence numbered 1 has been pre-classified under transactions and security categories. Also "services":[0,1] implies that there are two sentences encoded as 0 and 1 classified in services category with descending order of evocative potentials - 0 is more evocative than 1. In an advanced setting the ThoughtNet stores the lambda function composition parenthesized equivalent to the sentence and action taken upon evocation is to evaluate the most potent evocative lambda expression. On an evocative thought, state of "ThoughtNet" mind changes with corresponding action associated with that state (the usual mind-word-action triad). Philosophically, this simulates the following thought experiment:

- Word: Sensory receptors perceive stimuli - events
- Mind: stimuli evoke thoughts in past
- Action: Evocative thought is processed by intellect and inspires action - action is a lambda evaluation of a sentence.

##



and above is an infinite cycle. Previous schematic maps interestingly to Reinforcement Learning(Agent-Environment-Action-Reward).

##

In this aspect, ThoughtNet is a qualitative experimental inference model compared to quantitative Neural Networks.

It is assumed in present implementation that every thought edge is a state implicitly, and the action
for the state is the "meaning" inferred by recursive lambda evaluation (lambda composition tree evaluation for a natural
language sentence is not implemented separately because its reverse is already done through closure of a RGO graph in other
code in NeuronRain AsFer. Approximately every edge in Recursive Gloss Overlap wordnet subgraph is a lambda function with its
two vertices as operands which gives a more generic lambda graph composition of a text)

---

292. (THEORY) Recursive Lambda Function Growth Algorithm and Recursive Gloss Overlap Graph - 216 Continued

---

Lambda Function Recursive Growth algorithm for inferring meaning of natual language text approximately, mentioned in 216 relies
on creation of lambda function composition tree top-down by parsing the connectives and keywords of the text. Alternative to this top-down
parsing is to construct the lambda function composition graph instead from the Recursive Gloss Overlap WordNet subgraph itself where
each edge is a lambda function with two vertex endpoints as operands (mentioned in 291). Depth First Traversal of this graph iteratively
evaluates a lambda function f1 of an edge, f1 is applied to next edge vertices in traversal to return f2(f1), and so on upto function
fn(fn-1(fn-2(.....(f2(f1))...) which completes the recursive composition. Which one is better - tree or graph lambda function growth - depends
on the depth of learning necessary. Rationale in both is that the "meaning" is accumulated left-right and top-down on reading a text.

293.(FEATURE-DONE) Commits - 7 July 2016 - Experimental implementation of Recursive Lambda Function Growth Algorithm (216 and 292)

1.Each Sentence Text is converted to an AVL Balanced Tree (inorder traversal of this tree creates the original text)
2.Postorder traversal of this tree computes a postfix expression
3.Postfix expression is evaluated and a lambda function composition is generated by parenthesization denoting a function with two arguments.Logs
for this have been committed to testlogs/ which show the inorder and postorder traversal and the final lambda function grown from a text.
4.This is different from Part of Speech tree or a Context Free Grammar parse tree.
5.On an average every english sentence has a connective on every third word which is exactly what inorder traversal of AVL binary tree does.If not a general B+-Tree can be an ideal choice to translate a sentence to tree. Every subtree root in AVL tree is a lambda function with leaves as operands.

---
294. (FEATURE-DONE) Commits - 8 July 2016

Changed Postfix Evaluation to Infix evaluation in Lambda Function Growth with logs in testlogs/

---
295. (THEORY) Contextual Multi-Armed Bandits, Reinforcement Learning and ThoughtNet - related to 241, 291, 292 - 18 July 2016

Contextual Multi-Armed Bandits are the class of problems where a choice has to be made amongst k arms and each iteration fetches a reward. Choice of an arm in next iteration depends on rewards for previous iteration. Ideal examples are Recommender Systems, Contextual website advertisements etc., Thus an agent learns from past and policy action depends on rewards for previous actions. Translating this into ThoughtNet realm is fairly straightforward - Each class stack node in ThoughtNet is a multi-armed bandit and an evocative has to choose a hyperedge that 1) maximizes reward by fitting the context meaningfully and 2) learns from rewards for previous actions by virtue of ThoughtNet storage itself because ThoughtNet per partaker is built gradually by storing thought hyperedges colored with a sentiment mentioned in Ramsey coloring of ThoughtNet in 236 (or SentiWordNet score). In terms of a Markov Model, Reward for an evocative word w at time t is denoted as Reward(w,t) and returns a corresponding thought hyperedge which depends on Reward(w, t-1), Reward(w, t-2) and so on i.e Reward(w,t) = function_of(Reward(w,t-1), Reward(w,t-2), ..., Reward(w,0)). Recommender Systems in e-commerce websites which display similar items to a selected item in shopping cart have a striking resemblance to ThoughtNet evocation. This makes ThoughtNet a suitable candidate for Recommender System - Sales history is built as a ThoughtNet similar to the algorithm mentioned in 283 and 288. Every item added to shopping cart returns evocatives based on some scoring (sentiment ranked etc.,) which is a Recommender System.

Reference:

295.1 Multiword Testing Decision Service - http://arxiv.org/pdf/1606.03966v1.pdf

---
296. (FEATURE-DONE) Music Pattern Mining - Jensen-Shannon Divergence Distance between two FFT Frequency Distributions for similarity
---
Commits - 20 July 2016 - related to 68, 69

(NOTE: Because of some weird SourceForge/GitHub error, FFT txt files added to

repos on 19 July 2016 are still flagged as new(?). They have been
added to repos again along with new files)

FFTs of 2 audio files are parsed and written to *_trimmed.txt by awk to contain
only the frequencies for each sample.
These files are read by JensenShannonDivergence.py to compute the JS Distance
between these two FFT frequency distributions.
Preprocessing is done so as to normalize the frequency to map to a probability
(frequency/sum_of_frequencies) which gives
probability distribution from frequency distribution. Jensen-Shannon Distance
which is the weighted average of bidirectional
Kullback-Leibler Divergence measures of the two distributions, indicates
similarity or distance between two music samples
quantitatively.

There are 4 music samples: music_pattern_mining/FFT_classical_1_20July2016.txt
and music_pattern_mining/FFT_classical_2_20July2016.txt
are similar (similar notes sung by different musicians).
music_pattern_mining/FFT_classical_1_19July2016.txt and
music_pattern_mining/FFT_classical_2_19July2016.txt are also similar (different
set of notes sung by different musicians). Jensen-Shannon
distance across these 4 ordered pairs is captured and committed in testlogs/

Jensen-Shannon Distance across 2 FFT Frquency-Probability distributions of music
samples is a simple, basic measure for distance and can be
basis for clustering and classification of music. There could be some
inaccuracies because Audacity does not generate FFT for complete
music file but only for first ~5 minutes. Two similar notes with different
musicians could be distant and two dissimilar notes might be
close because of this Audacity limitation. Presently noise filtering and
cherrypick peak frequencies is not done and
entire frequency range is compared.

---
297. (FEATURE-DONE) Software Analytics - Cyclomatic Complexity from SATURN .dot
graphs - related to 65

Commits - 22 July 2016

New Python-Spark implementation that reads the SATURN program analyzer generated
.dot graph files processes them with Spark
MapReduce to find the number of edges and vertices in the graph represented
by .dot file for each snippet. From this
Cyclomatic Complexity is calculated (which is Edges - Vertices + 2) - a
standard Function Point Estimator for code complexity.
There were some JVM crashes frequently while starting up spark-submit logs for
which are also committed in testlogs along with
the two successful Spark computed Cyclomatic Complexity measures for two .dot
files. These .dot files are sourced from
VIRGO Linux saturn program analysis kernel driver.

---
298. (FEATURE-DONE) String Search - Longest Common Substring - Suffix
Trees(Suffix Arrays+LCP) Implementation

Commits - 26 July 2016

This implementation finds the most repeated substrings within a larger string by

constructing Suffix Trees
indirectly with Suffix Array and Longest Common Prefix (LCP) datastructures. An ideal application for String Search is
in Bioinformatics, Streamed Data analysis etc., For example, a Time Series data with fluctuating curve can be
encoded as a huge binary string by mapping ebb and tide to 0 and 1. Thus a function graph time series is projected on to
{0,1} alphabet to create blob. Usual Time Series analysis mines for Trends, Cycles, Seasons and Irregularities over time.
This binary encoding of time series gives an alternative spectacle to look at the trends (highs and lows). Longest repeating
pattern in this binary encoding is a cycle. Suffix Array by [UdiManber] has been implemented over Suffix Trees implementations
of [Weiner] and [Ukkonen] because of simplicity of Suffix Arrays and Suffix Trees=Suffix Arrays + LCPs.

---
299. (FEATURE-DONE) Binary String Encoding of Time Series - Commits - 27 July 2016
---
1. New python script to encode a time series datastream as binary string has been added to repository.
2. This writes to StringSearch_Pattern.txt which is read by StringSearch_LongestRepeatedSubstring.py
3. Encoding scheme: If next data point is smaller, write 0 else if greater write 1. This captures the
fluctuations in dataset irrespective of the amplitudes at each point.
4. For example a bitonic sequence would have been encoded
as ....111111110000000.... (ascends and descends)
5. Suffix Array + LCP algorithm on this sequence finds the Longest Repeated Substring. For a specific example of
Stock ticker time series data this amounts to frequently recurring fluctuation pattern.
6. Logs for the above have been committed to testlogs/
7. Every time series dataset is a union of bitonic sequences with varying lengths corresponding to peaks and troughs
and has self-similar fractal structure (zooming out the series has similarities to original series) . This implies the
binary string encoded as previously is fractal too.

---
300. (FEATURE-DONE) Tornado GUI Authentication Template - Commits - 27 July 2016

Rewritten Tornado GUI with Login Template and redirect to Algorithms Execution Template. Presently implemented for only a
root user by cookie setting. Entry point has been changed to
http://host:33333/neuronrain_auth (Login).

---
301. (FEATURE-DONE) OAuth2 authentication and credentials storage in MongoDB/Redis with passlib sha256 encryption
---
1.Login Handler has been augmented with OAuth2 authentication by python-oauth2 library.
2.MongoDB is used as OAuth2 credentials storage backend and Redis for token storage backend
3.Passlib is used to encrypt the password with SHA256 hash digest and stored once in MongoDB (code is commented)

4.Login page password is verified with passlib generated hash in MongoDB queried for a username argument
5.With this minimal standard authentication has been implemented for NeuronRain cloud deployment.
6.Prerequisite is to populate MongoDB neuronrain_oauth database and neuronrain_users collections with
passlib encypted JSON by uncommenting the collections.insert_one() code.

---

302. Commits - 29 July 2016 - boost::python extensions for VIRGO Linux Kernel system calls

---

virgo_clone() system call has been included in switch-case and invokes an exported kernel module function in kernelspace.

---

303. (THEORY) Update on Bitonic Sort Merge Tree for Discrete Hyperbolic Factorization Spark Cloud Implementation in 34.20

Bitonic Sort Merge Tree for Discrete Hyperbolic Factorization described in http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.pdf/download has following bound which can be derived as:

303.1 Bitonic Sort requires O((logN)^2) time with O(N*(logN)^2) processors.
303.2 Discretized Hyperbolic arc of length N can be split into N/logN segments of length logN each.
303.3 Above N/logN segments can be grouped in leaf level of merge tree into N/2logN pairs of (logN,logN) = 2logN length segments to be sorted together.
303.4 Each such pair can be sorted in log(2logN) time in parallel with N/(2logN) * N(logN)^2 = N^2/2logN comparator processors in leaf level which is the maximum number of processors required. Next level requires log(4logN), log(8logN) and so on till root.
303.5 Height of the merge tree is O(log(N/logN)). Total sort time is sum over sorts at each level of merge tree = log(2logN) + log(4logN) + ... + log((2^log(N/logN))*logN)
303.6 Total sort time of merge tree is upperbounded as (not tight one) height * maximum_sort_time_per_level:
      <= O(log(N/logN) * log(N/logN*logN)) = O(log(N/logN) * log(N))
      with maximum of N^2/2logN processor comparators at leaf which tapers down up the tree.
303.7 Binary Search on final sorted merged tile requires additional O(logN) which effectively reduces to:
      O(log(N/logN)*logN + logN) = O(log(N/logN)*logN) <= O((logN)^2) time
      with N^2/2logN processor comparators for finding a factor.
303.8 This is comparably better bound than the Parallel RAM ANSV algorithm based merge sort alternatives and easy to implement on a cloud as done in 34.20 and also is in NC because of polylog time and polynomial comparator processors required.
303.9 Again the input size is N and not logN, but yet definition of NC is abided by. It remains an open question whether Bitonic Sort Comparators are equivalent conceptually to PRAMs (Are cloud nodes same as PRAMs assuming the nodes have access to a shared memory?).

---

304. Commits - 31 July 2016 - boost::python C++ and cpython virgo_clone() system call invocations

---

1. Boost C++ Python extensions - virgo_clone() system call has been included in switch-case for cpython invocation of VIRGO linux kernel with test logs for it with use_as_kingcobra_service=1 (Routing: C++Boost::Python ---> virgo_clone() ----> virgo_queue driver ----> KingCobra message queue pub/sub service). Test logs for it have been committed in testlogs/
2. CPython extensions - test log for boost::python virgo_clone() invocation with use_as_kingcobra_service=0 which directly invokes a kernelspace function without forwarding to virgo_queue/kingcobra, has been committed to testlogs/.

---
305. NEURONRAIN VIRGO Commits for ASFER Boost::Python C++ virgo memory system calls invocations
---
(BUG - STABILITY ISSUES) Commits - 1 August 2016 - VIRGO Linux Stability Issues - Ongoing Random oops and panics investigation
---
1. GFP_KERNEL has been replaced with GFP_ATOMIC flags in kmem allocations.
2. NULL checks have been introduced in lot of places involving strcpy, strcat, strcmp etc., to circumvent
buffer overflows.
3. Though this has stabilized the driver to some extent, still there are OOPS in unrelated places deep
with in kernel where paging datastructures are accessed - kmalloc somehow corrupts paging
4. OOPS are debugged via gdb as:
        4.1 gdb ./vmlinux /proc/kcore
        or
        4.2 gdb <loadable_kernel_module>.o
   followed by
        4.3 l *(address+offset in OOPS dump)
5. kern.log(s) for the above have been committed in tar.gz format and have numerous OOPS occurred during repetitive telnet and syscall
invocation(boost::python C++) invocations of virgo memory system calls.
6. Paging related OOPS look like an offshoot of set_fs() encompassing the filp_open VFS calls.
7. In C++ Boost::Python extensions, flag changed for VIRGO memory system calls invocation from python.

---
306. Commits - 3 August 2016

Social Network Analysis with Twitter - Changes to Belief Propagation Potential Computation
---
1. Exception handling for UnicodeError has been added in SentimentAnalyzer
2. Belief Propagation Potential computation for the RGO graph constructed has been changed to do plain
summation of positivity and negativity scores for DFS of K-Core rather than multiplication which heuristically
appears to predict sentiments better
3. An example for tweets sentiments analysis for 2 search keywords has been logged and committed in testlogs/
---
4. Excerpts for sentiment scores - positive and negative - from RGO graph of few tweets
---

```
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/SourceForge/asfer-
code/python-src/testlogs# grep "score:"
SocialNetworkAnalysis_Twitter.out1.Republicans.3August2016
K-Core DFS belief_propagated_posscore: 6.078125
K-Core DFS belief_propagated_negscore: 8.140625
Core Number belief_propagated_posscore: 25.125
Core Number belief_propagated_negscore: 26.125
K-Core DFS belief_propagated_posscore: 6.078125
K-Core DFS belief_propagated_negscore: 8.140625
Core Number belief_propagated_posscore: 25.125
Core Number belief_propagated_negscore: 26.125
K-Core DFS belief_propagated_posscore: 55.09375
K-Core DFS belief_propagated_negscore: 54.3125
Core Number belief_propagated_posscore: 92.625
Core Number belief_propagated_negscore: 93.875
K-Core DFS belief_propagated_posscore: 60.09375
K-Core DFS belief_propagated_negscore: 59.3125
Core Number belief_propagated_posscore: 118.625
Core Number belief_propagated_negscore: 117.75
root@shrinivaasanka-Inspiron-1545:/media/shrinivaasanka/0fc4d8a2-1c74-42b8-8099-
9ef78d8c8ea2/home/kashrinivaasan/KrishnaiResearch_OpenSource/SourceForge/asfer-
code/python-src/testlogs# grep "score:"
SocialNetworkAnalysis_Twitter.out2.Democrats.3August2016
K-Core DFS belief_propagated_posscore: 34.0
K-Core DFS belief_propagated_negscore: 30.359375
Core Number belief_propagated_posscore: 90.25
Core Number belief_propagated_negscore: 85.125
K-Core DFS belief_propagated_posscore: 54.0
K-Core DFS belief_propagated_negscore: 46.734375
Core Number belief_propagated_posscore: 111.125
Core Number belief_propagated_negscore: 108.125
K-Core DFS belief_propagated_posscore: 97.203125
K-Core DFS belief_propagated_negscore: 91.015625
Core Number belief_propagated_posscore: 172.125
Core Number belief_propagated_negscore: 172.125
K-Core DFS belief_propagated_posscore: 97.203125
K-Core DFS belief_propagated_negscore: 91.015625
Core Number belief_propagated_posscore: 178.125
Core Number belief_propagated_negscore: 178.125
```

5. Majority Predominant Sentiment indicated by above scores for randomly sampled tweets can be used as one of the Election Approximation and Forecast Technique described in 14.

---

307. (THEORY) Sentiment Analysis from Recursive Gloss Overlap as a Voting Function in Majority Voting - related to 14

As mentioned in 306, Sentiment Analysis scores derived from Recursive Gloss Overlap graph of a text is an indirect voting function - when the overall sentiment is negative it is an "against" vote and when positive it is a "for" vote. In this sense, Sentiment Analysis shrouds a Satisfiability problem underneath it. In other words every polarised text is a boolean SAT circuit translated into natural language text which opens whole new vista to look at Sentiment Analysis per se. Infact it is a generalization of a boolean SAT circuit - it is bipolar and whichever polarity wins decides the vote.

---

308. (FEATURE-DONE) Markov Random Fields (MRF) Belief Propagation - Commits - 8 August 2016

Existing Sentiment Analyzer does belief propagation by computing product of potentials in DFS traversal only.
New function for Markov Random Fields Belief Propagation has been included which handles the generalized case of belief propagation in a
graph by factoring it into maximal cliques and finding potentials per clique. These per clique potentials are multiplied and normalized
with number of cliques to get polarized sentiments scores - positive, negative and objective. Logs for Sentiment Analysis of tweets stream
with MRF have been committed to testlogs/

Reference:

308.1 Introduction to Machine Learning - [Ethem Alpaydin] - Graphical Models

---
309. (FEATURE-DONE) Commits - 12 August 2016 - Boost 1.58.0 upgrade from 1.55.0 - Boost::Python C++ extensions for VIRGO

1.Rebuilt Boost::Python extensions for VIRGO Linux kernel system calls with boost 1.58 upgraded from boost 1.55.
2.kern.log for miscellaneous VIRGO system calls from both telnet and system call request routes has been compressed and committed to
testlogs (~300MB).
Following multiple debug options were tried out for heisencrash resolution (Note to self):
3./etc/sysctl.conf has been updated with kernel panic tunables with which the mean time between crashes has increased and crash location is
deep within kernel (VMA paging) - commits for this are in VIRGO linux tree. With these settings following usecase works:
        virgo_cloud_malloc()
        virgo_cloud_set()
        virgo_cloud_get()
        virgo_cloud_set() overwrite
        virgo_cloud_get()
through telnet route.
4.Debugging VIRGO linux kernel with an Oracle VirtualBox virtual machine installation and debugging the VM with a netconsole port via KGDB was explored, but there are serious issues with VirtualBox initialization in Qt5.
5.Debugging VIRGO linux kernel with QEMU-KVM installation and debugging the VM with a netconsole port via KGDB is being explored. Reference documentation at: https://www.kernel.org/doc/Documentation/gdb-kernel-debugging.txt

---
310. (THEORY) DFT of a Sliding Window Fragment of Time Series, Integer Partitions and Hashing
---
Time Series Data Stream can be viewed through a sliding window of some fixed width moving across the data. Captured data points in such
window are analyzed in frequency domain with a Discrete Fourier Transform. If there are n frequencies in the window of data,there are n
sinusoids which superimpose to form the original data. In discrete sense, sinusoids partition the data at any time point. Following the
relation between hash functions and integer partitions in https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions_2014.pdf , each of n discrete sinusoid in DFT corresponds to a hash function and the time-series sliding window is an amalgamation of n hash functions.

---
311. (THEORY) Pr(Good) Majority Voting Circuit, Percolation, PRG choice and Boolean Circuit Composition - related to 14, 53.12 and 129

Even though Percolation circuit is in Noise Stability Regime with zero sensitivity and 100% stability, delta term mentioned in matrix of 14 and 53 (second column) need not be zero which could force percolation circuit to be in BPNC. Problem is then derandomizing BPNC. (Open question: Is BPNC in NC/poly?). Thus LHS of Pr(Good) could be in BPNC which coincides with [Applebaum] NC1 PRG based pseudorandom choice described in - https://5d99cf42-a-62cb3a1a-s-sites.googlegroups.com/site/kuja27/LowerBoundsForMajorityVotingPseudorandomChoice.pdf and https://5d99cf42-a-62cb3a1a-s-sites.googlegroups.com/site/kuja27/IndepthAnalysisOfVariantOfMajorityVotingwithZFAOC_2014.pdf. Boolean Function Composition is preferred over Oracle results for Majority Voting circuit which is an NC1 circuit with CircuitSAT as inputs for each voter. This is infact "Boolean Circuit Composition" corresponding to Boolean Function Composition.

References:

311.1 Mathematical Techniques for Analysis of Boolean Functions - [AnnaBernasconi] - 2.8.2 Boolean Function Composition - www.di.unipi.it/~annab/tesi.ps.gz
311.2 Properties and Applications of Boolean Function Composition - [AvishayTal] - eccc.hpi-web.de/report/2012/163/download/
311.3 Boolean Circuit Composition - www.cs.tau.ac.il/~safra/ACT/CCandSpaceB.ppt

---
312. (THEORY) Circuit SAT lowerbounds for Pr(Good) Majority Voting - related 14 and 129

312.1 RHS of Pr(Good) Majority Voting circuit is an NC circuit in composition with CircuitSAT for each voter. This parallel voting by electorate requires some kind of barrier synchronization so that input from all voters is available to NC Majority Circuit. This is a non-trivial overhead to be ignored because the barrier delay depends on the voter whose decision time is the most. If in worst case the bottleneck voter function is in EXP then barrier synchronization is also in EXP. It is also assumed that voters decide independent of each other.
312.2 CircuitSAT is the circuit version of CNF SAT concerned with the bounds for finding satisfying assignments to a circuit to get the output true at root of Circuit DAG.
312.3 An arbitrary Circuit's Satisfiability requires O(2^0.4058m) where m is the number of gates in Circuit(i.e. size of circuit).This bound was proved in [SergeyNurk] - ftp://ftp.pdmi.ras.ru/pub/publicat/preprint/2009/10-09.pdf.gz. This is very generic bound without concerning the complexity classes of voter decisioning function.
312.4 Because of the fact that Pr(good) majority voting is a circuit composition with voter circuit SATs as inputs to NC majority circuit, previous bound applies and estimates the time for each voter's decision to be input to Majority circuit. In other words, a satisfying assignment to atleast half of the electorate CircuitSATs has to be found to pronounce a winner. Time required to convince a voter is therefore exponential in size of the voter circuit. For a voter k size of the CircuitSAT is m(k) and the time required to satisfy it is O(2^0.4058*m(k)).
312.5 Let number of voters be n and number of gates on the average per voter be m. NC1 majority requires O(logn) time because n voters input to it at leaf.There are mn gates for all voters. Per voter decision requires O(2^0.4058m) time in parallel to find a satisfying assignment to atleast n/2 voters for majority to output 1.
312.6 If number of voters is exponential in number of gates per voter then n=2^m

which is the worst case scenario and is the most realistic, then size of the Majority voting circuit in RHS of Pr(Good) is m*2^m - exponential in size of per voter circuit. LHS of Pr(Good) is NC/poly or BPNC percolation circuit while RHS is an exponential circuit (DC uniform - PH if depth restricted or EXP if depth unrestricted)

312.7 In a very generic case, as mentioned in 275, each voter can have a non-boolean voting function viz., a linear program or constraint satisfaction which is NP if integer-LP and is in P if real-LP (simplex, interior point). In terms of Oracles circuit for RHS Pr(Good) can be written as:

312.7.1 NC^P (if voting functions are real LP) and LHS NC/poly or BPNC percolation lowerbounds NC^P when both LHS and RHS are of same error

312.7.2 NC^NP (if voting functions are integer LP) and LHS NC/poly or BPNC percolation lowerbounds NC^NP when both LHS and RHS are of same error

312.8 Circuit Size of Pr(Good) RHS is non-trivial to determine. All that is known so far is the bound for CircuitSAT which is O(2^0.4058m) where m is number of gates. But number of gates itself is an open problem for SAT - it is not known if NP has polynomial size circuits.

312.9 Assumption 1: Each voter has different variables - set of variables of voters are all pairwise disjoint.

312.10 If number of voters is arbitrarily huge and if number of voters is exponential in number of gates per voter SAT, size of RHS is m2^m with unbounded fan-in, unrestricted depth most probably and thus in EXP, with AND-OR-NOT gates. Assumption 2: When voters have common variables it implies that they decide in unison. For example if variable x is common to voters v1 and v2, both v1 and v2 assign x=0 or x=1. This is negation of Assumption 1 in 312.9 and Assumption 1 is more realistic than Assumption 2.

312.11 Even if m (number of gates per voter SAT) is assumed to be polynomial in number of input variables i.e m=f(v) where v is average number of inputs per voter, size of RHS is f(v)*2^f(v) - exponential in number of variables. Thus in all probability RHS majority voting circuit is exponential DC-uniform circuit in PH or EXP even if NP has polynomial size circuits. This is a special case with lot of assumptions made in 312.6,312.9 and 312.10.

312.12 If both LHS and RHS of Pr(Good) are of zero or equal error, LHS lowerbounds RHS (an assumption made throughout this document - equal error with differing circuit sizes implies lowerbound). Here error is the usual (NoiseSensitivity +/- delta) on both LHS and RHS. For percolation circuit in LHS, NoiseSensitivity is zero . But for RHS NoiseSensitivity is unknown. With an assumption that RHS has non-zero error, it is a BP.EXP circuit assuming unrestricted depth of RHS. LHS is NC/poly with zero error, while RHS is BPEXP and thus implies BPEXP is in NC/poly which resembles BPP in P/poly. If LHS has an error with a PRG choice function in NC (e.g Applebaum NC PRG with linear stretch), LHS would have been a BPNC circuit implying BPEXP is in BPNC which is absurd and conflicts with assumption that equal error implies lowerbound (refer 53.9, 53.14 and 53.16) - this conflict is resolved only if LHS and RHS do not have equal error with differing circuit sizes (and also implies that error depends on circuit size?).

312.13 Important notion in representing a voter boolean function by a circuit is Hardness which is defined as:

A boolean function f:{0,1}^n->{0,1} is h-hard if there exists a circuit C of size s such that for all x in {0,1}^n, Pr(C(x)=f(x)) < h. Narrative above on CircuitSAT for voter boolean functions is incomplete without knowing how hard is it to compute or approximate the voter decision function with a circuit of size s. Harder a function, lower the value of h (0 < h < 1). This definition of hardness is far better generalization of computational complexity of a voter boolean function than identifying with complexity class names. [Yao] XOR Lemma is applied to amplify hardness of boolean function by XOR-ing values of f on many randomly distributed inputs.

312.14 There are two classes of errors: Voter error + Voting error. Voter error is the Noise Sensitivity of a voter boolean function while Voting error is the intrinsic error in voting system itself though user has done nothing wrong. E.g Malfunctioning Electronic Voting Machines. Voter error is captured by second and third column of matrix in 53.14 while Voting error is captured by probabilistic truth table of the function - inputs have no effect on this error. This implies delta in Pr(Good) series error (NoiseSensitivity +/- delta) might have to

account for voting error term also, over and above voter error.

References:

312.15 Complexity Hardness of Noisy Boolean Functions - http://cstheory.stackexchange.com/questions/18822/hardness-of-noisy-boolean-functions - hardness of a noise-operated boolean function. Has applications to voting error where a voter's decision boolean function is perturbed by noise to output a wrong vote. Also, comparability and p-selectivity of a set - [D.Sivakumar] - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.16.648&rep=rep1&type=pdf and [AgrawalArvind94] on "Polynomial time truth-table reductions to P-selective sets". Every voting function (and ranking function in general) over n candidates is a p-selector with candidate set being p-selective set because for any two candidates x,y voting function makes a choice.
312.16 Hardness of Boolean Functions and Amplification - [RyanODonnell] - https://www.cs.cmu.edu/~odonnell/boolean-analysis/lecture17.pdf
312.17 Probabilistic Boolean Logic - [LakshmiNBChakrapani-KrishnaPalem] - http://www.ece.rice.edu/~al4/visen/2008ricetr.pdf - Truth table of a boolean function has 1s and 0s with probabilities. This captures the notion of error in voting function truth table itself, not just limited to noise correleations of inputs.
312.18 How much a Linear Programming Oracle speedup Polytime algorithms - http://cstheory.stackexchange.com/questions/32646/how-much-would-a-sat-oracle-help-speeding-up-polynomial-time-algorithms - Pr(Good) majority voting circuit is NC with SAT/LP oracle for each voter.
312.19 Algorithms for Circuits and Circuits for Algorithms - [RyanWilliams] - http://web.stanford.edu/~rrwill/ICM-survey.pdf - Kolmogorov conjecture based on [ArnoldKolmogorov] answer to Hilbert's 13th problem - every continuous function in 3 variables can be expressed as finite composition of functions of 2 variables - discrete analog : Is 3SAT expressible as a finite composition of 2SAT (which implies composition of P instances yielding NP as counterexample)? A special case majority voting with 3 variables common to all voter functions and Non-boolean continuous voting functions in 3 variables precisely has application of Hilbert's 13th problem.
312.20 Depth 3 multilinear circuits - [OdedGoldreich] - http://www.wisdom.weizmann.ac.il/~oded/R3/kk.pdf - D-canonical depth-3 circuit is constructed by composition of 2 depth-2 circuits i.e F=H(F1,F2,...,Fn) in Section 1.2. Has strong applications to a Majority voting circuit composition where Fi(s) are voting functions and H is Majority function.
312.21 Size hierarchy theorem for circuits - http://cstheory.stackexchange.com/questions/5110/hierarchy-theorem-for-circuit-size. And also theorems 5.8 and 5.9 in http://www.complexity.ethz.ch/education/Lectures/ComplexityFS12/skript_ch5.pdf - there are functions computable by (1+o(1))2^n/n and not computable by circuits of size 2^n/n.
312.22 Size Hierarchy [Lupanov] - Number of functions computable by circuit of size s - Lemma 2.1 counting argument - http://eccc.hpi-web.de/resources/pdf/cobf.pdf (Wegener - pages 88-92)
312.23 Efficient Parallel Computation = NC - [AroraBarak] - http://theory.cs.princeton.edu/complexity/book.pdf - Theorem 6.24 - Parallel Voting Simulation can be done by massive parallel processing assuming the input votes are precomputed.
312.24 Size hierarchy of 312.22 in Pr(Good) majority voting circuit context implies that number of functions computable by LHS has to be different from number of functions computable by RHS when LHS is NC/poly and RHS is PH or EXP i.e if EXP does not have polysize circuits. The central motivation for equating LHS and RHS of Pr(Good) is both sides are 2 algorithms to decide same question: Non-majority choice and Majority choice - which is more efficient where efficiency implies less error? This motivation therefore leads to assumption stated in disclaimer earlier: less circuit size implies a lowerbound (i.e least circuit size of LHS and RHS is chosen as separating class containing the other) when errors are equal or less.
312.25 If circuit sizes differ in zero-error case on both sides i.e when 100%

convergence occurs (or) equal error on both sides of series, then by size hierarchy, size(LHS) < size(RHS) implies RHS computes functions not possible by LHS and viceversa (Open question: Does this contradict lowerbound assumption? Or is this just a special case because both LHS and RHS compute same function i.e they answer same question in which case size hierarchy does not apply which counts all possible functions for specific circuit size?).
312.26 There is an alternative definition of Boolean function hardness in http://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/NOAM/HARDNESS/final.pdf (Section 2.2) which states hardness = circuit size complexity hardness + approximation hardness.

---
313. (FEATURE-DONE) Astronomical Sequence Mining based Precipitation Forecast (from 100 year historic dataset) - related to 172, 278

Following the earlier experimental pattern mining done on astronomical datasets and their striking correlational coincidence with weather vagaries, below rule search explores highly probable heavy precipitation - shows peaks after 29 November 2016 :

```
python MaitreyaEncHoro_RuleSearch.py  --min_year=2016 --min_month=11 --min_days=15 --min_hours=10 --min_minutes=10 --min_seconds=10 --min_long=77 --min_lat=7 --max_year=2016 --max_month=12 --max_days=30 --max_hours=10 --max_minutes=10 --max_seconds=10 --max_long=78 --max_lat=10 |grep "a Class Association"

{  --date="2016-11-29 10:10:10 5"  --location=" x 77:0:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-11-30 10:10:10 5"  --location=" x 77:0:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-12-1 10:10:10 5"  --location=" x 77:0:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-12-2 10:10:10 5"  --location=" x 77:0:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-11-29 10:10:10 5"  --location=" x 77:0:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-11-30 10:10:10 5"  --location=" x 77:0:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-12-1 10:10:10 5"  --location=" x 77:0:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-12-2 10:10:10 5"  --location=" x 77:0:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-11-29 10:10:10 5"  --location=" x 77:0:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-11-30 10:10:10 5"  --location=" x 77:0:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-12-1 10:10:10 5"  --location=" x 77:0:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
{  --date="2016-12-2 10:10:10 5"  --location=" x 77:0:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign Sagittarius
```

{ --date="2016-11-29 10:10:10 5" --location=" x 77:0:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:0:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:0:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:0:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:0:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:0:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:0:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:0:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:0:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:0:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:0:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:0:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:0:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:0:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:0:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:0:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:0:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:0:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:0:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:0:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:0:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:0:0 7:8:0 " --planet-list }

- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:0:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:0:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:0:0 8:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:0:0 8:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:0:0 8:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:0:0 8:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:0:0 8:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:0:0 8:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:0:0 8:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:0:0 8:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 8:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 8:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 8:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 8:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 8:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 8:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 8:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 8:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in

sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 9:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 9:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 9:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 9:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius

{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 9:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 9:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 9:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 9:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 9:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 9:6:0 " --planet-list

} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 9:6:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 9:6:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 9:7:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 9:7:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 9:7:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 9:7:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:0:0 9:8:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:0:0 9:8:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:0:0 9:8:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:0:0 9:8:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:1:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:1:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:1:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:1:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:1:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:1:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:1:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:1:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:1:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:1:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:1:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign

Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:1:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:1:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:1:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:1:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:1:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:1:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:1:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:1:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:1:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:1:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:1:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:1:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:1:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:1:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:1:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:1:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:1:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:1:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:1:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:1:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:1:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius

{ --date="2016-11-29 10:10:10 5" --location=" x 77:1:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:1:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:1:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:1:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:1:0 8:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:1:0 8:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:1:0 8:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:1:0 8:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 8:1:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 8:1:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 8:1:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 8:1:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 8:2:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 8:2:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 8:2:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 8:2:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 8:3:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 8:3:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 8:3:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 8:3:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 8:4:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 8:4:0 " --planet-list

} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 8:4:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 8:4:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 8:5:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 8:5:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 8:5:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 8:5:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 8:6:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 8:6:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 8:6:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 8:6:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 8:7:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 8:7:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 8:7:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 8:7:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 8:8:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 8:8:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 8:8:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 8:8:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 9:0:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 9:0:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 9:0:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in

sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 9:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 9:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 9:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 9:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 9:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius

{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 9:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 9:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 9:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 9:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 9:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 9:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 9:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 9:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:1:0 9:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:1:0 9:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:1:0 9:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:1:0 9:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:2:0 7:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:2:0 7:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:2:0 7:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:2:0 7:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:2:0 7:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:2:0 7:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:2:0 7:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:2:0 7:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:2:0 7:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:2:0 7:2:0 " --planet-list }

- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:2:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:2:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:2:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:2:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:2:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:2:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:2:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:2:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:2:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:2:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:2:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:2:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:2:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:2:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:2:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:2:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:2:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:2:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:2:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:2:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:2:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign

Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:2:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:2:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:2:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:2:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:2:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 8:0:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 8:0:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 8:0:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 8:0:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 8:1:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 8:1:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 8:1:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 8:1:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 8:2:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 8:2:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 8:2:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 8:2:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 8:3:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 8:3:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 8:3:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 8:3:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius

{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 9:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 9:0:0 " --planet-list

} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 9:0:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 9:0:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 9:1:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 9:1:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 9:1:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 9:1:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 9:2:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 9:2:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 9:2:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 9:2:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 9:3:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 9:3:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 9:3:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 9:3:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 9:4:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 9:4:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 9:4:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 9:4:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 9:5:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 9:5:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 9:5:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in

sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 9:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 9:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 9:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 9:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 9:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 9:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 9:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 9:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 9:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:2:0 9:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:2:0 9:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:2:0 9:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:2:0 9:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:3:0 7:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:3:0 7:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:3:0 7:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:3:0 7:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:3:0 7:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:3:0 7:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:3:0 7:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:3:0 7:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius

{ --date="2016-11-29 10:10:10 5" --location=" x 77:3:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:3:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:3:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:3:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:3:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:3:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:3:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:3:0 7:3:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:3:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:3:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:3:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:3:0 7:4:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:3:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:3:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:3:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:3:0 7:5:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:3:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:3:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:3:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:3:0 7:6:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:3:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:3:0 7:7:0 " --planet-list }

```
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-12-1 10:10:10 5"  --location=" x 77:3:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-12-2 10:10:10 5"  --location=" x 77:3:0 7:7:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-11-29 10:10:10 5"  --location=" x 77:3:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-11-30 10:10:10 5"  --location=" x 77:3:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-12-1 10:10:10 5"  --location=" x 77:3:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-12-2 10:10:10 5"  --location=" x 77:3:0 7:8:0 " --planet-list }
- There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-11-29 10:10:10 5.5"  --location=" x 77:3:0 8:0:0 " --planet-list
} - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-11-30 10:10:10 5.5"  --location=" x 77:3:0 8:0:0 " --planet-list
} - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-12-1 10:10:10 5.5"  --location=" x 77:3:0 8:0:0 " --planet-
list } - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in
sign  Sagittarius
{  --date="2016-12-2 10:10:10 5.5"  --location=" x 77:3:0 8:0:0 " --planet-
list } - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in
sign  Sagittarius
{  --date="2016-11-29 10:10:10 5.5"  --location=" x 77:3:0 8:1:0 " --planet-list
} - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-11-30 10:10:10 5.5"  --location=" x 77:3:0 8:1:0 " --planet-list
} - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-12-1 10:10:10 5.5"  --location=" x 77:3:0 8:1:0 " --planet-
list } - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in
sign  Sagittarius
{  --date="2016-12-2 10:10:10 5.5"  --location=" x 77:3:0 8:1:0 " --planet-
list } - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in
sign  Sagittarius
{  --date="2016-11-29 10:10:10 5.5"  --location=" x 77:3:0 8:2:0 " --planet-list
} - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-11-30 10:10:10 5.5"  --location=" x 77:3:0 8:2:0 " --planet-list
} - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-12-1 10:10:10 5.5"  --location=" x 77:3:0 8:2:0 " --planet-
list } - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in
sign  Sagittarius
{  --date="2016-12-2 10:10:10 5.5"  --location=" x 77:3:0 8:2:0 " --planet-
list } - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in
sign  Sagittarius
{  --date="2016-11-29 10:10:10 5.5"  --location=" x 77:3:0 8:3:0 " --planet-list
} - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-11-30 10:10:10 5.5"  --location=" x 77:3:0 8:3:0 " --planet-list
} - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in sign
Sagittarius
{  --date="2016-12-1 10:10:10 5.5"  --location=" x 77:3:0 8:3:0 " --planet-
list } - There is a Class Association Rule match [ ['Mercury', 'Venus'] ] in
```

sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 8:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 8:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 8:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 8:6:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 8:7:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 8:8:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius

{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 9:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 9:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 9:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 9:0:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 9:1:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 9:2:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 9:3:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 9:4:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 9:5:0 " --planet-list } - There is a Class Association Rule match [['Mercury', 'Venus']] in sign Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 9:5:0 " --planet-list

} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 9:5:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 9:5:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 9:6:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 9:6:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 9:6:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 9:6:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 9:7:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 9:7:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 9:7:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 9:7:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5.5" --location=" x 77:3:0 9:8:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5.5" --location=" x 77:3:0 9:8:0 " --planet-list
} - There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5.5" --location=" x 77:3:0 9:8:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-12-2 10:10:10 5.5" --location=" x 77:3:0 9:8:0 " --planet-
list } - There is a Class Association Rule match [['Mercury', 'Venus']] in
sign Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:4:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:4:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:4:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:4:0 7:0:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:4:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:4:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:4:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign

Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:4:0 7:1:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-29 10:10:10 5" --location=" x 77:4:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-11-30 10:10:10 5" --location=" x 77:4:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-1 10:10:10 5" --location=" x 77:4:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius
{ --date="2016-12-2 10:10:10 5" --location=" x 77:4:0 7:2:0 " --planet-list }
- There is a Class Association Rule match [['Mercury', 'Venus']] in sign
Sagittarius

---

314. (THEORY) Pseudorandomness, Voter Decision Error and lambda-tolerant Randomized Decision Trees of Voter Boolean Functions

---

Pseudorandom generators are those which stretch a seed l(n) to n : f:{0,1}^l(n)->{0,1}^n indistinguishable from perfect random source
with bounded error. References 314.1 and 314.2 describe a counting argument to show that there are PRGs. For a seed length d there are 2^d possible strings stretched to length n. For a single random string the probability that a circuit or an algorithm distinguishes it from perfect random source is e. By iterating over all 2^d strings generated by a PRG the expected fraction of strings distinguished is e*2^d where each iteration is a random variable with probability e and follows from union of expectations - E(sigma(Xi)) = E(X1) + E(X2) + ... + E(X(2^d)) = sigma(Xi*e). Probability that there are tails where deviation from this expected value is less than e so that indistinguishability is preserved is by Chernoff bounds: Pr(sigma(Xi)-mean > delta*mean) <= 2*exp(-delta^2*mean/3) where 0 < delta < 1 and e=delta*mean. Substituting for mean, this Probability <= 2*exp(-2^d*e*delta^2/3). For all possible algorithms or circuits of size s which could be 2^s, by union bound cumulative probability is 2^s*2*exp(-2^d*e*delta^2/3) < 1. It implies probability of distinguishability is < 1 and there should be a lurking PRG which achieves indistinguishability. This previous description is mooted to connect PRGs with Voter Decision Error. Voters have a boolean decision function in Pr(Good) majority voting circuit. Alternatively each voter decides by evaluating the leaves of the boolean decision tree of the corresponding function. Randomized decision trees allow randomness by allowing coin tosses to choose the next edge in decision tree. Reference 314.3 describes a decision tree evaluation by voter which allows error by including decision tree choices not necessarily belonging to the voter. That is, voter errs in judgement during decision process wherein (pseudo)random error pollutes his/her discretion. This formalizes the notion of Voting error (analogous somewhat to Probabilistic CNFs). The level of error probability to what decision tree deviates is upperbounded by lambda and called as lambda-tolerance. There are known results which bound the error-free decision tree complexity and erring decision tree complexity. Quoting 314.3, "...The possible algorithms can output anything. The mistake can be either way...". 1-way error occurs in one direction - 1 is output instead of 0 while 2-way error occurs in both directions - 1 for 0 and 0 for 1. This error is intrinsic to the decision process itself not just limited to sensitivity measures which depend on flipped inputs affecting outputs. Thus if error is modelled as a function of PRGs, error in voting process as a whole implies existence of PRGs. For example above distinguisher can be mapped to a voter who fails to distinguish two candidates with differing goodness - voter is fooled - a less merited candidate if had access to the intrinsic PRG that vitiates the decision tree evaluation can exploit the voter's flawed decision tree.

References:

314.1 Definition of Pseudorandomness - Simpler Distinguisher - http://people.seas.harvard.edu/~salil/pseudorandomness/prgs.pdf - Page 217 and Proposition 7.8 in Page 218
314.2 Definition of Pseudorandomness - Counting argument - http://www.cse.iitk.ac.in/users/manindra/survey/complexity25.pdf - Page 3 - Definition 21
314.3 Lambda-tolerant Randomized Boolean Decision Trees - https://www.math.u-szeged.hu/~hajnal/research/papers/dec_surv.gz - Page 5

---
315. (FEATURE-DONE and BUG-STABILITY ISSUES) NeuronRain AsFer Commits for Virgo Linux - Commits - 25 August 2016

Kernel Panic investigation for boost::python c++ invocations of Virgo System Calls
---
1.Python code in AsFer that invokes Virgo Linux system calls by boost::python C++ bindings is being investigated further for mystery and random crashes that occur after system call code and driver end code is finished.
2.kern.log with 3 iterations of virgo_malloc()+virgo_set()+virgo_get() invocations succeeded by panics in random points at kernel has been committed.
3.Logs also contain the gdb vmlinux debugging showing line numbers within kernel source where panics occur. Prima facie look unrelated directly to virgo system calls and driver code. Debugging through KVM+QEMU is ruled out because of lack of VT-x.

---
316. (FEATURE-DONE and BUG-STABILITY ISSUES) NeuronRain AsFer Commits for Virgo Linux - Commits - 26 August 2016

Kernel Panic investigation for boost::python c++ invocations of Virgo System Calls
---
kern.log with panic in FS code after boost::python C++ AsFer-VIRGO Linux system calls invocations

---
317. (THEORY) Generic Definitions of Majority and Non-majority Choice Errors and some contradictions - related to 14 and 53.14

Majority Voting Social Choice has usual meaning of aggregation of for and against votes. Non-majority Social Choice can be in two flavours:
317.1 Pseudorandom Choice - a Pseudorandom Generator chooses a voting function from a set of Voting functions of electorate
317.2 Social choice by ranking - set of boolean functions are ranked ascending in their error probabilities and least error boolean function is chosen.

Algorithm for 317.1:

There are pseudorandom generators in NC(Applebaum PRG, Parallel PRGs etc.,) which choose a boolean function at random. This chosen boolean function can be

Percolation boolean function too which is again in non-uniform NC. Thus Non-majority social choice can be done in BPNC as below:

- Invoke a PRG in NC or P to obtain pseudorandom bits X.
- Choose an element in electorate indexed by X.
- Goodness of LHS is equal to the goodness of chosen element.

LHS is BPNC algorithm to RHS BPEXP Condorcet Jury Theorem (CJT) unbounded circuit and if CJT converges to 1 in RHS and Goodness of PRG choice is 1, LHS is a BPNC algorithm to EXP RHS.

Algorithm for 317.2:

- foreach(voter)
- {
- Interview the voter boolean function : Rank the voters by merit (error probabilities e.g noise stability)
- }
- choose the topmost as non-majority social choice

LHS is a PSPACE-complete algorithm and RHS is either BPEXP or EXP algorithm depending on convergence or divergence of Goodness of CJT circuit in RHS. Goodness of LHS is the error probability of topranked boolean voter function.

Above loop can be parallelized and yet it is in PSPACE. Proof of CJT circuit in RHS to be EXP-complete would immediately yield a lowerbound and either EXP is in BPNC or EXP is in PSPACE under equal goodness assumption in LHS and RHS. EXP in PSPACE implies EXP=PSPACE because PSPACE is in EXP. An EXPTIME problem is EXP-Complete if it can solve Exponential Time Bounded Halting Problem (i.e output 1 if a Turing Machine halts after exponential number of steps). Unbounded depth RHS CJT Majority voting composition circuit (Majority+SAT) is in EXPTIME. Following reduction from Majority voting to Bounded Halting Problem is a proof outline for EXP-completeness of CJT circuit:

(*) Let there be exponential number of voters (i.e exponential in number of variables)

(*) Number of voters = Number of steps in Bounded Halting Problem input Turing Machine = exponential

(*) Voting halts when all exponential voters have exercised franchise applying their SAT = EXPTIME Turing Machine Halts after exponential number of steps.

If there are n boolean functions in total and probability that a boolean function i with goodness e(i) is chosen is:

= x(e(i))/n, where x(e(i)) is the number of boolean functions with goodness e(i)

When voter boolean functions have unequal error probabilities, the distribution is Poisson Binomial (which is for bernoulli trials with unequal probabilities for each event) and not Binomial or Poisson(which is the limit of Binomial distribution). This fraction is the LHS and in special case can be 1 when all boolean functions are of same goodness (1-error) probability. Thus LHS goodness is conditional probability x(e(i))/n * e(i) and can be 1 when all boolean functions are perfect.

Thus most generic Majority voting case is when:

317.3 Voters have unequal decision errors

317.4 Set of voter boolean functions is an assorted mix of varied error probabilities ranging from 0 to 1.

317.5 Modelled by Poisson Binomial Distribution

In both Majority and Non-majority Choice, error of a boolean function comprises:

317.6 Voting Error - Misrecorded votes, Probabilistic CNFs

317.7 Voter Error - Noise Sensitivity of boolean function which has extraneous reasons like correlated flipped bit strings and Error intrinsic to boolean function in Decision tree evaluation (lambda-tolerant randomized

decision trees) mentioned in 314.
Randomized decision tree evaluation in 317.7 adds one additional scenario to error matrix of 8 possibilities in 14 and 53.14 which is beyond just correlation error.

As mentioned elsewhere in this document and disclaimer earlier, LHS circuit is a lowerbound to RHS circuit for a C-complete class of problems when errors are equal, assuming complete problems exist for class C. When both LHS and RHS have errors, which is the most realistic case, LHS is a BPNC circuit and RHS is a BPEXP circuit with unrestricted depth of exponential sized circuit. This raises a contradiction to already known result as follows: BPP is not known to have complete problems. [Marek Karpinski and Rutger Verbeek - KV87] show that BPEXP is not in BPP and EXP is in BPEXP. If there are BPEXP-complete problems then contradiction is LHS is a BPNC algorithm to RHS BPEXP-complete problem. But BPNC is in BPP (logdepth, polysize circuit with error can be simulated in polytime with error) and BPEXP is not in BPP. This contradiction is resolved only if there are no BPEXP-complete problems or there are no BPNC/BPEXP voting functions.

From https://www.math.ucdavis.edu/~greg/zoology/diagram.xml, following chain of inclusions are known:

NC in BPNC in RNC in QNC in BQP in DQP in EXP
RP in BPP in BQP in DQP in EXP

If LHS is a BPNC or RNC or BPP or RP pseudorandom algorithm to unbounded CJT RHS EXP-complete problem under equal goodness assumption, previous class containments imply a drastic collapse of EXP below:

EXP=BPNC (or) EXP=RNC (or) EXP=RP (or) EXP=BPP

This is rather a serious collapse because the containments include Quantum classes BQP and QNC, implying classical parallelism is equivalent to quantum parallelism . One of the interpretations to Quantum parallelism is "Many Worlds" - computations happen in parallel (states of the wave function) and interfere construnctively or destructively (superposition of wave functions when some state amplitudes cancel out while others reinforce) to curtail some of the worlds. This special case occurs only if all voters have decision correctness probability p=1. For 1 > p > 0.5, RHS CJT circuit goodness converges to 1 for infinite electorate but LHS expected goodness of Pseudorandom social choice is always less than 1, infact tends to 0 as N is infinite as below. Referring to 359:

Let number of voter decision functions with goodness xi = m(xi). Thus N = m(x1)+m(x2)+m(x3)+...+m(xn)

Expected goodness of a PRG choice is:

1/N * summation(xi*m(xi)) = (x1*m(x1) + x2*m(x2) + x3*m(x3) + ... + xn*m(xn)) / N

When all voter functions have goodness 1 then PRG choice in LHS of P(good) has goodness 1.

For p=0.5, RHS goodness is 0.5 and LHS goodness is 0.5 => LHS is in BPNC or RNC or RP or BPP and RHS is BPEXP. If RHS is BPEXP-complete, as mentioned previously, contradicts result of [KV87], implying BPEXP=BPP. For p < 0.5, RHS CJT series goodness tends to 0 while LHS pseudorandom chouce expected goodness is always > 0. Thus only feasible lowerbound looks to be:

EXP=BPNC (or) EXP=RNC (or) EXP=RP (or) EXP=BPP

when goodness p=1 uniformly for all infinite homogeneous electorate.

References:

317.8 Derandomization Overview - [Kabanets] - http://www.cs.sfu.ca/~kabanets/papers/derand_survey.pdf
317.9 Unconditional Lowerbounds against advice - [BuhrmanFortnowSanthanam] - http://homepages.inf.ed.ac.uk/rsanthan/Papers/AdviceLowerBoundICALP.pdf
317.10 EXPTIME-Completeness - [DingZhuDu-KerIKo] - Theory of Computational Complexity - https://books.google.co.in/books?id=KMwOBAAAQBAJ&pg=PT203&redir_esc=y#v=onepage&q&f=false
317.11 BPP with advice -

http://people.cs.uchicago.edu/~fortnow/papers/advice.pdf - "...It can be shown using a translation argument that BPEXP is not in BPP [KV87], but this translation argument does not extend to showing a lower bound against advice. Since BPP and BPEXP are semantic classes, it is unknown whether for instance BPEXP ⊆ BPP/1 implies BPEXP/1 ⊆ BPP/2.". LHS pseudorandom social choice algorithm is non-uniform because randomly choosing an element from a varying number of electorate requires maximum size of electorate as advice.

---

318. (THEORY) Yao's XOR Lemma, Hardness of Pr(Good) Majority Voting Function Circuit and Majority Version of XOR Lemma

---

Caution: Majority Hardness Lemma derivation is still experimental with possible errors.

Hardness of a boolean function is defined as how hard computationally it is to compute the function with a circuit of size s:

if Pr(C(x) != f(x)) = delta where C(x) computes boolean function f(x), then f(x) delta-hard.

Majority function is known to have formula of size O(n^5.3) - from Sorting networks of [AjtaiKomlosSzemeredi] and non-constructive proof by[Valiant]. Majority is computable in log-depth and hence in non-uniform NC1.

Hardness Amplification result by [AndrewYao] states that a strongly-hard boolean function can be constructed from mildly-hard boolean function by amplification of hardness using XOR of multiple instances of function on some distribution of inputs. Following derivation develops intuition for amplification in XOR lemma:

318.1 Let f be a boolean function with delta-hardness. Therefore there is a circuit C(x) with size s,such that Pr(C(x)!=f(x))=delta. XOR lemma states that there is a circuit C1(x) with size s1, such that Pr(C1(x)!=f1(x)) tends to 0.5 > delta where f1(x) = f(x1) XOR f(x2) XOR ... XOR f(xk) for some distribution of inputs x1,x2,x3,...xk.

318.2 Each f(xi) in XOR function f1 in 318.1, is either flipped or not flipped. Let Zi be the random variable for flip of f(xi) - Zi=0 if there is no flip and Zi=1 if there is a flip using coin toss. Now the probability of correctly computing XOR of f(xi)s is equal to the sum = Probability that none of f(xi)s are flipped + Probability that even number of f(xi) is flipped. Even flips do not alter XOR outcome while odd flips do. This is nothing but Pr[Z1 XOR Z2 XOR Z3 XOR ... Zk = 0] because even flips cause XOR of Zi=1 to zero. If (1-2*delta) is probability of no flip and 2*delta is probability of a flip, for all Zi(s) Pr(XOR is correctly computed) = (1-2*delta)^k + 0.5*(1-(1-2*delta)^k). The second term is halved because probability of even number of flips is required whereas atleast 1 flip implies both (odd+even).

318.3 Pr(Good) majority voting boolean function computes composition of Majority function with Voting functions of individual voters. This can be expressed as: Majn(f1,f2,f3,...fn) for n voters. Till now hardness of this composition is not known in literature. Experimentally, following derivation computes hardness of this composition as below.

318.4 For simplicity, it is assumed that all voters have same boolean function with hardness delta computable by circuit size s.

318.5 Majority function makes error when the inputs are flipped by some correlation. This Probability that Pr(Maj(x) != Maj(y)) for two correlated strings x and y is termed as Noise Sensitivity. This together with the probability that a circuit with access to pseudorandom bits incorrectly computes majority function is an estimate of how flawed majority voting is(denoted as randomerror). Then probability that Majority function is correctly computed is 1-NoiseSensitivity(Majn) - randomerror . This implies majority function is in BPNC if there is a parallel voting.

318.6 Noise Sensitivity of Majority function is O(1/sqrt(n*epsilon)) where epsilon is probability of flip per bit. Therefore epsilon is nothing but probability that a voter boolean function is incorrectly computed and sent as input to Majority function. This implies Pr(f(x) != C(x)) = delta = epsilon =

hardness of all voter functions.

318.7 Probability that Majority function is computed incorrectly = NoiseSensitivity +/- randomerror = [c/sqrt(n*epsilon)] +/- randomerror = [c/sqrt(n*delta) +/- randomerror] which is the hardness of Majority+VoterFunctions composition. (Related: points 14, 53.14 and 355 for BP* error scenarios matrix which picturises overlap of Noise Sensitivity and Error)

318.8 Above derivation,if error-free,is the most important conclusion derivable for hardness of Pr(Good) majority voting boolean function circuit composition. Hardness of RHS of Pr(Good) Majority Voting circuit is expressed in terms of hardness of individual voter boolean functions.

318.9 Therefore from 318.7, Pr(C(Majn(f(x1),f(x2),...,f(n)) != Majn(f(x1),f(x2),...,f(n))] = [c/sqrt(n*delta) +/- randomerror] is the probability that how incorrectly a circuit of size s1 computes Majn+VoterFunction composition = hardness of majority+voter composition.

318.10 Hardness is amplified if [c/sqrt(n*delta) +/- randomerror] >= delta as follows:

$$\frac{\text{Hardness of Maj+voter composition}}{\text{Hardness of voter function}} = \frac{[c/\sqrt{n*\text{delta}}] +/- \text{randomerror}}{\text{delta}} >> 1$$

318.11 Let randomerror=r. From above, [c/sqrt(n*delta) +/- r] / delta must be >> 1 for hardness amplification. For large n, this limit tends to r/delta >> 1 implying error in majority circuit has to be huge for large electorate compared to error in voter boolean function SATs for hardness amplification. Caveat: There are scenarios where numerator is comparably equal to denominator and hardness amplification may not occur. From error scenarios matrix in 355,relation between error and noise can be precisely defined as: Error = Noise + (column2 error entries) - (column3 no error entries) where randomerror = r = (column2 error entries) - (column3 no error entries) which substituted in amplification becomes:

$$\frac{\text{Hardness of Maj+voter composition}}{\text{Hardness of voter function}} = \frac{[c/\sqrt{n*\text{delta}}] + [\text{sum(column2 error entries)}] - [\text{sum(column3 no error entries)}]}{\text{delta}} >> 1$$

For large n, c/sqrt(n*delta) tends to 0 and If [sum(column2 error entries) - sum(column3 no error entries)] >> delta then following applies:

318.12 318.11 implies that for weakly hard functions (low delta), if quantity in numerator (hardness of majority+voter composition) is considerably huge compared to hardness of individual voter functions, hardness is amplified .

318.13 318.11 also implies that Majority+Voter composition is extremely hard to compute concurring with exponential circuit size (PH=DC or EXP-completeness) of Pr(Good) RHS.

318.14 Being extremely hard implies that RHS of Pr(Good) could be one-way function. One-way functions are hard to invert defined formally as:

$$Pr(f(f\_inverse(y))=f(x)) \text{ is almost } 0.$$

For example inverse for Pr(Good) majority voting function is the one that returns set of all voters who voted for a candidate to win. This implies that any electoral process has to be one-way function hard so that secret balloting is not in jeopardy and finding such inverse for majority has to be hardest. Similarity of definition of one-way function and hardness of boolean functions is obvious. Hardness for majority derived previously is for forward composition direction which itself is high.

318.15 Conjecture: Circuit for reverse direction (i.e decomposition of

majority to voters who voted in favour) is considerably harder than composition and thus Majority(n)+Voter composition is an one-way function. An intuitive proof of this conjecture: Search for all permutations of voters who voted in favour which is of size O(2^n) - equivalent to solving SAT of Majority function boolean formula to find assignments which is counting problem and is #P-Complete. Indices of 1s in the assignment strings are the voter permutations. Next step is to find satisfying assignments to individual voter SATs which is also #P-Complete. This completely inverts the Majority+Voter composition and cumulatively is atleast #P-Complete. While the forward direction is composition of NC instance to voter SATs bottom-up, inverse is harder because it is completely #P-Complete top-down.

318.16 Huge hardness of Pr(Good) RHS also implies that very strong pseudorandom number generators can be constructed from it.

318.17 The counting problem in 318.15 finds all possible permutations of voters who could have voted in favour while what is required is the exact permutation which caused this majority outcome. From this, probability of finding exact permutation of voting pattern = 1/#SAT = 1/number_of_sat_assignments_to_MajoritySAT. This counting and search problem is defined as function Majorityinverse(1)=(v1,v2,v3,...vn) and returns the exact permutation of voters who voted for Majority outcome to be 1.

318.18 318.17 implies Pr(Majority(Majorityinverse(1))==Majority(v1,v2,...,vn)) = 1/#SAT <= 1/2^n in worst case.

318.19 In average case, expected number of SAT assignments to MajoritySAT = 1*1/2^n + 2*1/2^n + ...+ 2^n*1/2^n = 2^n*(2^n+1)/2 *1/2^n = (2^n+1)/2, assuming all permutations are equally probable. Average case Pr(Majority(Majorityinverse(1))==Majority(v1,v2,...,vn)) = 1/#SAT = 2/(2^n+1) ~= 1/2^(n-1) which is same as definition of a one-way function and thus Majority is hard to invert in average case.

318.20 318.19 implies Majority is one-way. Therefore FP != FNP and thus P != NP if hardness is amplified in 318.11.

References:

318.20 [Impagliazzo-Wigderson] - P=BPP if EXP has 2^Omega(n) size circuits - http://www.math.ias.edu/~avi/PUBLICATIONS/MYPAPERS/IW97/proc.pdf - Pr(Good) has exponential sized circuits (by theorem 6.29 of [AroraBarak] mentioned in 129 previously) because of unbounded nature of RHS.

318.21 [Trevisan] - XOR Lemma Course Notes - http://theory.stanford.edu/~trevisan/cs278-02/notes/lecture12.ps

318.22 [Goldreich-RaniIzsak] - One-way functions and PRGs - http://www.wisdom.weizmann.ac.il/~oded/PDF/mono-cry.pdf

318.23 Noise Sensitivity of Majority Function - http://pages.cs.wisc.edu/~dieter/Courses/2008s-CS880/Scribes/lecture16.pdf

318.24 Definition of One-way functions - https://en.wikipedia.org/wiki/One-way_function

---

319. (THEORY) Integer Partitions, Multiway Contests, Hash Table Functions and Ordered Bell Numbers - related to 256 and 272

Number of hash table functions derived in https://sites.google.com/site/kuja27/IntegerPartitionAndHashFunctions_2014.pdf, which sums
all possible permutations of hash buckets per partition does not consider order of elements with in each bucket. Bell number (named after
Eric Temple Bell) is the number of all possible partitions of a set (induced by all possible equivalence relations). Each hash
table is a set of equivalence classes (i.e each bucket is an equivalence class and hash function is the relation), partitioning the set.
Ordered Bell Number takes into account order of keys in each bucket and gives all possible Ordered Hash Table Functions. This is a stronger

estimate of number of hash table functions than plain counting from integer partitions which is unordered. Unordered n-th Bell Number is given by: sigma_k=0_to_n({n,k}) where {n,k} is the Stirling Number of Second kind computing number of ways of partitioning set of size n into subsets of size exactly k.

Let there be n keys and size of hash table be m. Let hi(x) mod m be a hash function.

```
For each hi
{
      For each x < n
      {
            Compute hi(x) mod m and append to corresponding chain of buckets in hash table
      }
}
```

Above loops create all possible partitions of set of size n. If just the integer partition is reduced to above hash table chains, sum(mP(lambda(i)) is only an approximation. Also Ordered Bell Number is an approximation as it does not permute each partition within a larger set of slots. Stirling Number of second kind {n,k} is equal to number of partitions of an n-element set into k nonempty subsets.Each partition is permuted within m slots in a separate chained hash table and each such permutation is a configuration of a hash function after all keys are populated. Total number of such functions = sum_k=1_to_m(mP({n,k}) which can be called as Augmented Stirling Number and is greater than Ordered Bell Number:

      sum_k=1_to_m(mP({n,k})) >= Ordered Bell Number = sum_k=1_to_m({n,k})

In multipartisan voting with more than 2 candidates, above augmented stirling number is also the number of all possible voting patterns with an electorate of size n voting on m candidates - each chain is the set of votes for corresponding candidate. Hash function for each voter chooses a candidate index. Thus Chained Hashing with optional sorting of chains on number of votes is Majority Function generalized. From Arrow, Friedgut-Kalai-Nisan and Gibbard-Satterthwaite theorems elections with more than 3 candidates can be manipulated implying multiway majority function has non-zero noise sensitivity.

Probability that above multiway majority function has an error (i.e chooses wrong outcome) =
Probability of changing one voting pattern to the other that changes ranking of candidates

Each voting pattern can be deemed to be a point vertex on an m-dimensional metric space and edges between these vertices are distances to other patterns. If each edge has a probability of occurrence, Probability of flip in voting pattern due to a malpractice is the probability of a path existing between these two voting patterns in this random graph. If each each edge has equal probability, path probabilities vary between p and p^(augmented_stirling_number). This assumes that any voting error is a converging random walk on the random graph of point vertices of voting pattern in a metric space. It could be a complete graph too which does not require a random walk.

In a sense the multiway majority hashing previously is a locality sensitive hashing wherein similarity between items is defined as voters voting for same candidate who are chained together in a bucket i.e Probability(two voters x and y voting for same candidate are colocated in a bucket) = 1 and Probability(two voters x and y voting for different candidates are colocated in a bucket) = 0 with strict equalities while the distance function is defined as d(x,y) <= r if x and y vote for same candidate and d(x,y) > r else. From m-balls and n-bins problem bounds, for m >> n, maximum number of balls per bin with high probability could be m/n + theta(sqrt(mlogn/n)) - in terms of votes, this is a rough estimate of maximum votes a candidate can get with high probability in

random voting. For example, a locality sensitive hash function f(x) mod m returning k for f(x) mod m with f(x) defined over [-l, +l] maps a ball of radius l comprising similar voters to a candidate k - ball is the chained bucket for k.

As mentioned in previous paragraphs, each voting pattern is a point vertex on a metric space and it is necessary to define distance d between any two voting patterns. Let v1 and v2 be two voting patterns in the chained hashing previously described for m candidates and n voters. These two patterns differ in votes per candidate. Hence it makes sense to define the distance between the patterns as number of votes gained/lost by candidates across these two patterns. Let p be the probability that single vote is transferred by rigged voting from candidate k to candidate l. Also let c1, c2, c3,..ck be the candidates who either all gained (or) all lost across voting patterns v1 and v2. Then the probability that voting pattern changes from v1 to v2 = p^d where d is the metric distance between two voting patterns defined as d = v2(c1) - v1(c1) + v2(c2) - v1(c1) + ...+ v2(ck) - v1(ck) where va(cb) is the votes for candidate cb in voting pattern va. L2 norm is not preferred because probability for voting pattern change requires gained/lost votes in one direction only. For example voting pattern {5,3,4} becoming {6,4,2} implies first 2 candidates gained 1 vote each while third candidate lost 2 votes. L2 norm is sqrt(6) while metric distance as per previous definition is 2. Greater the distance between patterns, exponentially rarer is the probability p^d of voting pattern flip. It is assumed that all voters vote independently and each vote flip in pattern change occurs independently. Thus the random graph edges between voting pattern vertices are weighted by these probabilities(p^d1, p^d2, etc., for distances d1,d2,...).This is an alternative definition of edge probability different from one described previously. Number of vertices in this random graph = augmented_stirling_number = sum_k=1_to_m(mP({n,k})) >= Ordered Bell Number

Probability of multiway majority choosing a wrong outcome = Probability that voting pattern noise flip changes sorted ranking of candidates. For example, {12,10,5} becomes {7,12,8} which changes winner from first to second candidate with total votes remaining same. Not all edges of random graph of patterns change the ranking despite noise. If majority in multiway contest is defined as candidate index getting more than half of total votes, winning voting patterns are number of integer partitions of n voters with largest part > n/2. These partitions with largest part greater than n/2 correspond to subset of voting patterns which are not affected by noise flip because only parts other than largest are shuffled. Therefore probability that outcome is not changed by noise flip of voting pattern = number_of_partitions_of_n_voters_greater_than_n/2 / partition_number(n). From this probability of wrong outcome because of noise flip in voting pattern = 1 - { (number_of_partitions_of_n_voters_greater_than_n/2) / partition_number(n) } = Noise sensitivity of Multiway Majority Function.

319.8 is an NVIDIA CUDA parallel implementation of hash table chaining. 319.9 is a recent parallel locality sensitive hashing algorithm for multicore machines, and segregates twitter stream of tweets into similar buckets. Availability of parallel algorithms for hash table chains implies that hash chaining and LSH especially could be in NC (logarithmic table construction time) though there is no known result thus far. Parallelism in multiway majority function LSH is obvious because people vote in parallel and hash chain buckets are populated in parallel across multiple voting machines in compartmentalized voting. Each such voting machine can correspond to a node in Multiway Majority circuit. The rho parameter of LSH is log(1/p1)/log(1/p2) where p1=Pr(h(x)=h(y)) if x and y had voted for same candidate and p2=Pr(h(x)=h(y)) if x and y had voted for different candidates. In multiway majority LSH mentioned previously, p1=1 and p2=0 and rho=0 where as in usual approximate neighbours problem rho is only required to be < 1. Intuitively, each hash table slot is lock-free and each bucket for a slot is lock-synchronized. Thus votes across candidates is parallelizable and votes for a candidate are serially incremented(linked-list bucket is appended). Addition of votes per candidate slot should be doable in NC (because integer addition is in NC). Thus there are as many NC circuits as there are candidates. Set of votes from all candidates has to be then sorted to find the winner which is also in NC (e.g Sorting networks). Optional comparator circuits for half-way

mark comparison is also in NC. Thus Multiway majority function could be computable in Non-uniform NC which places it in same league as boolean 0-1 majority function.

References:

319.1 Unordered Bell Number - https://en.wikipedia.org/wiki/Bell_number
319.2 Ordered Bell Number - https://en.wikipedia.org/wiki/Ordered_Bell_number
319.3 Mining Massive Data Sets - [UllmanRajaramanLeskovec] - http://infolab.stanford.edu/~ullman/mmds/book.pdf
319.4 Lowerbounds for Locality Sensitive Hashing - [RyanODonnell] - https://www.cs.cmu.edu/~odonnell/papers/lsh.pdf
319.5 Theory of Partitions - Bell numbers - [GeorgeEAndrews] - http://plouffe.fr/simon/math/Andrews%20G.E.%20The%20Theory%20of%20Partitions%20(Enc.Math.Appl.%202,%20AW,%201976)(266s).pdf
319.6 Power of Simple Tabulation Hashing - [PatrascuThorup] - http://arxiv.org/pdf/1011.5200v2.pdf - Lemma 4 - Balls and Bins hashing
319.7 Balls and Bins, Chained Hashing, Randomized Load balancing - http://pages.cs.wisc.edu/~shuchi/courses/787-F09/scribe-notes/lec7.pdf
319.8 Parallel Hash tables - Chaining buckets with arrays - http://idav.ucdavis.edu/~dfalcant//downloads/dissertation.pdf
319.9 Parallel Locality Sensitive Hashing - [Narayanan Sundaram†,Aizana Turmukhametova,Nadathur Satish†,Todd Mostak,Piotr Indyk,Samuel Madden and Pradeep Dubey†] - http://istc-bigdata.org/plsh/docs/plsh_paper.pdf

---
320. (BUG-STABILITY ISSUES) Commits - 6 September 2016

VIRGO Linux Kernel Stability Analysis - 2 September 2016 and 6 September 2016
---
kern.log(s) for Boost::Python invocation of VIRGO system calls with and without crash on two dates have been committed:
Logs on 2/9/2016 have a crash as usual in VM paging scatterlist.c.Today's invocation logs show perfect execution of whole end-to-end user/kernel space without any kernel panic much later. Such randomness in behaviour is quite unexplainable unless something lurking beyond the code path intercepts this. Could be a mainline kernel bug in 4.1.5 tree.

---
321. (BUG-STABILITY ISSUES) Commits - 8 September 2016

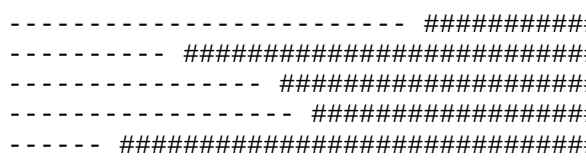
Continued kernel panic analysis of boost::python-VIRGO system calls invocations. Similar pattern of crashes in vma paging is observed with a successful panic-free invocation in the end.

---
322. (BUG-STABILITY ISSUES) Commits - 9 September 2016

Boost::python-VIRGO system calls invocation kernel panic analysis showing some problem with i915 graphics driver interfering with VIRGO system calls.

---
323. (THEORY) Complement Functions for Hash Chains, PAC Learning, Chinese Remaindering and Post Correspondence Problem - related to 19,24,319
- important draft additions to http://arxiv.org/pdf/1106.4102v1.pdf

---
A hash table chain can be viewed as a subset segment of a larger rectangular region. Complement of this subset is another hash table chain.
For example following schematic ilustrates hash table complements with respect to chained buckets:



Above diagram has two hash table chains shown in different constituent colors. Together these two cover a rectangular region. Thus left(f1) and right(f2) are complements of each other. In partition parlance, both f1 and f2 have same number of parts. Let f1 partition a set of n1 and f2 partition a set of n2 elements into chains. Let m be the number of parts in both (breadth of rectangle). Then length is (n1 + n2) / m. If Pi(f1) and Pi(f2) denote the i-th chain partition on both f1 and f2, then Pi(f1) + Pi(f2) = (n1 + n2)/m. Hash function which created f1 is known while that of f2 is unknown. This is a special case of Decidability of Complement Function mentioned in 24 where "complementation" is defined with reference to a larger universal rectangular set. From the chain pattern on right the hash function for f2 can be reverse engineered. When the left is viewed as a multiway majority voting pattern or a Locality Sensitive Hash function, right is a complementary voting pattern or a Complementary Locality Sensitive Hash Function which "inverts" the ranking. A tabulation for hash function of f2 can be constructed like an example below:

```
f(x01) mod m = 0
f(x11) mod m = 1
f(x12) mod m = 1
...
f(x21) mod m = 2
f(x22) mod m = 2
...
```

where xij is the j-th element in i-th chain part in f2 on right. Above tabulation can be solved by Euclid's algorithm. For example, f(x11) mod m = 1 can be solved as:

```
[-m*y + f(x11)] = 1 => f(x11) mod m = 1
```

Therefore above table can be written as system of congruences:

```
f(x01) = a01*m + 0
f(x11) = a11*m + 1
f(x12) = a12*m + 1
...
f(x21) = a21*m + 2
f(x22) = a22*m + 2
...
```

Since m is known and all aij can take arbitrary values, complement function for f2 can be constructed by fixing aij and applying either PAC learning(approximation only) or CNF multiplexor construction/Interpolation/Fourier Series (described in 19 and 24). There are indefinite number of Complement Functions constructible by varying aij. But Number of hash functions is upperbounded by Augmented Stirling Number. This is because hash functions which internally apply complement functions, create finite chained bucket configurations or voting patterns bounded by augmented stirling number modulo size of the table whereas complement functions don't have such restrictions in construction. This implies two complement functions could give rise to similar hash table chain configurations. Thus set of complement functions is partitioned by hash chained configurations i.e. there are augmented stirling number of sets of complement functions. Main advantage of PAC learnt complement construction over exact multiplexed CNF construction is there are no bloating of variables and boolean conjunctions are minimal, but the disadvantage is approximation and not exact. For first n prime numbers there are log(n) conjunctions each of log(n) literals.

Boolean 0-1 majority function is a special case of multiway majority function (or) LSH, where hashtable is of size 2 with 2 chained buckets i.e.

Augmented Stirling Number of Boolean Majority = 2P(n,1} + 2P{n,2}

Chinese Remaindering Theorem states that there is a ring isomorphism for N=n1*n2*n3*....*nk (for all coprime ni) such that:

X mod N <=> (X mod n1, X mod n2, X mod n3, ..., X mod nk)

(or)

Z/NZ <=> Z/n1Z * Z/n2Z * .... * Z/nkZ

Chinese Remaindering has direct application in hash table chains as they are created in modular arithmetic. K hash functions on the right of isomorpshism correspond to K hash table chains (or) LSH (or) Voting patterns for hash tables of sizes n1,n2,....,nk. Left of isomorphism is a hash table of size N (LSH or a Voting Pattern).

From Post Correspondence Problem, if f1 and f2 in schematic diagram are two voting patterns, finding a sequence such that:

p1p2p3...pm = q1q2q3....qm

is undecidable where pi and qi are parts in voting patterns f1 and f2 as concatenated strings. What this means is that finding a sequence which makes serialized voters in both patterns equal is undecidable.

Alternative proof of undecidability of Complement Function Construction with PCP :

Complement Functions are reducible to Post Correspondence Problem. PCP states that finding sequence of numbers i1,i2,i3,...,ik such that:

s(i1)s(i2)s(i3)....s(ik) = t(i1)t(i2)t(i3)...t(ik)

where s(ik) and t(ik) are strings is undecidable.

Example inductive base case:

Let f(x) = 2,4,6,8,....
and g(x) = 1,3,5,7,....
f(x) and g(x) are complements of each other.

Define a(i) and b(i) as concatenated ordered pairs of f(xi) and g(xi):

a1 = 2 b1 = 2,3
a2 = 3,4 b2 = 4,5
a3 = 5,6 b3 = 6,7
a4 = 7,8 b4 = 8,9
a5 = 9,10 b5 = 10

[e.g a2 = 3, 4 where g(a2) = 3 and f(a2) = 4; b3= 6,7 where f(b3) = 6 and g(b3) = 7 and so on. Here complement of an element is the succeeding element in universal set Z=1,2,3,4,5,...]

Then, 1,2,3,4,5 is a sequence such that:

a1a2a3a4a5 = b1b2b3b4b5 = 2,3,4,5,6,7,8,9,10

which can be parenthesised in two ways as:

2(3,4)(5,6)(7,8)(9,10) = (2,3)(4,5)(6,7)(8,9)10 = 2,3,4,5,6,7,8,9,10

Each parenthesization is a string representation of a possible way to construct a complement function. Ideally, process of complementation has two symmetric directions created by the gaps in the range a function maps a domain to - predecessors and successors - which is captured by the two parenthesizations(i.e f is complement of g and g is complement of f). Each substring within the parentheses is indexed by sequence numbers ai and bi. Thus an order of sequence numbers have to be found for both symmetric directions to get matching concatenated union of complements (universal set). Both directions have to converge because complementation is symmetric relation. But finding such an order of sequence numbers is a Post Correspondence Problem and undecidable(question of if turing machine halts on input). Since each a(i) and b(i) are concatenated ordered pair of value of f(x) and its complement g(x),

complement function is constructible if such ordered pairs exist for indefinite length. In essence: Finding an order of sequence numbers = Construction of complement functions which converge in opposing directions => Undecidable by Post Correspondence Problem. Above example can be generalized for any function, and its complement ordered pair and thus is an alternative proof (with a tighter definition of complementation that complements are bidirectional) of undecidability of Complement Function Existence and Construction. Another example is 2 parenthesizations for prime complementation below - elements succeeding a prime in left and elements preceding a prime in right:

f(x) = 2,3,5,7,11,...
g(x) = 4,6,8,9,10,12,...

([2,3],4)(5,6)(7,[8,9,10])(11) = ([2,3])(4,5)(6,7)([8,9,10],11) = 2,3,4,5,6,7,8,9,10,11

Generic inductive case:

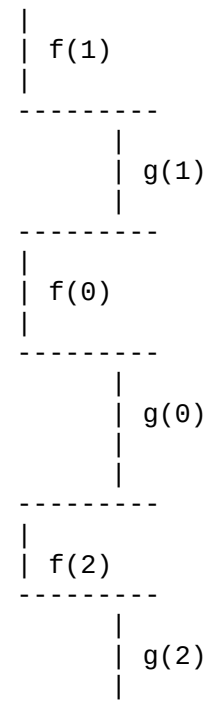
(f(x1),succ(f(x1))(f(x2),succ(f(x2))).... = (predec(g(x1),g(x1))(predec(g(x2),g(x2)).... = Z

(or)

concatenation_of(f(xi),g(xi)) = concatenation_of(g(xi),f(xi)) = Z

where succ() and predec() are successor and predecessor functions respectively in lambda calculus jargon and xi take some integer values. This assumes nothing about the sorted order of f(xi) and g(xi) and each ordered pair is arbitrarily sampled and aggregated. f(x) is known and from this complement g(x) has to be constructed(though g(xi) sampled data points are known, g(x) is not). To construct g(x), ordered pairs for g(xi) have to be sequenced as g(0), g(1), ... which amounts to assigning values to xi which is Post Correspondence Problem and undecidable.

Two complementation parenthesizations can be drawn as step function where indentation denotes complementary g points to f in example below:



which represent two parenthesization correspondences (sequence on right is staggered by one segment):

(f(1),g(1))(f(0),g(0))(f(2),g(2)) = (f(1))(g(1),f(0))(g(0),f(2))(g(2)) = f(1)g(1)f(0)g(0)f(2)g(2)

Hence x1,x2,x3 take values 1,0,2 for string on left. String on the right despite staggering also takes values 1,0,2 for x1,x2,x3 with only slight change - 1,0,2 is represented as 1-0, 0-2, 2 by coalescence of any two juxtaposed xi. This

correspondence is undecidable by PCP. This complementation scheme relaxes the definition of complements in http://arxiv.org/pdf/1106.4102v1.pdf by allowing the pre-image of complementary function segments to be in unsorted order - e.g 1,0,2 is not sorted.

Two Generic parenthesizations for f and its complement g are:
[f(x1)][g(y1)f(x2)][g(y2)f(x3)][g(y3)]... = [f(x1)g(y1)][f(x2)g(y2)] [f(x3)g(y3)]... = f(x1)g(y1)f(x2)g(y2)f(x3)g(y3)...
where each f(xi) and g(yi) are contiguous streak of set of values of f and g. xi is not necessarily equal to yi. Required sequence numbers are x1-y1, x2-y2, x3-y3, ... in right and x1, y1-x2, y2-x3, ... in left to make the strings correspond to each other. Here sequence numbers are identifiers which are concatenation of 2 numbers xi and yi as xi-yi or yi-x(i-1). Thus original undecidability of finding x1-y1,x2-y2,x3-y3,... still remains. This is the most relaxed version of complementation.

Another alternative definition of complement function is: function f and its complement g are generating functions for the exact disjoint set cover of size 2 of a universal set formed by values of f and g. Disjoint Set Cover is the set of subsets of a universal set where each element of universal set is contained in exactly one set and their union is the universal set. Notion of complementation can be extended to arbitray dimensional spaces. Each half-space is generated by a function complement to the other. Complementation is also related to concepts of VC-Dimension and Shattering where each half-space is a class classified by complement functions (set A is shattered by a set C of classes of sets if for all subsets s of A, s = c intersection A for c in C)

Complementation Disjoint Set Cover can be represented as bipartite graph where edges are between two sets A and B. Set A has sequence numbers xi(or yi) and set B is the universal set (Z for example). For each f(xi) and g(yi) there is an edge from xi----f(xi) and yi-----g(yi).

Boolean 0-1 majority special case of Hash chains and Complement Functions:

A hash table of size 2 with 2 chains partitions the set of keys into 2 disjoint sets. Each chain in this hash table is created by 2 functions complement to each other (examples previously described: set of odd and even integers, set of primes and composites). Thus if there exists a hash function h that accepts another function f as input and partitions a universal set into two chained buckets with mutually complementary elements belonging to f and g(complement of f), then it is an indirect way to construct a complement. But undecidability of complement construction by post correspondence precludes this.

---
324. (FEATURE-DONE) PAC Learning for Prime Numbers encoded as binary strings - Commits - 12 September 2016

PAC Learning implementation has been augmented to learn patterns in Prime Numbers encoded as binary strings. For each prime bit a boolean conjunction is learnt. Separate Mapping code has been written in python-src/PACLearning_PrimeBitsMapping.py which json dumps the mapping of first 10000 prime numbers to corresponding i-th prime bit. Logs for this have been committed to python-src/testlogs/PACLearning.PrimeBitsMappingConjunctions.out.12September2016

---
325. (FEATURE-DONE) PAC Learning for Prime Numbers - Commits - 14 September 2016

Some errors corrected in bit positions computation in JSON mappings for PAC learning of Prime Numbers. Logs committed to testlogs/

---
326. (BUG-STABILITY ISSUES) Boost::Python-VIRGO System calls invocations random kernel panics - Commits - 15 September 2016

Further Kernel Panic analysis in i915 driver for Boost::python-VIRGO system calls invocations.

---
327. (THEORY) Algorithmic Fairness, Pr(Good) Majority Voting Circuit and Algorithmic Decision Making - Related to 14,53,275,317

In Majority Voting hardness analyzed thus far, voting functions of Individual Human Voters are assumed. Recent advances in algorithmic decision making (High Frequency Algorithmic Trading, Predictive Policing etc.,) involve decision making by algorithms than humans. It was implicit so far that algorithms cannot have bias. But there is a new emerging field of algorithmic fairness which highlights growing bias by machine learning algorithms in decision making (bias could be in training dataset, algorithm's false assumptions leading to wrong conclusions based on correlations etc.,) that could subvert stock trading buy-sell decisions, criminal justice system and so on. Similar unfairness could happen in Majority voting also if the voters are algorithms (algorithm internally using a boolean function, non-boolean function, past training data to make future decisions among others). Unfair voting algorithms imply that Pr(Good) summation would never converge to 100% and therefore error is non-zero. Unfair voting is detrimental to distributed cloud computing involving majority voting choice - e.g loadbalancing of requests get skewed to a node unfairly by bad voting. Most importantly a proof of existence of 100% fair voting algorithm implies that LHS of Pr(Good) summation is 1 (From 317.2 least error algorithm is chosen as Non-majority social choice).

References:

327.1 Algorithmic Fairness - https://algorithmicfairness.wordpress.com
327.2 Leader Election Algorithms in Cloud - HBase ZooKeeper - [Mahadev Konar - Yahoo] - http://wiki.apache.org/hadoop/ZooKeeper/ZooKeeperPresentations?action=AttachFile&do=view&target=zookeeper_hbase.pptx
327.3 Leader Election and Quorum in ElasticSearch - https://www.elastic.co/blog/found-leader-election-in-general - Each node votes for a leader and minimum number of votes required for a leader is Quorum - This is realworld application of Multiway Majority function.

---
328. (FEATURE-DONE) Locality Sensitive Hashing Implementation - Nearest Neighbours Search - Commits - 16 September 2016

1.This commit implements locality sensitive hashing in python by wrapping defaultdict with hashing and distance measures.
2.Locality Sensitive Hashing is useful for clustering similar strings or text documents into same bucket and is thus an unsupervised classifier and an inverted index too.
3.In this implementation, very basic LSH is done by having replicated hashtables and hashing a document to each of these hashtables with a random polynomial hash function which itself is aggregation of random monomials.
4.For a query string, random hash function is again computed and buckets from

all hashtables corresponding to this hash value is returned as
nearest neighbour set.
5.Each of these buckets are sieved to find the closest neighbour for that
hashtable.
6.Sorting the nearest neighbours for all hash tables yields a ranking of
documents which are in the vicinity of the query string.
7.The input strings are read from a text file LocalitySensitiveHashing.txt
8.Logs with hashtable dumps and nearest neighbour rankings are in
testlogs/LocalitySensitiveHashing.out.16September2016

---
329. (FEATURE-DONE) LSH WebCrawler Support - Commits - 19 September 2016

Locality Sensitive Hashing now accepts scrapy crawled webpages as datasources.

---
330. (BUG - STABILITY ISSUES) Boost::Python AsFer-VIRGO system calls kernel
panics - Commits - 19 September 2016

Boost::python AsFer - VIRGO system call ongoing kernel panic analysis - i915 DRM
race condition kernel panic in VM pages freeing

---
331. (FEATURE-DONE) ZeroMQ based Concurrent Request Servicing CLI - Client and
Multithreaded Server Implementation

Commits - 21 September 2016

1.NeuronRain already has support for RESTful GUI implemented in Python Tornado
and NeuronRain code can be executed by
filling up HTML form pages.
2.ZeroMQ has a lowlevel highly performant low latency concurrency framework for
servicing heavily concurrent requests.
3.ZeroMQ is a wrapper socket implementation with special support for Request-
Reply, Router-Worker, Pub-Sub design patterns.
4.Important advantage of ZeroMQ is lack of necessity of lock synchronization
(i.e ZeroMQ is lock-free per its documentation)
for consistency of concurrent transactions
5.Hence as a CLI alternative to HTTP/REST GUI interface,a C++ client and server
have been implemented based on ZeroMQ Request-Reply
Router-Dealer-Worker sockets pattern to serve concurrent requests.
6.ZeroMQ client: ./zeromq_client "<neuronrain executable command>"
7.ZeroMQ server: invokes system() on the executable arg from zeromq client
8.With this NeuronRain has following interfaces:
        8.1 telnet client --->
NeuronRain VIRGO ports
        8.2 VIRGO system call clients -->
NeuronRain VIRGO ports
        8.3 AsFer boost::python VIRGO system call invocations ---------------->
NeuronRain VIRGO ports
        8.4 Tornado GUI RESTful -->
NeuronRain VIRGO ports
        8.5 ZeroMQ CLI client/server --->
NeuronRain VIRGO ports

---

332. (FEATURE-DONE) KingCobra VIRGO Linux workqueue and Kafka Publish-Subscribe Backend Message Queue support in Streaming Generator
---
Commits - 22 September 2016

1.Streaming_AbstractGenerator has been updated to include KingCobra request-reply queue disk persisted store, as
a data storage option (reads /var/log/kingcobra/REQUEST_REPLY.queue). Initial design option was to integrate a Kafka client into
KingCobra kernelspace servicerequest() function which upon receipt of a message from linux kernel workqueue,also publishes it in turn
to a Kafka Topic. But tight coupling of Kafka C Client into KingCobra is infeasible because of conflicts between userspace include header
files of Kafka and kernelspace include header files of Linux. This results in compilation errors. Hence it looks sufficient to read the
persisted KingCobra REQUEST_REPLY.queue by a standalone Kafka python client and publish to Kafka subscribers. This leverages analytics
by python code on KingCobra queue.
2.NeuronRain backend now has a support for Kafka Pub-Sub Messaging.
3.New publisher and subscriber for Kafka (with Python Confluent Kafka) have been written to read data from
Streaming_AbstractGenerator and to publish/subscribe to/from a Kafka Message Broker-Topic. Thus any datasource that
AsFer may have(file,HBase,Cassandra,Hive etc.,) is abstracted and published to Kafka. This also unites AsFer backend
and KingCobra disk persistence into a single Kafka storage.

---
333. (BUG-STABILITY ISSUES) Boost::Python AsFer-VIRGO system call invocations kernel panic ongoing investigation
---
Commits - 23 September 2016

Ongoing Boost::Python AsFer-VIRGO system call invocation kernel panic analysis: Random crashes remain, but this time no crash logs are printed in kern.log and finally a successful invocation happens. It could be
same as i915 GEM DRM crash similar to earlier analyses.

Pattern observed is as follows:
	1. First few invocations fail with virgo_get() though virgo_malloc() and virgo_set() succeed.
	2. After few failures all virgo calls succeed - virgo_malloc(), virgo_set() and virgo_get() work without any problems.
	3. Sometimes virgo_parse_integer() logs and few other logs are missing. When all logs are printed, success rate is very high.
	4. Some failing invocations have NULL parsed addresses. When there are no NULL address parsings, success rate is very high.
	5. a rare coincidence was observed: Without internet connectivity crashes are very less frequent. (Intel Microcode
updates playing spoilsport again?).
	6. There have been panic bugs reported on i915 GEM DRM and recent patches for busy VMA handling to it:
		6.1 https://bugs.launchpad.net/ubuntu/+source/linux/+bug/1492632
		6.2 https://lists.freedesktop.org/archives/intel-gfx/2016-August/102160.html
	7. Intel GPU i915 GEM DRM docs - https://01.org/linuxgraphics/gfx-docs/drm/gpu/drm-mm.html

---

---
334. (BUG-STABILITY ISSUES) Boost::Python AsFer-VIRGO systemcall/drivers invocations kernel panics - new findings
---
Commits - 28 September 2016

Continued analysis of kernel panics after VIRGO systemcalls/drivers code in i915 GPU driver. Has hithertoo unseen strange OOM panic stack dumps in GPU memory after kmalloc() of 100 bytes in virgo_cloud_malloc() and set/get of it. Logs with panic stack dumps and High and Low Watermark
memory details have been committed to cpp_boost_python_extensions/testlogs/.

---
335. (BUG-STABILITY ISSUES) Boost::Python AsFer-VIRGO systemcall invocations panics - more findings
---
Commits - 29 September 2016

Kernel Panic Analysis for Boost::Python AsFer - VIRGO system calls invocations: This log contains hitherto unseen crash in python itself deep within kernel (insufficient logs) followed by random -32 and -107
errors. Finally successful invocations happened. Connections between -32,-107 errors and random panics/freezes were analyzed few years
ago (Blocking and Non-blocking socket modes). i915 DRM related stacks were not found in kern.log. Random disappearance of
logs is quite a big travail. Crash within python could be i915 related - cannot be confirmed without logs. Pattern emerging is that
of: Something wrong going on between CPU and GPU while allocating kernel memory in CPU domain - kmalloc() is likely allocating from
GPU and not CPU - quite a weird bug and unheard of.

---
336. (BUG-STABILITY ISSUES) VIRGO kernel panics - final findings - 30 September 2016

Further kernel panic investigation in VIRGO - probably the last. Logs with analysis have been committed to cpp_boost_python_extensions/
testlogs/.

---
337. (THEORY) Ramsey Theorem, Edge Labelling of Voting Graph and Multiway Majority Function - 5 October 2016 - related to 256,272,319

Multiway majority voting can be drawn as a directed graph. There is an edge between vertices v1 and v2 if v1 votes for v2 forming a
Voting Graph with a weight >= 0. Realworld example of this is web link graph where incoming links to a webpage are votes to it and
outgoing links from a webpage are votes for adjacent pages. PageRank is a special case of Multiway Majority Function which ranks the
candidate webpages by a converging random walk markov chain of transition probabilities, with a rider that all webpages are both voters
and candidates making it a peer-to-peer majority voting. PageRank is thus a Non-boolean voting function. If web link graph is a tree
of depth d, it is equivalent to depth-d recursive majority function. Edge labelling of a graph assigns colors to edges of a graph

(Edge coloring is a special case of labelling with restriction no two adjacent edges are of same color). Voting graphs can be edge
labelled where each color of an edge denotes a voter affiliation. Ramsey Number in combinatorics states that there exists a number
v=RN(r,s) for every graph of order v, there exists a clique of size r or independent set of size s. For any Voting graph Ramsey Theorem
implies emergence of a clique or an independent set i.e voters who vote among themselves or who do not vote for each other. If the Voting
graph is complete then Ramsey Theorem implies emergence of monochromatic cliques r (red cliques) or s (blue cliques).

References:

337.1 Ramsey Theorem Lecture Notes - http://math.mit.edu/~fox/MAT307-lecture05.pdf
337.2 Ramsey Theorem - https://en.wikipedia.org/wiki/Ramsey%27s_theorem

---

338. (THEORY) Van Der Waerden Number, Schur, Szemeredi and Ramsey Theorems, Coloring of Integers and Complement Function - Related to 323 - important draft updates to http://arxiv.org/pdf/1106.4102.pdf

---

Complement Function over Integer sequences can be defined in terms of 2-colorings of the integers. Described previously in (323) PCP
undecidability proof of complementation, a function f and its complement g can be construed as 2-colorings of the Disjoint Set Cover Union i.e
the set of natural numbers - each color is a function - for example f is red and g is blue.

Schur's Theorem for Ramsey coloring of integer sequences states (quoted from http://math.mit.edu/~fox/MAT307-lecture05.pdf):

" ... Schur's theorem
Ramsey theory for integers is about finding monochromatic subsets with a certain arithmetic structure. It starts with the following theorem of Schur (1916), which turns out to be an easy application of Ramsey's theorem for graphs.

Theorem 3.
For any k >= 2 , there is n > 3 such that for any k-coloring of 1,2,3,...,n, there are three integers x, y, z of the same color such that
x + y = z ... "

For complement function special case (i.e 2-coloring of the sequences), Schur Theorem implies that there are always integers x,y and z in the image of a function or its complement obeying x+y=z. Van Der Waerden Therem for coloring integers is the variant of Ramsey Theorem for graphs (quoting https://en.wikipedia.org/wiki/Van_der_Waerden%27s_theorem):

" ... for any given positive integers r and k, there is some number N such that if the integers {1, 2, ..., N} are colored, each with one of r different colors, then there are at least k integers in arithmetic progression all of the same color. ... "

Van Der Waerden Theorem for 2-colorings of natural numbers is equivalent to complementation disjoint set cover of natural numbers. When this is a prime complementation special case, prime integers are colored in red and composites in blue. Thus Van Der Waerden theorem for prime-composite complementation implies that either set of primes or composites have arithmetic progressions (because of same coloring) of size atleast k (Related: Green-Tao theorem states that primes have arbitrarily long arithmetic progressions). Upperbound for N in this 2-coloring prime complementation is deriveable from [TimothyGowers] bound as: N <= 2^2^2^2^(k+9) for a family of k sized arithmetic progressions (also

called as k-regularity). This implies arbitrarily long monochromatic arithmetic progressions can be found in prime-composite complementation by choosing N and k.

Finding the arithmetic progressions for Van Der Waerden numbers have been formulated as SAT instances which involves a SAT solver to search for a satisfying monochromatic arithmetic progression. Similar SAT solving approach applies also to prime-composite bichromatic complementation for finding arithmetic progressions. This takes the notion of complementation a fine grained step further in the sense: prime-composite complementation is a pattern miner in primes and arithmetic progressions in prime-composite complementation are patterns within pattern.

Concept of complement graphs have been studied in Perfect Graph Theorem and has strong resemblance to function complementation. Complement graph H is obtained from a graph G by adding edges to make it complete graph and removing edges in G i.e H has edges which are not in G and vice versa. Function complementation is precisely a sequence corollary of Graph complementation. Perfect Graph Theorem (PGT) states that a graph is perfect if its complement is perfect where perfect graph is one in which all its induced subgraphs have chromatic number (number of colors required to k-color a graph) equal to size of maximum clique. Function complementation can be translated to Graph complementation by defining a relation: There is an edge in F, f(x1)----f(x2) for every f(x1) and f(x2) and there is an edge in G, g(x1)----g(x2) for every g(x1) and g(x2) where f and g are mutual complements. Obviously F and G have a single maximum clique with an independent set without any edge amongst them. Then it follows that order of F and G are their respective Ramsey Numbers.

A contrived independent set can be created by adding edges from constituent vertices of the maximum clique which is also maximal such that there are no edge overlaps with its complement graph - thus every graph for a function and its complement has a single maximum and maximal clique and an independent set connected only to the clique with no edges among them i.e. Vertices of the clique form vertex cover of the graph

Any 2 graphs corresponding to 2 functions defined on same set obtained from previous construction are isomorphic. For example if a complete graph of 3 vertices (K3) is constructed from elements 2,3,5 for function f1 where f1 is defined as f1(0)=2, f1(1)=3 and f2(2)=5, the vertices can be numbered by the inverse(f1) as 0,1,2. Similarly for a different function f2 over 2,3,5 K3 obtained can be vertex numbered by the inverse(f2) where f2(0)=5, f2(1)=2 and f3(2)=3 where numbering is 1,2,0. This also implies two graphs for function f and its complement g are not isomorphic.

In the context of complement function graphs constructed previously, functions are mapped to maximum clique subgraph and only this subgraph is vertex-renumbered which is the problem of subgraph isomprphism to find out if clique subgraph is isomorphic to another function clique subgraph.

Szemeredi's Theorem which is the generalization of Van Der Waerden's Theorem states: For any subset A of N of natural density > 0, there are infinitely many arithmetic progressions of size atleast k where natural density is defined as lim {n->infinity} sup |A intersection {1,2,3,...,n}|/n > 0. Green-Tao theorem applies a relative Szemeredi theorem because prime numbers have 0 natural density. Szemeredi's Theorem has been generalized to polynomial progressions where the polynomial progression is defined as [t(i) = t(i-1) + p(i)] in A for some integer valued polynomials p(i). Thus Generalized Szemeredi Theorem for Finding polynomial arithmetic progressions in a subset of natural numbers is closest to finding the complement of a function when set of natural numbers is 2-colored - only a subset of complement set is expressed as polynomials whereas complement function requires, by definition, the complete set to be expressed as polynomials (e.g interpolation polynomials, fourier polynomials).

Roth Estimate for 2-coloring of integer sequences (in references 338.4 and

338.5) is a measure of order or randomness in a colored sequence and hence in a complementation by functions f and g (where each function is a color - e.g f is blue and g is red) over a universal set. If arithmetic progressions are thought of as all possible "sampled points" on 2-colored integers (functions f and g) i.e an approximation of f and g,then Roth estimate is a measure of difference in size of f(x) and g(x) with probability equal to natural density of arithmetic progression chosen in minimax() step. This is because natural density is a fraction of size of set of elements of an arithmetic progression on natural numbers - in other words, natural density is the size of a sample of f and g. From Roth estimate bounds, |{f(x)}| - |{g(x)}| is approximately N^(1/4+epsilon) with probability equal to natural density of arithmetic progression in minimax() step and |{f(x)}| + |{g(x)}| is N with probability 1. This implies:

|{f(x)}| = {N+N^(1/4+epsilon)} / 2

|{g(x)}| = {N-N^(1/4+epsilon)} / 2

with probability equal to natural density of arithmetic progression in minimax()

Previous approximation with sampling relates size of complement functions,coloring and arithmetic progressions by Roth estimate.

Complement Graphs F and G constructed previously for function f and its complement g have set of all values of inverse(f) and inverse(g) as the respective vertex labels in their maximum monochromatic cliques. This maps a 2-colored integer sequence with each color representing a function to 2 graphs, mutually complement, each with a maximal monochromatic clique for respective function (f or g) and an independent set of other color. This construction adds one more dimension to the coloring problem: numbering of vertices for either color and subgraph isomorphism. From Van Der Waerden and later newer theorems, this implies numbered vertices in F and G have atleast one monochromatic arithmetic progression.

Another important question is: Can a complement function polynomial be constructed from constituent monochromatic arithmetic progressions (or) Is the complement function polynomial interpolated by Lagrange Theorem or Fourier Analysis a composition of constituent arithmetic progressions? This is answered in reference 338.11 - Interpolation Polynomial for n points can be approximated from low discrepancy arithmetic progressions. References 338.12 and 338.13 are related to Polynomial Sequences of length n (P(n) - Polynomial Points) represented by rings of polynomials F(x) with integer coefficients in Zn and for each sequence S(k)=(a1,a2,...,an) there exists a polynomial f(x) in F(x) such that f(i)=ai,i=1,2,3,...,n (f(x) is a Lagrange Interpolation Polynomial). For example, monochromatic arithmetic progressions from Van Der Waerden and other theorems are polynomial sequences represented by arithmetic progression polynomial points. Thus Complement Function Polynomial interpolated (Fourier or Lagrange) from maximal monochromatic subset of 2-colored integer sequence is a Polynomial point and corresponding maximum monochromatic sequence subset is Polynomial Sequence.

Let f1,f2,f3,... be functions and g1,g2,g3,... be their respective complements. They are represented as graphs constructed previously as F1,F2,F3,....(each with a monochromatic red clique and independent set of blue color) and G1,G2,G3,... (each with a monochromatic blue clique and independent set of red color). All fi(s) have isomorphic red clique subgraphs and gi(s) have isomorphic blue subgraphs. Vertices are numbered with fiinverse() and giinverse() labels. Vertices of cliques in fi(s) and gi(s) have monochromatic arithmetic progressions in fi(vertexlabel) and gi(vertexlabel).For each arithmetic progression APj in monochromatic clique in Fk corresponding to fk(),fkinverse(APj) is the preimage of APj. Define AP1,AP2,AP3,AP4,AP5,... as the monochromatic arithmetic progressions in a monochromatic clique (vertices corresponding to some function fk()). Union of APj is a subset of fk(). Define C(x) as the interpolation polynomial of the Union of APj. C(x) is an approximation of fk(). Accuracy of approximation is determined by the density ratio |C(x)|/|fk()|.When Union of APj covers all points in sequence for generating function fk(), then density is 1 because C(x) is 100% correct approximation of fk() - C(x) = fk(). A naive construction of C(x) from APj is

done by parallel mergesort of: AP1,AP2,AP3,AP4,AP5,...,APn to get a total ordering. Interpolation on the merged splintered arithmetic progressions scattered over the sequence gives an approximation C(x) of fk(). This construction is in NC. Let Interpolation polynomial for the sequence be P(x)=fk().
There are 2 possibilities of arithmetic progressions - APs have both colors (elements from both function and its complement) or APs are all monochromatic (elements from a function or its complement).For example the sequence 1,2,3,4,5,.... of natural numbers have two arithmetic progressions:

AP1 = 2x+1 => 1,3,5,7,9,.....
AP2 = 3x+1 => 1,4,7,11,14,....

Merging them gives 1,3,4,5,7,9,10,11,13.... which is a subset of 1,2,3,4,5,....

Scenario1: Monochromatic APs

Approximation polynomial for sequence interpolated from constituent monochromatic APs mergesorted = C(x). Let AP1(x),AP2(x),AP3(x),...,APn(x) be the arithmetic progression polynomials. Let C(x) = E(x)AP1(x)AP2(x)AP3(x)...APn(x) i.e arithmetic progressions are factors of C(x) with E(x) as quotient over some polynomial ring F[x]. C(x) approximates a function fk() or its complement gk(). This is described previously.

Scenario2: Multichromatic APs

When APs have elements of both colors (from both functions fk() and gk()), parallel mergesort of APs and interpolation by polynomial C(x) cannot approximate either fk() or gk() because APs consist of both colors. This is where minimum discrepancy plays its part - when discrepancy is low AP has elements from both colors or functions in almost equal number. Therefore arithmetic progression samples both functions with least error and interpolation succeeds with high accuracy - this is an intuitive explanation for 338.11.

References:

338.1 Bound for Van Der Waerden Numbers - https://www.emis.de/journals/INTEGERS/papers/a20int2005/a20int2005.pdf

338.2 Exact Ramsey Theory and Van Der Waerden Numbers by SAT-solvers - [OliverKullmann] - https://arxiv.org/pdf/1004.0653v2.pdf

338.3 Erdos-Turan Conjecture - https://en.wikipedia.org/wiki/Erd%C5%91s_conjecture_on_arithmetic_progressions - If sum of reciprocals of elements of a set of positive integers diverges, then the set contains arbitrarily long arithmetic progressions

338.4 Roth Estimate - http://matwbn.icm.edu.pl/ksiazki/aa/aa9/aa9125.pdf - Consider a two coloring of set of natural numbers N in red and blue. For all arithmetic progressions in the set N let Discrepancy be the difference between number of red and blue colored integers. Maximum of Discrepancy for all arithmetic progressions be Maximum(Discrepancy). Minimum(Maximum(Discrepancy)) for all 2-colorings is the Roth Estimate lowerbounded by N^1/4.

338.5 Roth Estimate is nearly sharp - Beck, J. Combinatorica (1981) 1: 319. doi:10.1007/BF02579452 - http://link.springer.com/article/10.1007/BF02579452 - Previous Roth Estimate is upperbounded by N^(1/4+epsilon).

338.6 Discrepancy in Arithmetic Progressions - [MatousekSpencer] - Roth estimate is best possible - ROTH(N) <= N^1/4 - http://www.ams.org/journals/jams/1996-9-01/S0894-0347-96-00175-0/S0894-0347-96-00175-0.pdf - This paper has some interesting remarks - "...In words, we show the existence of a two-coloring χ of the first n integers so that all arithmetic progressions A have imbalance |χ(A)| ≤ Cn1/4. We remark that the proof does not give a construction of χ in the usual sense and is indeed not satisfactory from an algorithmic point of view. The methods of §2 (see comments in [5]) are such that we have not been able to obtain an algorithm that would output this coloring χ in time polynomial in n. Our proof involves variants of the probabilistic method; we give [1] as a general reference ..." which is about algorithm for constructing a coloring for a given discrepancy - coloring algorithm is nothing but construction of a function and its complement for a finite set of integers. Thus notions of

integer sequence coloring and complement function are two sides of a coin with respect to integer valued functions and complements. Undecidability of Infinite complementation from 19,24,319 and 323 PCP based proof also implies "Infinite Integer Sequence Coloring is Undecidable" i.e there is no coloring algorithm for 2-coloring of infinite integer sequences.
338.7 Discrepancy Minimization by Walking on Edges - [BansalLovettMeka] - https://arxiv.org/pdf/1203.5747v2.pdf - This proves Spencer's Theorem which states that for any system of subsets S of universal set V of size both N there always exists a 2-coloring with minimax(discrepancy) < 6*sqrt(N) also known as Six Standard Deviation and gives a randomized polytime algorithm for constructing such a 2-coloring mentioned non-constructively in 338.6. System of subsets can be set of arithmetic progressions. This is a special case of function-complement (2-coloring) construction subject to minimax discrepancy criterion.
338.8 Fourier Interpolation of n data points with polynomial of degree m - http://www.muskingum.edu/~rdaquila/m350/fourier-interdn10-1.ppt
338.9 Fourier Analysis and Szemeredi's Theorem - Roth's Argument - Documenta Mathematica ICM Extra Volume 1998 - http://www.mathunion.org/ICM/ICM1998.1/Main/Fields/Gowers.MAN.ocr.pdf - Fourier Transform of set of integers mod N (for prime N) and deriving the arithmetic progressions of length 3 from inverse Fourier Transform.
338.10 Hales-Jewett Theorem - [MichelleLee] - http://www.math.uchicago.edu/~may/VIGRE/VIGRE2009/REUPapers/Lee.pdf - generalization of Van Der Waerden Theorem to higher dimensions (n-cubes and n-lines) - any k-coloring of n-dimensional cube (or a matrix of points) has monochromatic line(with fixed coordinates for some dimensions and varying for others)
338.11 Polynomial Interpolation from Low Discrepancy Arithmetic Progressions - [The Discrepancy Method: Randomness and Complexity - BernardChazelle] - https://books.google.co.in/books?id=dmOPmEh6LdYC&pg=PA338&lpg=PA338&dq=polynomial+interpolation+arithmetic+progression&source=bl&ots=CDNgOUBnua&sig=pxKMQx9ciY7_d4GcOkFwnCrOUEU&hl=en&sa=X&ved=0ahUKEwiSqOv0ovPPAhVMpo8KHZsbA3wQ6AEIPzAE#v=onepage&q=polynomial%20interpolation%20arithmetic%20progression&f=false - Page 338 and Figure 9.4 in Page 342 illustrating the distribution of Fourier Coefficients over a circle in Discrete Fourier Transform of a subset of Z16 (also explained in 338.8).
338.12 Sequences Generated By Polynomials - [CorneliusSchultz] - https://www.researchgate.net/profile/E_Cornelius_Jr/publication/258155515_Sequences_Generated_by_Polynomials/links/0deec52725419439380000000.pdf
338.13 Polynomial Points, Green-Tao Theorem and Arithmetic Progressions - [CorneliusSchultz] - https://cs.uwaterloo.ca/journals/JIS/VOL10/Schultz/schultz14.pdf
338.14 Recent Advanced Theorems in k-coloring of integer sequences - http://people.math.sc.edu/lu/talks/ap4.pdf - the set [n] is randomly r-colored and bounds for number of monochromatic k-APs and K-APs in [n] are mentioned where k is common difference and K is Van Der Waerden number.

---
339. (FEATURE-DONE) Boyer-Moore Streaming Majority Algorithm - Commits - 7 October 2016

This commit implements Boyer-Moore algorithm for finding Majority element in Streaming Sequences. It uses the Streaming Generator
Abstraction for input streaming datasource. Logs have been committed to python-src/testlogs/

---
340. (FEATURE-DONE) GSpan Graph Substructure Mining Algorithm Implementation - Commits - 13 October 2016 and 14 October 2016

1.Graph Substructure Mining GSpan algorithm implementation with logs in

testlogs/
2.This code requires the Graph vertices to be labelled by unique integers
3.Integer labelling of vertices makes it easier for DFSCode hashes to be generated uniquely.

---

341. (THEORY) Graph Mining Algorithms and Recursive Gloss Overlap Graph for text documents - Document Similarity

---

Recursive Gloss Overlap Algorithm implementations in python-src/InterviewAlgorithm and its Spark Cloud Variants generate graphs with word labels from Text Documents. GSpan algorithm implemented in (340) mines subgraphs and edges common across graph dataset.
This allows extraction of common patterns amongst text document graphs and thus is an unsupervised similarity clustering for
text analytics. GSpan has provisions for assigning minimum support for filtering patterns which amounts to finding prominent
keywords in recursive gloss overlap graphs.

---

342. (FEATURE-DONE) Graph Mining Recursive Gloss Overlap Graph for text documents - Document Similarity

---

Commits - 17 October 2016

1.Code changes have been done to choose between numeric and word labelling of Graph vertices in GSpan GraphMining implementation.
2.New file GraphMining_RecursiveGlossOverlap.py has been added to JSON dump the Recursive Gloss Overlap graph of a text document
3.New directory InterviewAlgorithm/graphmining has been created which contains the numeric and word labelled Recusive Gloss Overlap
graphs of 5 example text documents in topic class "Chennai Metropolitan Area Expansion"
4.logs for commong edges mined between two Recursive Gloss Overlap document graphs has been committed in testlogs/
5.Spidered web text has been updated
6.GraphMining_RecursiveGlossOverlap.py JSON dumps the RGO edges into a text file and GraphMining_GSpan.py JSON loads them from
InterviewAlgorithm/graphmining/

---

343. (THEORY) Streaming Majority and 2-Coloring(Complement Functions) - related to 14.16 and 19,24,323,338,339 - important draft
updates to http://arxiv.org/pdf/1106.4102.pdf - 1 November 2016

In simple 2 candidate majority voting done on stream of votes, a 2-colored sequence of votes by voters is generated. This is
already mentioned in infinite majority (Erdos Discrepancy Theorem) - 14.16.Each color symbolizes a candidate voted. This partitions
the sequence of votes into 2 monochromatic sets. Obviously, if the voters are uniquely identified by a sequence number, all sequence coloring
theorems imply there are monochromatic and multichromatic arithmetic progressions in voter unique identities. Boyer-Moore algorithm
computes streaming majority (implemented in 339),

---

344. (BUG-STABILITY ISSUES) AsFer-VIRGO Boost::Python system calls invocations analysis - commits - 4 November 2016

Some resumed analysis of AsFer-VIRGO boost::python invocations of VIRGO memory system calls which were stopped few months ago.
The i915 DRM GEM error is highly reproducible pointing something unruly about it. No logical reason can be attributed to this except
some kernel sync issues. This further confirms that there is nothing wrong with VIRGO layer of Linux kernel and Linux kernel deep
within inherently has a chronic problem.

---
345. (THEORY) Polynomial Reconstruction (PR) Problem, Error Correcting Codes and 2-Coloring/Complement Functions - important
draft updates to http://arxiv.org/pdf/1106.4102.pdf - 10 November 2016 and 17 November 2016

Polynomial Reconstruction Problem states that:
Given a set of n points (x1,y1),(x2,y2),...,(xn,yn) recover all polynomials p of degree less than k such that pi(xi) != yi
for maximum of i points in {1,2,3,...,n}. In other words PRP is a curve-fitting interpolation problem and is related to
Error correcting problems like List Decoding (list of polynomials approximating a message one of which is correct),Reed-Solomon Codes.

2-Coloring scheme algorithms and Complement Function constructions are alternative spectacles to view the Polynomial Reconstruction
Problem i.e Complement Function and 2-Coloring are special settings of Polynomial Reconstruction with exact curve fitting.

References:

345.1 Polynomial Reconstruction and Cryptanalysis - Berlekamp-Welch, Guruswami-Sudan algorithms - https://eprint.iacr.org/2004/217.pdf
345.2 Hardness of Constructing Multivariate Polynomials over Finite Fields - [ParikshitGopalan,SubhashKhot,RishiSaket] - https://www.cs.nyu.edu/~khot/papers/polynomial.pdf - If there is a polynomial P(X1,X2,...,Xn) such that P(xi) = f(xi) for all points (xi,f(xi)), it can be found by interpolation. This is exact agreement. Can we find a polynomial that agrees on most points? This approximation is proved to be NP-hard.
345.3 Lagrange Interpolation and Berlekamp-Welch PRP algorithm - [SadhkanRuma] - https://www.academia.edu/2756695/Evaluation_of_Polynomial_Reconstruction_Problem_using_Lagrange_Interpolation_Method?auto=download

---
346. (THEORY) Data written in electronic storage devices (e.g CD/DVD) and 2-Coloring/Complement Functions - important
draft updates to http://arxiv.org/pdf/1106.4102.pdf - 24 November 2016 and 28 November 2016

CD/DVD storage devices contain binary data burnt on concentric cylindrical tracks. A radial line from central bull's eye of the circular storage to outer rim crosscuts the data to form a binary string of length n where n is the radius of circular storage. This radial line sweeping the
complete circle in a scan covers 2^n possible binary strings. In the best case all 2^n words are distinct. Total Number of 1s or 0s in these
2^n words are in fibonacci sequence:
      f(n) = 2f(n-1) + 2^(n-1)
with f(0)=0 and f(1)=1. In 2-coloring parlance, DVD is 2-colored with 1(red) and

0(blue) where number of monochromatic bits is lower
bounded by:
f(n) = 2f(n-1) + 2^(n-1) out of minimum n*2^n possible bits on the storage. Thus any data written to storage ultimately gets translated into a 2-coloring complement function scheme.For example, device with 3 concentric tracks has 3*2^3=24 minimum possible bit positions and:
f(3) = 2f(2) + 2^2 = 2(2f(1) + 2) + 2^2 = 8 + 4 = 12 minimum possible monochromatic bit positions
This has some applications of Hales-Jewitt Theorem for multidimensional 2-coloring - storage device always has a monochromatic k-line.(where k is dimension and k-line is a hyperline. 2-line is a square and 3-line is a cube). As a simple example for n=2:
00
01
10
11
are minimum possible distinct binary words swept by a radial scan of the circle and there are f(2)=2f(1) + 2 = 4 monochromatic bits spread across out of 2*2^2=8 possible bits in 4 binary strings. A crucial insight is that any high level data stored has an order in low level 2-coloring or Complement Function scheme (Monochromatic APs etc.,) though high level data is usually alphanumeric and appears random. A conjectural question is: Does the low level 2-coloring binary order imply high level non-binary order and are they functionally related?

The previous question has been answered in 2.10. There could be \/ shaped gaps in circular arrangement of 2^n binary word radial lines. These gaps could be resolved by recursive application of f(n) for all lengths < n. Let this new function be g(n) defined as:
g(n) = f(n) + f(n-1) + f(n-2) + ... + f(1) + f(0)
Thus g(n) is the tightest bound for number of 1s and 0s. Prefix "minimum" has been added in this bound because repetitions of some binary words in radial line during circular scan have been ignored and hence a lowerbound. Total number of bit positions is k*2*pi*(1+2+3+...+n) = k*2*pi*n(n+1)/2 summing up all concentric tracks for some constant k. Lowerbound g(n) for 1s and 0s implies a unique distribution of all binary word patterns and any number above this lowerbound implies predominance and emergence of a binary pattern.

---
347. (FEATURE-DONE) NeuronRain C++-Python System calls invocation - Commits - 29 November 2016
---
1. Further analysis of NeuronRain AsFer-VIRGO boost::python system calls invocation - there has been
an erratic -32 and -107 errors
2. But VIRGO system calls work without problems - malloc,set and get work as expected
3. There is a later panic outside VIRGO code but logs have not been found.
4. Logs for this have been added in testlogs/
5. boost C++ code has been rebuilt.

---
348. (THEORY) Kleinberg Lattice , Random Graph Ontologies, Bose-Einstein model and Recursive Gloss Overlap algorithm for ranking documents by intrinsic merit - 3 December 2016 and 20 December 2016 - related to 202, 229 and 230

Result of r=2 in [Kleinberg - 202.19] is an equilibrium state. Random shortcut edges are created with some probabilities between any two nodes n1 and n2 on a lattice of all possible nodes. Thus there are 2 distance measures between any two nodes on a lattice - 1) lattice distance which is the usual manhattan distance step function - this is circuitous. 2) random edge distance between

pair of vertices n1 and n2 on the lattice which is shortcut. The random edge has the probability as a function of lattice distance:

random edge probability(d(n1,n2)) = |l(n1,n2)|^(-r) where l is the lattice distance

When r is less than 2 or close to zero, random edges are as numerous as lattice edges (random edges exist in abundance but they are no better than lattice paths) and when r is more than 2, random edges are rarer (random edges fade, only lattice paths are possible and no shortcut random paths) and it is difficult to find a path for message to be delivered from one extreme to the other. Steady state converges and these two conflicts are resolved when r=2 and it is optimum to find a path between two nodes.

Above small world phenomenon can be mapped to ontology of linguistic concepts/words. Let set of all concepts/words form a lattice. Set of random edges with probability embeds a random graph on some or all of these lattice points and thus is a probabilistic ontology - probability depends on distance measure of 2 concepts. Document definition graph obtained from this ontology with recursive gloss overlap is optimal (easy to find path between concepts in a document and grasp meaning) when it has r=2 by previous lattice-randomedge relation. Usual distance measures are based on least common ancestor principle - node which is conceptually common to two other nodes creates a path. Lower the distance, greater the meaningfulness. Reference 202.20 adds disambiguation to finding distance by aligning and intersecting all possible senses of two concepts with a random walk and doing argmax() to find distance. Probabilistic random edge ontology created on a lattice of concepts provides dynamism in text analytics.

Kleinberg criterion of r=2 is an alternative way to assess the meaningfulness of a document. Document definition graph with r=2 should theoretically have easy paths between concepts and average short distance measures across nodes and hence has high intrinsic merit. From the above relation:

r * log(l(n1,n2)) = log(1/d(n1,n2))

When r=2:

2log(l(n1,n2)) = log(1/d(n1,n2))

which stipulates conditions for high intrinsic merit for a document in terms of lattice distance and random edge probability between two concepts n1 and n2.

Bose-Einstein model for networks relates fitness of a vertex (ability to attract edges) in a graph and Bose-Einstein condensation. Kleinberg's small-world graph has average clustering coefficient significantly higher than a random graph and mean shortest distance approximately same as random graph (on same vertices). Here clustering coefficient is the ratio of number of edges to neighbours and number of all possible edges to neighbours. Kleinberg's small-world graph, Bose-Einstein condensation in network graphs and Intrinsic meaningfulness merit in graph representation of documents are closely related because all three are multiple views of "meaningfulness" or "connectedness" in a document.

References:

348.1 Bose-Einstein Model and Complex Networks - https://en.wikipedia.org/wiki/Bose%E2%80%93Einstein_condensation_(network_theory)

348.2 Golden mean as a clock cycle of brain waves - http://www.v-weiss.de/chaos.html - "...In 2001 Bianconi and Barabási [15] discovered that not only neural networks but all evolving networks, including the World Wide Web and business networks, can be mapped into an equilibrium Bose gas, where nodes correspond to energy levels and links represent particles. Still unaware of the research by Pascual-Leone, for these network researchers this correspondence between network dynamics and a Bose gas was highly unexpected [16]..."

---

349. (THEORY) Randomness, Quantum Machine Learning and a Schroedinger Cat

simulation of learning patterns in BigData - 23 December 2016

Caution: This section postulates a drastically new theory mapping quantum mechanics to learning patterns in bigdata which is subject to errors.

State of a subatomic particle is defined by a wave function on Hilbert Space (Complex State Vector Space). In Dirac notation wave function for particle p is, |p> = a1|s1> + a2|s2> + ... + an|sn> is state of a particle where each ai is a complex number named amplitude of particle at state si - particle's state is a linear superposition of all states si with amplitude ai. Dirac delta function is fourier transform of Schroedinger wave equation of a particle. State vector |p> is a linear superposition of all possible alternatives (or) Hilbert space dimensions that a particle can exist with respective amplitude for each dimension. Set of states form orthonormal basis for this Hilbert space. Wave function state vector collapses to one of the states to give a classical probability for the state estimated by square of amplitude for the state dimension. For example, in Young's Double Slit experiment a single photon acts as a wave by duality and interferes with itself by travelling through both slits to establish an interference pattern with state vector wavefunction amplitudes which collapses to a classical probability when one of the slits is closed. Schroedinger's cat paradox is an imaginary thought experiment (not possible in reality) where a black box with cat and a radioactive material triggered by a particle's spin exists in both states dead and alive simultaneously when viewed from outside with amplitude 1/sqrt(2): |state of cat> = 1/sqrt(2) (|dead> + |alive>). Squaring amplitude 1/sqrt(2) gives classical probability 1/2 for each state. There are two observers - one within the black box and one outside of it. For observer within black box, particle spin is measurable and wavefunction collapses to one of the states "dead or alive" while for observer outside black box there is no way for such collapse to occur and state is a superposition "dead and alive".

In terms of BigData analytics, learning a variable in a blackbox is reducible to Schroedinger's Cat paradox. Assuming there exists a Pseudorandom Generator with Quantum Mechanical source, Cat is an algorithm with access to randomness - e.g Double Slit experiment, radioactive decay etc., and chooses one of the alternative outputs - output1, output2, ..., outputn - based on quantum mechanical measurement (e.g spin) with "if...else" branches. This randomized algorithm can be thought of as a subroutine F. Observer1 which measures output of F is another subroutine G. Thus both F and G together constitute the blackbox. Observer2 outside the blackbox is another subroutine H. Only G measures output of F and state vector collapses to one of the alternative outputs. But for H, F+G is impervious and state vector remains as: a1|output1> + a2|output2> +...+ an|outputn> with complex amplitudes ai. This extends the notion of Pseudorandomness in traditional complexity literature to a more generic state vector with complex amplitudes over a Hilbert space of dimensions(alternatives) - here algorithm is itself a "particle" with a superposed state of outputs.

Any BigData set with apparent randomness can be construed as a series of outputs over time generated by a randomized algorithm (or) an algorithm with access to randomness including quantum mechanical randomness (For example, streamed datasets like stock market tickers) i.e the algorithm F. Subroutines F+G together create a BigData set from randomness. When viewed from H, the blackbox F+G has a state vector wavefunction evolving over time: W(t) = a1|output1> + a2|output2> + ... + an|outputn>. Any machine learning algorithm which tries to learn patterns from such a dataset is equivalent to what H does above. If H learns the pattern in the dataset with x% accuracy, it implies algorithms F+G are reverse engineered with x% correctness. This presents a contradiction if the source of randomness is quantum mechanical - state vector collapses for both observers G and H, not only for G. If Schroedingers cat paradox is axiomatically true, it raises questions: Does this limit the scope of machine learning to only classical randomness? Is quantum randomness not learnable?

An alternative formulation of F+G is:
F is the quantum randomness source (e.g Double slit) and G invokes (or measures) F to produce a datastream over time. Here G as observer within blackbox is an inseparable entity from F i.e F is an internal routine of G.

There are two levels of learning possible:
*) Learning within blackbox - Learning patterns from observations of F+G. F+G are pre-equipped with ability to learn patterns in collapsed wavefunction generated dataset. Such a pattern learnt exists only within the blackbox. External observer H has no way to learn the dataset and the pattern. Even within F+G there are problems with learning patterns from quantum randomness. Logical independence implies any two observations (boolean propositions) are independent and learning pattern from such independent observations contradicts it - pattern in independent observations imply observations are dependent on each other. For example, two propositions "Today is holiday" and "There are 100 cars" are logically independent while propositions "There are 100 flights" and "There are 100 cars" are logically dependent. Former 2 propositions have no common patterns while latter 2 propositions have a common pattern (100, vehicles) - new information "100 and vehicles" is deducible from 2 propositions. Quantum randomness stems from logical independence and thus there should not be a common pattern to learn from independent streaming set of observations.

*) Learning outside blackbox - This is obvious contradiction mentioned previously because dataset generated by F+G is never visible to H because wavefunction of H never collapses.

Thus above seems to imply learning is impossible with quantum randomness.

References:

349.1 Feynman Lectures on Physics - [RichardFeynman] - Volume 3 - Chapter 1 and 2
349.2 Emperor's New Mind - [RogerPenrose] - Chapter 6 - Quantum Magic and Quantum Mystery
349.3 Quantum Randomness and Logical Independence - https://arxiv.org/pdf/0811.4542v2.pdf - Mutually independent logical propositions cause quantum randomness
349.4 Elitzur-Vaidman Bomb tester - https://en.wikipedia.org/wiki/Elitzur%E2%80%93Vaidman_bomb_tester - Detecting if bomb inside a box has detonated or not by quantum superposition - Constructive and Destructive self-interference - Previous thought experiment replaces the bomb in the box by a bigdata source which has to be predicted. Difference is one of the observers (detectors) is within the box and other is outside.

---
350. (THEORY) Quantum Parallelism, Quantum Interference, Integer Factoring, Periodicity Finding, Polynomial Reconstruction and Complement Function/2-Coloring - related to 24,34,323,338,345 and 347 - 29 December 2016 - important draft updates to :
- http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.tex/download and
- http://arxiv.org/pdf/1106.4102.pdf

Discrete Hyperbolic Factorization draft in 34 claims (disputed because of input size though references exist to prove PRAM=NC equivalence and Bitonic Batcher Sort is in NC) that integers can be factored by NC parallel computation circuit of logdepth and polynomial size (by PRAM or Cloud Bitonic sorting). Similar parallelism is the basis for Quantum Factorization Algorithm of [Shor] - factorization is in BQP and NC is in BQP. Quantum Computation relies on two phenomena:

*) Quantum Parallelism in which a function f(x) is computed to get many values of f(x) in parallel
) Quantum Interference by Hadamard transform where cancellation occurs (|0> = 1/sqrt(2)(|0>+|1>) and |1> = 1/sqrt(2)*(|0>-|1>)) by interference

Period Finding in Quantum Computation finds smallest r such that f(x)=f(x+r) and Shor's factorization internally applies period finding by quantum fourier transform. Ramsey theory of integers and related theorems viz., Van Der Waerden, Szemeredi etc., imply existence of monochromatic arithmetic progressions in 2-coloring of integer sequences where each color is equivalent to a function and its complement. Period finding problem reduces to finding arithmetic progressions in integer sequences by defining a function f(x+iy)=f(x+(i+1)y) for any x in the sequence and some integers y and i. More generically, factoring and period finding are special cases of Hidden Subgroup Problem i.e f(g)=f(hg) for g in G and h in subgroup H of G and hG is a coset of H. In terms of polynomial reconstruction, period finding computes the function f passing through the arithmetic progression points. In Discrete Hyperbolic Factorization of 34, instead of quantum parallelism , classical NC parallelism is applied to k-merge sort and binary search the pixelated hyperbola tiles.

References:

350.1 Quantum Computation Course notes - Order Finding, Fermat's Little Theorem and Simon, Shor Factorization algorithms - [AndrisAmbainis] - http://www.cs.ioc.ee/yik/schools/win2003/ambainis2002-2.ppt
350.2 Progress on Quantum Algorithms - [PeterShor] - http://www-math.mit.edu/~shor/papers/Progress.pdf

---
351. (FEATURE-DONE) Recommender Systems Implementation based on ThoughtNet Hypergraph - 1 January 2017
---
*) Input to python-src/DeepLearning_ReinforcementLearningRecommenderSystems.py is a set of observations which are user activities
(shopping cart items, academic articles read by user etc.,) and already built thoughtnet hypergraph history is searched by usual
evocatives from the input
*) Evocatives returned from ThoughtNet hypergraph are items recommended relevant to user's activities.
*) This technique of recommendation is more qualitative than usual methods of Collaborative Filtering and mimicks the human
thought process of "relevance based evocation" based on past experience (i.e ThoughtNet history)
*) New input text file and folder RecommenderSystems has been added to python-src with Neo4j Graph Database support
*) Logs have been committed in testlogs/

---
352. (FEATURE-DONE) Kafka data storage for Streaming Abstract Generator - 2 January 2017

Kafka Streaming Platform has been added as a data storage in Streaming Generator Abstraction Single-Window entrypoint. Kafka subscriber
code for a neuronraindata topic polls for incoming messages in iterator. As an example Streaming CountMeanMinSketch implementation has
been updated to source streamed data published in Kafka.

---
353. (FEATURE-DONE) An example usecase of Recommender Systems for online

shopping cart - 2 January 2017
---
*) RecommenderSystems folder has been updated with new text file from an example past shopping cart history items (e.g books from miscellaneous topics in Amazon online bookstore)
*) From the above edges a Hypergraph is created by classifying the shopping cart items into classes and constructing hyperedges across the classes (by Recursive Gloss Overlap Graph maximum core number classifier)
*) Text files for above usecase have .shoppingcart suffixes
*) Above shopping cart hypergraph has a past history of items chosen by a user from variety of topics.
*) New input file RecommenderSystems.shoppingcart.input.txt has present items in user's shopping cart and new items have to be recommended to user based on present choice and past history.
*) For this, items in present shopping cart are lookedup in past sales history Hypergraph created previously and recommendations are returned
*) Items in present shopping cart are looked up in two ways: 1) By classifying with Recursive Gloss Overlap graph and looking up rare classes less than a threshold (presently core number 5) and 2) Raw token lookup
*) Logs for these two lookups based Recommendations generated are committed to testlogs/

An important note: Rationale for Hypergraph based past history is that a meaningful text can be classified on multiple classes vis-a-vis usual supervised classifiers which classify text on exactly one class. This is more realistic because text can delve into multiple topics simultaneusly e.g an article on medical imaging (MRI scans) could contain details also on nuclear physics and pattern recognition and can exist in 3 classes simultaneously. There are still impurities in recommendations generated from lookups as gleaned from logs. These are inaccuracies in the way WordNet infers because of small size of the text description about book. Larger the description per shopping cart item, greater is the information connectedness of the WordNet subgraph generated and accuracy of core number based classification.

---
354. (FEATURE-DONE) More detailed shopping cart example 2 for RGO+ThoughtNet based Recommender System - 3 January 2017
---
*) python-src/DeepLearning_ReinforcementLearningRecommenderSystems.py has been updated to choose top percentile classes of text input so that vertices with large core numbers get more weightage.
*) New shoppingcart2 product reviews text has been added and corresponding Hypergraph history has been created. This example has detailed product description with an assorted mix (TV reviews, Washing Machine reviews, Home Theatre reviews etc.,) - python-src/RecommenderSystems/RecommenderSystems_Edges.shoppingcart2.txt, python-src/RecommenderSystems/RecommenderSystems_Hypergraph_Generated.shoppingcart2.txt
*) New input file python-src/RecommenderSystems.shoppingcart2.input.txt has been added with an example product (TV).
*) Recommender System chooses relevant TV products from Hypergraph history and displays to user.
*) Logs have been committed to testlogs/

---
355. (THEORY) NEXP not in non-uniform ACC, P(Good) majority voting circuit and some contradictions - 4 January 2017 and 5 January 2017
- related to all P(Good) majority voting circuit related points in this document (e.g 14,53 etc.,)

RHS of P(Good) majority voting circuit has exponential size with unrestricted depth in worst case and is in EXP (if depth restricted , RHS is in PH direct-connect uniform circuit). RHS is a boolean function composition of non-uniform NC and individual voter decision functions. Known result by [RyanWilliams] limits that NEXP is not in ACC (AC circuits with counter mod[m] gates which output 1 if sum of inputs is a multiple of m). Non uniform ACC is contained in TC and AC is contained in ACC (AC in ACC in TC). This makes a contradiction if LHS is 100% efficient percolation circuit in non-uniform NC (NC/poly). If P(Good) binomial coefficient series summation converges to 100%, then RHS majority voting circuit is also 100% efficient and LHS=RHS. This implies if RHS has NEXP-complete algorithm/circuit, because of convergence, NEXP is in NC/poly. But NC is in ACC and this implies NEXP has non-uniform ACC circuits - contradicts NEXP does not have non-uniform ACC circuits. This is one more counterexample implying some or all of the following:

*) There is no 100% efficient RHS majority voting boolean function composition

*) There is no 100% efficient LHS boolean function (pseudorandomly chosen boolean function, ranked by social choice function etc.,)

*) RHS Majority voting is not in NEXP.

*) LHS pseudorandom choice boolean cannot have a non-uniform NC circuit

Above, "efficiency" implies "No error" cases in scenarios matrix of 53.14 and a ninth error scenario of Randomized Decision Tree Evaluation Error(zero-error decision tree evaluation) mentioned in 314, 317. This adds one more possibility where P(Good) binomial summation for majority voting diverges already mentioned in 53.16.3.1, 256, 265, 275. Ranking by social choice function includes Interview algorithm implemented in Asfer and any other ranking function. Thus there exists a setting where both LHS and RHS fail to converge. It has to be noted that above counterexamples do not rule out the possibility of convergence of LHS and RHS of P(Good) circuit. There could be decision functions which adhere to "no error" cases in 10 possibilities as below:

x	f(x) = f(x/e)	f(x) != f(x/e) Noise
x in L, x/e in L	No error	Error
x in L, x/e not in L	Error	No error if f(x)=1,f(x/e)=0 else Error
x not in L, x/e in L	Error	No error if f(x)=0,f(x/e)=1 else Error
x not in L, x/e not in L	No error	Error

x	f(x)

Randomized Decision tree evaluation	No error	Error

If LHS social choice ranking function is Interview algorithm which is a PSPACE=IP algorithm (by straightforward reduction from polynomial round prover-verifier protocol) or a PSPACE function fitting within "No error" scenarios in matrix above, then an unrestricted depth 100% errorfree RHS EXP circuit could imply, EXP=PSPACE. Hence whether such perfect resilient errorfree decision functions in previous "No error" cases can be learnt is an open question.

There are natural processes like Soap Bubble formation known to solve Steiner Tree NP-hard problem efficiently. Soap Bubbles between two glass plates are formed and bubbles are connected by line segments of optimum total length which converge at Steiner vertices. Voting is a also a natural process with human element involved. Does human judgement overwhelm algorithmic judgement in decision correctness is also an open question. If neural networks are algorithmic equivalents of human reasoning, then each voter decision function could be a TC (threshold) circuit and majority voting could be a composition of NC majority function with TC voter circuit inputs. Thus entire RHS of P(Good) is a huge non-uniform TC neural network. Error of majority voting is then equal to error of this majority function + neural network composition i.e majority voting involving humans is BPTC algorithm.

Previous matrix of 10 scenarios basically subdivides the traditional BP* definition which says: For x in L, a BP* turing machine accepts with probability > 2/3 and for x not in L rejects with probability > 2/3. In other words, a BP* algorithm allows false positives and false negatives and thus errs in "judgement" with probability 1/3. All "Error" entries in the previous matrix are :

*) (f(x)=f(x/e)) in which case two correlated strings one in L and other not in L are both accepted (false positive) and rejected by f (false negative)

*) (f(x) != f(x/e) - Noise sensitivity) in which case two correlated strings in L or not in L are erroneously accepted and rejected by f (false positives and false negatives)

*) Previous two are input related while the last scenario with error in decision tree evaluation is internal to the boolean function itself with false positive and negative decision tree evaluation based on pseudorandom advice bits.

This false positive + false negative voter judgement error applies to BPTC NC+neural network circuit composition also described previously. Derandomizing BPTC should give a non-uniform TC circuit for RHS of P(Good) majority voting implying zero-error voting by neural network voters. This non-uniform TC circuit is an alternative way to specify the enormity of RHS earlier described by PH=DC (depth restricted) and EXP (depth unrestricted) classes. Thus these two unbounded circuit models for RHS of P(Good) majority voting are equivalent: non-uniform TC and EXP. But ACC is contained in TC and NEXP is not in non-uniform ACC. This does not rule out the possibility that EXP has non-uniform TC circuits which is a superset of ACC.

References:

355.1 NEXP not in non-uniform ACC - [RyanWilliams] - http://www.cs.cmu.edu/~ryanw/acc-lbs.pdf

355.2 Soap Bubble Steiner Tree and P=NP - [ScottAaronson] - https://arxiv.org/pdf/quant-ph/0502072v2.pdf

355.3 Constant depth Threshold circuits - http://people.cs.uchicago.edu/~razborov/files/helsinki.pdf

355.4 Circuit Complexity of Neural networks - https://papers.nips.cc/paper/354-on-the-circuit-complexity-of-neural-networks.pdf

355.5 Exact Threshold circuits - http://www.cs.au.dk/~arnsfelt/Papers/exactcircuits.pdf

355.6 Universal Approximation Theorem - [Cybenko] - https://en.wikipedia.org/wiki/Universal_approximation_theorem - Multilayered Perceptrons with single hidden layer and finite inputs can approximate continuous functions [It has to be noted that single layer perceptron cannot compute XOR function and spatial connectedness - from Perceptrons:an introduction to computational geometry by MarvinMinsky-SeymourPapert]. Thus BPTC threshold networks with multiple layers as voter decision functions are good approximators of voting decisions.
355.7 L* algorithm for exact learning of DFAs - [Dana Angluin] - https://people.eecs.berkeley.edu/~dawnsong/teaching/s10/papers/angluin87.pdf - membership and counterexample queries to learn DFA
355.8 Efficient Learning Algorithms Yield Circuit Lower Bounds - [Lance Fortnow] - http://lance.fortnow.com/papers/files/fksub.pdf - Theorem 1 and Corollary 1 for depth-2 threshold circuits (neural networks) - " ... If there exists an algorithm for exactly learning class C (e.g depth-two neural networks) in time 2^s^o(1) with membership and equivalence queries then EXP^NP is not in P/Poly(C) (e.g TC0[2]) ... ". This theorem has direct implications for learning voter decision functions belonging to TC and BPTC classes in majority voting. Intuitively, this implies if an exact voter neural network without error is learnt in time exponential in size of neural network (e.g error minimization by gradient descent, backpropagation etc.,), then EXP^NP is not computable by neural networks - EXP^NP is not in TC/poly.

---
356. (FEATURE-DONE) PythonSpark+Cython Interview Algorithm cloud implementation update - 4 January 2017 - related to 348

PythonSpark+Cython Cloud Implementation of Interview Algorithm has been updated to print average clustering coefficient of the definition
graph for text. Average clustering coefficient is average(per node clustering coefficient) for all nodes where clustering coefficient of
a vertex is a ratio of edges to neighbours to all possible edges to neighbours (or) amount of "cliqueness" of each neighbourhood of all
vertices in a graph. This adds an alternative way for computing intrinsic merit of a document recursive gloss overlap graph mentioned in
348 (Kleinberg's small world graph and clustering coeffient = 2). Average clustering coefficient supplements the intrinsic merit quantitative
score in deciding semantic relatedness of a graph. Closer the clustering coefficient is to 2, it is easier to find paths in graph of a text
and greater the linguistic meaningfulness.

---
357. (THEORY) P(Good) non-majority versus majority social choices, BPTC and PSPACE classes - related to 317,355 - 6 January 2017

Caution: Following tries to prove a major lowerbound result with some assumptions and is subject to errors.

Main motivation for so much emphasis on P(Good) binomial summation convergence is: LHS is non-majority social choice and RHS is majority
social choice and convergence to 100% on both sides implies LHS and RHS are of equal merit with varying complexity classes giving a lowerbound. In non-majority social choice one of the elements in the population set has to be chosen without voting. If there are n voters and out of them m are of x% goodness, probability of pseudorandom choice to have x% efficiency = m/n. If LHS is an Interview algorithm based social choice,
choice process is as follows:

```
foreach(voter)
{
```

Interview the voter (a PSPACE-complete problem where polynomial number of question-answering reduces to prover-verifier protocol)
}
Rank the voters by merit and choose the topmost as non-majority social choice - voting is obviated.

It is interesting to note that web search engines use both non-majority (Ranking by merit) and majority choice (e.g PageRank,Hub-Authority) to rank websites. Above algorithm is polynomial time in number of voters. Previous loop is parallelizable per voter and could be an NC circuit with PSPACE algorithms as inputs. Thus effectively LHS is a PSPACE algorithm. RHS majority voting is better approximated by BPTC circuit with threshold neural network circuits for each voter, mentioned previously. It is known that BPP is in PSPACE. Thus LHS interview social choice with error is already accounted for in PSPACE. This equates an error-prone LHS PSPACE-complete problem to an error-prone RHS BPTC problem. BPTC is in BPP (couldn't find reference for this, but looks obvious because every TC circuit with error can be simulated by a polynomial time Turing Machine with error). This implies BPTC is in PSPACE because BPP is in PSPACE. If equal error on both LHS and RHS implies a lowerbound, LHS PSPACE-complete interview non-majority social choice has a BPTC majority social choice circuit and therefore all PSPACE problems are in BPTC. These two directions together imply PSPACE=BPTC. But BPTC is in BPP and BPP is in PSPACE implying PSPACE=BPP. From https://www.cse.buffalo.edu/~regan/papers/ComplexityPoster.jpg, this might imply a collapse of entire Polynomial Hierarchy upto second level and some consequences for NP and coNP.

---
358. (THEORY) KRW Conjecture on Boolean Function Composition, Non-majority (PSPACE) and Majority (harder than PSPACE) cicuits - related to 14,53,311,357 - 7 January 2017

Previous point mooted the idea of the parallel interview of voters and ranking them as a non-majority social choice. This requires sorting the interview algorithm scores of individual voters and finding topmost. Sorting of numbers is in AC=NC=TC (Sorting networks, parallel sorting, [ChandraStockMeyerVishkin] Constant Depth Reducibility for Sorting - http://www.cstheory.com/stockmeyer@sbcglobal.net/csv.pdf etc., Threshold circuits for sorting) and each voter's PSPACE-complete interview circuit is input to NC sorting circuit to find non-majority social choice. Thus LHS is a circuit composition of NC and PSPACE voter interview circuits denoted as NC+PSPACE. RHS of P(Good) is, as already mentioned by neural network approximation, a bounded probabilistic non-uniform BPTC circuit which is a circuit composition of BPNC and voter BPTC circuits denoted as BPNC+BPTC. If circuit compositions NC+PSPACE = PSPACE and BPNC+BPTC = BPTC then LHS is a PSPACE algorithm to RHS BPTC algorithm assuming equal error.

This creates a following intriguing possibility: If RHS majority voting has voter functions harder than PSPACE (i.e EXP, EXPSPACE, NEXP, coNEXP etc., and not in BPTC) , under equal error assumption, LHS PSPACE-complete non-majority interview algorithm lowerbounds the RHS. PSPACE is in EXP and EXP is in PSPACE. Thus equal error assumption for lowerbound raises possibilities of PSPACE=EXP, PSPACE=NEXP, PSPACE=coNEXP, PSPACE=EXPSPACE etc., It has to be observed that both PSPACE and other harder-than-PSPACE classes contain PP(Probabilistic Polynomial) and thus derandomization (error removal) is implicit. KRW Conjecture for Boolean Function Composition of two boolean functions f and g says: DepthComplexity(f + g) ~ DepthComplexity(f) + DepthComplexity(g). PSPACE-complete interview is equivalent to a TrueQBF phi(q1,a1,q2,a2,...,qn,an) computed by an Alternating Turing Machine (AP=PSPACE) where "forall" quantifier is equivalent to a question qi and "exists" quantifier is equivalent to its answer ai. It is not known if there is a Circuit Composition equivalent of KRW conjecture for depth-size lowerbound of composition of two circuits and it is assumed that NC+(PSPACE=AP) = PSPACE because PSPACE is harder than NC and

composition of NC with PSPACE should increase circuit depth proportional to number of quantifiers in interview TQBF. RHS BPNC+BPTC=BPTC is somewhat obvious because NC=TC=AC and only circuit depth-size increases when composed.

References:

358.1 Karchmer-Raz-Wigderson (KRW) Conjecture of Boolean Function Composition and Information Complexity - http://cs.haifa.ac.il/~ormeir/papers/krw_info.pdf

---
359. (THEORY) Non-majority social choices - Interview Ranking Function and Pseudorandom Choice Function - 11 January 2017 - related to 317, 358 and all other non-majority versus majority voting points
---
Previous sections mentioned about two possible ways of non-majority social choice:

*) Parallel interview of voters and ranking based on sorted interview scores

*) Psuedorandom choice

Interview circuit:

Error in interview circuit which is an NC sorting network with PSPACE-complete voter interviews as inputs is defined in usual sense as:
False positives + False negatives + False questions = Percentage of wrong answers marked as right + Percentage of right answers marked as wrong + Percentage of wrong questions

For sorting purposes output of TQBF is not a binary 0 or 1 but a binary string with value equal to number of correct answers. Like all other boolean functions sensitivity of TQBF phi(q1,a1,q2,a2,q3,a3,...,qn,an) is defined as how flipping of qi and ai affects the output.
Formally,

Sensitivity(TQBF) = Pr(TQBF(X) != TQBF(Xcorrelated)) where Xcorrelated is an interview with erroneous questions and answers.

Thus sensitivity captures the error in interview. It has to be noted that wrong questions also measure the flaw in interview process while wrong answer to a right question is already accounted for by a binary 0. When sensitivity of TQBF is 0, interview is flawless and LHS if P(good) is 1. BKS Majority is least stable conjecture by [Benjamini-Kalai-Schramm] predicts existence of a linear threshold function with stability greater than Majority function of n variables, n odd. Interview Circuit which is a composition of an NC sorting network with PSPACE TQBF interviews of voters, is also a linear threshold function which outputs candidate above a threshold (i.e sum of correct answers > threshold). If Stability(Interview) > Stability(Majority) then it is a proof of BKS conjecture. For large n, Stability(Majority) = 1-2/pi*(delta). When delta is close to 1, Stability(Majority) tends to 0.35.

Pseudorandom Choice:

Set of all voter decision functions is partitioned into n sets where each set has goodness xi. Here goodness of a voter decision function f is defined as: 1-error(f).

Let number of voter decision functions with goodness xi = m(xi). Thus N = m(x1)+m(x2)+m(x3)+...+m(xn)
Expected goodness of a PRG choice is:

1/N * summation(xi*m(xi)) = (x1*m(x1) + x2*m(x2) + x3*m(x3) + ... + xn*m(xn)) / N

When all voter functions have goodness 1 then PRG choice in LHS of P(good) has goodness 1.

These two non-majority choices are just hypothetical examples of how a social choice can be made without voting. In reality how non-majority social choice occurs is quite complex and determined by various factors of society (economic disparities, ethnicity etc.,)

---
360. (FEATURE-DONE) Commits - 11 January 2017 - DeepLearning Convolution Networks update
---
Some changes done to Convolution computation:
*) Made sigmoid perceptron optional in final neural network from Maxpooling layer so that weighted sum is printed instead of sigmoid value
*) This causes the convolution network to be very sensitive to presence of pattern
*) 2 more example bitmaps have been added with varying degree of pattern prominence (a pronounced X and a thin 1)
*) With this final neurons from maxpooling layer print values of neurons which quite closely reflect the pattern's magnitude: Huge pattern results in big value while small pattern results in small value proportionately.
*) This can rank the bitmaps in increasing degree of magnitude of pattern presence
*) Logs for this have been committed to testlogs

---
361. (FEATURE-DONE) DeepLearning Convolution Network - an example pattern recognition from bitmap images - 12 January 2017
---
*) Few more bitmap images have been included with same pattern but of different sizes
*) There are 5 patterns and 9 bitmaps:
        - Thin X and Boldfaced X
        - Thin 0 and Boldfaced 0
        - Thin 8 and Boldfaced 8
        - No pattern
        - Thin 1 and Boldfaced 1
*) Expectation is that Convolution Network final neuron layer must output similar values for same pattern and different values for different patterns
*) pooling_neuron_weight has been reintroduced in final neuron layer of 10 neurons each with a randomly chosen weight.
*) Logs show for all 10 neurons, final neural activation values are close enough for similar patterns and different for different patterns.
*) Thus Convolution Network which is a recent advance in DeepLearning works quite well to recognize similar image patterns.

In above example , 9th and 10th neurons output following values. Example 11 and 12 are similar patterns - 0 and 0. Example 21 and 22 are similar patterns - 8 and 8. Example 41 and 42 are similar patterns - X and X. Example 51 and 52 are similar patterns - 1 and 1. Neurons reflect this similarities :
##########
Inference from Max Pooling Layer - Neuron 8
##########
Example 11:
##########
[27.402311485751664]

```
###########
Example 12:
###########
[27.36740767846171]
###########
Example 21:
###########
[32.15200939997122]
###########
Example 22:
###########
[30.835940458495205]
###########
Example 3:
###########
[17.133246549473675]
###########
Example 41:
###########
[29.367605168715038]
###########
Example 42:
###########
[27.446193773968886]
###########
Example 51:
###########
[22.12145242997834]
###########
Example 52:
###########
[24.17603022706531]
################################################################################################
###########
Inference from Max Pooling Layer - Neuron  9
################################################################################################
###########
Example 11:
###########
[24.667080337176497]
###########
Example 12:
###########
[24.635666910615544]
###########
Example 21:
###########
[28.941808459974094]
###########
Example 22:
###########
[27.757346412645685]
###########
Example 3:
###########
[15.424921894526316]
###########
Example 41:
###########
[26.435844651843524]
###########
Example 42:
###########
[24.70657439657199]
```

```
###########
Example 51:
###########
[19.914307186980512]
###########
Example 52:
###########
[21.763427204358788]
```

---

362. (FEATURE-DONE) DeepLearning BackPropagation Implementation Update - 17 January 2017

---

*) DeepLearning BackPropagation code has been changed to have 3 inputs, 3 hidden and 3 output layers
with 3*3=9 input-hidden weights and 3*3=9 hidden-output weights with total of 18 weights
*) Software Analytics example has been updated with a third input and logs for it have been committed
to testlogs/.
*) Logs include a diff notes of how to extend this for arbitrary inputs and accuracy of backpropagation
which beautifully converges at ~10^-26 error after ~1000000 iterations.

---

363. (FEATURE-DONE) Software Analytics with DeepLearning - 18 January 2017

---

*) An example software analytics code based on DeepLearning implementation has been added to
software_analytics which import BackPropagation,Convolution and RecurrentLSTM to learn models
from software analytics input variables (CPU%, Memory% and TimeDuration%)
*) Logs for all 3 models learnt with same input variables have been added to software_analytics/testlogs

---

364. (THEORY and IMPLEMENTATION) Polynomial Encoding of a Text and 2 new distance measures based on it - related to 2.9
- 19 January 2017

---

*) It has been earlier mentioned that an alphanumeric text can be construed as a polynomial defined as f:position->alphabet
*) This python implementation encodes a text as a polynomial by applying NumPy Polyfit function on ordinal values of the text
letter positions as y-axis and letter positions as x-axis.
*) A sample polynomial plotted with matplotlib has been added to testlogs/ alongwith logs
*) Once a text is plotted as a polynomial curve, it is natural to define distance between two strings as distance
between two respective polynomial encodings.
*) Usual distance measures for strings e.g Edit distance are somewhat qualitative and do not quantify in numeric terms exactly.
*) Distance between polynomials of two texts is numerically quite sensitive to small changes in texts and visually match the
definition of "distance" between two polynomials of strings.

There are two distance functions defined for polynomial representation of texts:
*) Distance between polynomials is defined by inner product of polynomials f(x)

```
and g(x) and in discrete version is the
L2 norm : sum(i(f(xi)-g(xi))^2)
*) Distance between polynomials considering them as two discrete set of
probability distributions - Kullback-Leibler and Jensen-Shannon
 Divergence measures.

--------------------------------------------------------------------------------
-----------------------------------------------
For example following are two polynomials fitted for ordinal values of 2 texts
and comparative distance values are printed for
edit distance and 2 polynomial distance measures:
text9="fdjfjkkkkkkkkkkkkfjjjjjjjjjjjskwwwwwwwwwwwwwwwwwwiiiiiiiiiiiiiiiiiiiiii
iiiiiiiiiiiiiiiiiiii"
text10="wjejwejwkjekwjkejkwjekjwkjekwjjoisjdiwidoiweiwie0iw0eio0wie0wie0iw0ei0cn
dknfndnfndnfkdjfkjdfkjd
--------------------------------------------------------------------------------
-----------------------------------------------
X-axis: xrange(94)
Y-axis: [102, 100, 106, 102, 106, 107, 107, 107, 107, 107, 107, 107, 107, 107,
107, 107, 107, 107, 102, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106, 106,
115, 107, 119, 119, 119, 119, 119, 119, 119, 119, 119, 119, 119, 119, 119, 119,
119, 119, 119, 119, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105,
105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105,
105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105]
Coefficients of fitting polynomial of degree 5: [  1.07744279e+02  -
1.44951431e+00   1.33928005e-01  -3.69850703e-03
   4.03119671e-05  -1.53297932e-07]
X-axis: xrange(95)
Y-axis: [119, 106, 101, 106, 119, 101, 106, 119, 107, 106, 101, 107, 119, 106,
107, 101, 106, 107, 119, 106, 101, 107, 106, 119, 107, 106, 101, 107, 119, 106,
106, 111, 105, 115, 106, 100, 105, 119, 105, 100, 111, 105, 119, 101, 105, 119,
105, 101, 48, 105, 119, 48, 101, 105, 111, 48, 119, 105, 101, 48, 119, 105, 101,
48, 105, 119, 48, 101, 105, 48, 99, 110, 100, 107, 110, 102, 110, 100, 110, 102,
110, 100, 110, 102, 107, 100, 106, 102, 107, 106, 100, 102, 107, 106, 100]
Coefficients of fitting polynomial of degree 5: [  1.13816036e+02  -
2.17630157e+00   1.96357195e-01  -6.45349037e-03
   8.34267469e-05  -3.67775407e-07]
...
################################################################################
##
Two text polynomial ordinal points are represented as set of ordered pairs
################################################################################
##
Wagner-Fischer Edit Distance for text9 and text10: 76
Polynomial Encoding Edit Distance (JensenShannon) for text9 and text10:
str1str2_tuple: [(102, 119), (100, 106), (106, 101), (102, 106), (106, 119),
(107, 101), (107, 106), (107, 119), (107, 107), (107, 106), (107, 101), (107,
107), (107, 119), (107, 106), (107, 107), (107, 101), (107, 106), (107, 107),
(102, 119), (106, 106), (106, 101), (106, 107), (106, 106), (106, 119), (106,
107), (106, 106), (106, 101), (106, 107), (106, 119), (106, 106), (115, 106),
(107, 111), (119, 105), (119, 115), (119, 106), (119, 100), (119, 105), (119,
119), (119, 105), (119, 100), (119, 111), (119, 105), (119, 119), (119, 101),
(119, 105), (119, 119), (119, 105), (119, 101), (119, 48), (119, 105), (105,
119), (105, 48), (105, 101), (105, 105), (105, 111), (105, 48), (105, 119),
(105, 105), (105, 101), (105, 48), (105, 119), (105, 105), (105, 101), (105,
48), (105, 105), (105, 119), (105, 48), (105, 101), (105, 105), (105, 48), (105,
99), (105, 110), (105, 100), (105, 107), (105, 110), (105, 102), (105, 110),
(105, 100), (105, 110), (105, 102), (105, 110), (105, 100), (105, 110), (105,
102), (105, 107), (105, 100), (105, 106), (105, 102), (105, 107), (105, 106),
(105, 100), (105, 102), (105, 107), (105, 106), (1, 100)]
421.059610023
Polynomial Encoding Edit Distance (L2 Norm) for text9 and text10:
str1str2_tuple: [(102, 119), (100, 106), (106, 101), (102, 106), (106, 119),
(107, 101), (107, 106), (107, 119), (107, 107), (107, 106), (107, 101), (107,
```

107), (107, 119), (107, 106), (107, 107), (107, 101), (107, 106), (107, 107), (102, 119), (106, 106), (106, 101), (106, 107), (106, 106), (106, 119), (106, 107), (106, 106), (106, 101), (106, 107), (106, 119), (106, 106), (115, 106), (107, 111), (119, 105), (119, 115), (119, 106), (119, 100), (119, 105), (119, 119), (119, 105), (119, 100), (119, 111), (119, 105), (119, 119), (119, 101), (119, 105), (119, 119), (119, 105), (119, 101), (119, 48), (119, 105), (105, 119), (105, 48), (105, 101), (105, 105), (105, 111), (105, 48), (105, 119), (105, 105), (105, 101), (105, 48), (105, 119), (105, 105), (105, 101), (105, 48), (105, 105), (105, 119), (105, 48), (105, 101), (105, 105), (105, 48), (105, 99), (105, 110), (105, 100), (105, 107), (105, 110), (105, 102), (105, 110), (105, 100), (105, 110), (105, 102), (105, 110), (105, 100), (105, 110), (105, 102), (105, 107), (105, 100), (105, 106), (105, 102), (105, 107), (105, 106), (105, 100), (105, 102), (105, 107), (105, 106), (1, 100)]
200.743617582
##
##
From above, usual edit distance between 2 strings is less than polynomial distances of JensenShannon and L2 norms. Ordinal values were not normalized by subtracting lowest possible ordinal value of unicode.

---
365. (THEORY) BKS conjecture and Stability of Interview TQBF - related to 359 - 21 January 2017

Caution: This derivation is still experimental with possible errors.

Stability of a boolean function is defined as:
      Stability(f) = Expectation(f(x)*f(y))
where y is correlated version of x.

Stability of Majority is defined as (From page 120 of AnalysisOfBooleanFunctions - Chapter 5 Majority and Threshold Functions - https://www.cs.cmu.edu/~odonnell/boolean-analysis/)
      Stability(Majn) = Expectation(Majn(x)*Majn(y))
where y is correlated version of x.

For infinite n, Stability(Majn) is bounded by 2/pi*arcsin(rho). Similar stability measure can be derived for Interview TQBF threshold function also. Interview as a linear threshold function (LTF) can be defined as:
      w1*a1 + w2*a2 + ... + wn*an > threshold_cutoff
were each ai is an answer for a question qi and wi is the weightage associated with it. Berry-Esseen Central Limit Theorem applies for Interview threshold function. CLT implies that a random variable S=X1 + X2 + X3 + ... + Xn which is a sum of random variables Xi(s) converges to Gaussian Normal Distribution. Berry-Esseen CLT generalizes it to S=a1*x1 + a2*x2 + ... + an*xn where sum(ai^2) is normalized and = 1. Thus interview threshold function and Berry-Esseen CLT are structurally similar when weights wi are normalized and sum(wi^2) = 1. This implies interview scores are approximately Gaussian (Intuitively obvious because, bell curve corresponds to large proportion of voters scoring medium while tails correspond to very small percentage scoring very low and very high).

Stability of Interview can be equivalently defined as:
      Stability(Interview) = Expectation(Interview(x)*Interview(y))
where y is correlated version of x.

Substituting the threshold definition of interview in Stability:
      Stability(Interview) = Expectation((w1*a1+w2*a2+...
+wn*an)*(z1*b1+z2*b2+...+zn*bn))
                           = Expectation(w1*z1*a1*b1 + w1*z1*a2*b2 + ... +
wn*zn*an*bn)
                           = a1*b1*Expectation(w1*z1) + ... +
an*bn*Expectation(wn*zn)

Assuming for all ai=bi=a:
      Stability(Interview) = a^2 * (Expectation(w1*z1) + Expectation(w1*z2) + ... + Expectation(wn*zn))

Expectation(wi*zk) = 0*0*1/4 + 0*1*1/4 + 1*0*1/4 + 1*1*1/4 for 4 possible values of (wi,zk) each with probability 1/4.

=>   Stability(Interview) = a^2 * (n^2 * 0.25) = a^2*n^2/4

      Stability(Interview) = a^2*n^2/4 < 1 => a < 2/n

If Stability(Interview) > Stability(Majority), BKS conjecture is true.

=>   a^2*n^2 / 4 > 1 - 2/pi
=>   a > 2/n * sqrt(1-2/pi)
=>   a > 1.2056205488/n [For minimum case of n=2, a > 0.6028102744]

Thus for Stability(Interview) to exceed Stability(Majority) and BKS Conjecture to be true , weight per answer has to be > 1.2056/n approximately. For minimal base case n=2, a > 0.6028.

Stability is alternatively defined as:
      (+1)*Pr(f(x) = f(y)) + (-1)*Pr(f(x) != f(y))
which is just expansion of Expectation and there are two random variables (+1 for f(x)=f(y)) and (-1 for f(x) != f(y))
=> Stability(f(x)) = 1-2*Pr(f(x) != f(y))
=> Stability(f(x)) = 1-2*Sensitivity(f(x))
=> Sensitivity(f(x)) = 0.5-0.5*Stability(f(x))

NoiseSensitivity(Interview) which is the dual of Stability is defined as:
      NoiseSensitivity(Interview) = 0.5-0.5*Stability(Interview) = 0.5-0.5*a^2*n^2/4

---
366. (FEATURE-DONE) Commits - 25 January 2017 - related to 2.9 and 345

Berlekamp-Welch Polynomial Reconstruction Decoder Implementation (for Reed-Solomon codes, texts, numerical points etc.,)
---
*) Implements Berlekamp-Welch Polynomial Reconstruction Algorithm for reconstructing data from errors.
*) System of linear equations has been solved with NumPy/SciPy Linear Algebra solve().
*) Logs show how well the reconstructed polynomial closely approximates the original polynomial.
*) Error locator polynomial E is degree 1 and Numerator Q is of degree = number of points thus yielding Q/E=P (reconstructed)
*) Special case is natural language text with errors and a polynomial can be reconstructed from ordinal values of unicode

---
367. (FEATURE-DONE) Berlekamp-Welch Algorithm implementation - update for text message decoding - 27 January 2017
---
*) Using unicode values for ordinals with ord() and inverting with chr() causes overflow errors in SciPy/NumPy
Linear Algebra Equation solver (solve()) because evaluating polynomials for large messages creates huge numbers

*) Hence unicode has been replaced with a simple dictionaries - alphanumeric to numeric values and numeric to
alphanumeric values.
*) With above 2 dictionaries an example text message with error (garbled version of "thisissentence") is list
decoded with a window of possible values for each letter positions.
*) Logs show an approximate decoding of original message from message with error.
*) Garbled text with error was created from a previous execution of Berlekamp-Welch algorithm.

---
368. (THEORY) KRW Conjecture, KW relations, Majority Voting Circuit Composition - 30 January 2017 and 31 January 2017 - related to 358

Majority Voting Circuit for P(Good) binomial series RHS is a boolean circuit composition of NC Majority voting circuit with
Voter Decision Functions for each voter variable input to Majority function defined as:
Maj(m) + Voter(n) =
Maj(Voter1(x1,x2,...,xn),Voter2(x1,x2,...,xn),...,Voterm(x1,x2,...,xn)).
Assumption is Voter Decision Functions can be exactly learnt in Angluin Exact Learning model i.e Voters can have zero errors.
KRW Conjecture for boolean formula composition implies Depth(f+g) ~ Depth(f) + Depth(g) for composition f+g of two boolean formulae
(fanout 1) f and g. Applying KRW conjecture to Maj(m) + Voter(n) Majority voting composition:
Depth(Maj(m) + Voter(n)) ~ Depth(Maj(m)) + Depth(Voter(n))
where m is the number of voters and n is the number of variables per voter decision function.

KW relations R(f) for a function f imply Depth(f) = CommunicationComplexity(R(f)).
Karchmer-Wigderson relations for the composition are the following:

R(Maj,Voter):

Alice: x in Inverse(Maj + Voter)(0)
Bob: y in Inverse(Maj + Voter)(1)
Find xi and yi such that x != y

Depth(Maj + Voter) = CommunicationComplexity(R(Maj,Voter))
Thus Majority voting circuit composition is a KW relation.
There is an equivalent notion of Universal relation U(n) for KW relation defined as:
Alice: x in {0,1}^(n)
Bob: y in {0,1}^(n)
Find xi and yi such that x != y

Result in 368.2 prove a lowerbound:
Depth(g+U(n)) = CommunicationComplexity(R(g,U(n))) >= log L(g) + n - O(m*logm/n) where L(g) is the formula size of g.
If g is Majority function and U(n) is individual voter decision function, depth of composition of Majority and Voter circuits is:
D(Maj + Voter) = CommunicationComplexity(R(Maj,Voter)) >= log L(Majority) + n - O(m*logm/n)
But formula size of Majority is O(m^5.3) implying depth of Majority+VoterDecisionFunction composition is:
D(Maj + Voter) = CommunicationComplexity(R(Maj,Voter)) >= 5.3*logm + n - O(m*logm/n)
Thus depth of Majority voting circuit for P(Good) RHS is lowerbounded by a function of number of voters and number of variables per voter decision

function.

368.1 prove a result which is an extension of 355.8:
If a class C is exactly learnable in polynomial time, DTIME(n^omega(1)) is not in C. In mistake bounded learning model, learner updates the hypothesis in each round based on mistake it makes in labelling a point on hypercube. Mistake bounded learning very closely matches human experiential learning and hence human voting error. If a class C is mistake bounded learnable in time T and mistakes M then, DTIME(n + MT) is not in C yielding a separation. Corollary: If a function in NP is polynomial time exact learnable, there is no polynomial time algorithm for NP. For unbounded number of voters(m) and variables per voter decision function(n), depth of composition is unbounded. Depth Hierarchy Theorem 368.4 immediately applies for functions computable by circuits of various depths of RHS composition. Number of variables per voter and depth of composition are linearly related while Number of voters and depth of composition are logarithmically related.

Exact Learning implies zero error on both sides of P(Good) binomial series. Assuming all boolean function are exactly learnt, a Pseudorandom choice amongst those functions yields 100% perfect choice. Quoting from 359:

> Let number of voter decision functions with goodness xi = m(xi). Thus N = m(x1)+m(x2)+m(x3)+...+m(xn)
> Expected goodness of a PRG choice is:
> 1/N * summation(xi*m(xi)) = (x1*m(x1) + x2*m(x2) + x3*m(x3) + ... + xn*m(xn)) / N
> When all voter functions have goodness 1 then PRG choice in LHS of P(good) has goodness 1.

Similarly, exact learning implies all voter decision functions are 100% perfect with zero-error in LHS. Binomial series converges to 1 on both sides. LHS is a polynomial time algorithm (because there is a PRG in P to make a choice) to RHS majority voting NP problem (assuming the exactly learnt boolean functions are in NP) implying P=NP. But from 368.1, if NP is learnable in polynomial time, NP does not have polynomial time algorithms and therefore P != NP - a contradiction. Thus polynomial time exact learning of an NP boolean function (e.g 3SAT learnt from a truth table) conflicts with P=NP and binomial series convergence.

Usual process of high level human learning vis-a-vis boolean function learning is bottom-up. Atomic concepts are learnt in leaves and complexity is built from leaf to root by learning new concepts which are functions of subtrees. Recursive Lambda Function Growth algorithm for learning texts described earlier in this document is based on this paradigm but constructs a composition tree top-down (Mind grows lambda functions/circuits). Generalization of Recursive Lambda Function Growth is to learn a graph as opposed to trees. This is already done in Recursive Gloss Overlap special case by merging/removing isomorphic, repetitive subtrees while tree is grown top-down. This is why core number based classifier for the recursive gloss overlap graph works - repeating gloss nodes are connected to already discovered nodes by a new edge thereby converting a tree to graph top-down - Because most synset paths between words in a text go through "class label" vertices making them high degree and all such high degree vertices belong to a dense subgraph e.g k-core.

Hypothetical exact learner for 3SAT - NP voter decision functions :

Given a truth table of size n*2^n for n variables with truth values:

- *) Find a set of binary strings, satisfying truth values 1
- *) Construct a DNF for complement of the previous set
- *) Complement the DNF to get CNF for original set

Example - For following truth table, to construct CNF for satisfying assignments {000, 001, 011, 101, 110}:

```
      ----------------------------------------
      x1 x2 x3    truthvalue  complement
      ----------------------------------------
      0  0  0          1          0
      0  0  1          1          0
      0  1  0          0          1
      0  1  1          1          0
      1  0  0          0          1
      1  0  1          1          0
      1  1  0          1          0
      1  1  1          0          1

DNF for complement set {010,100,111} is:
      !x1x2!x3 + x1!x2!x3 + x1x2x3
and the CNF learnt is complement of DNF:
      (x1 + !x2 + x3)(!x1 + x2 + x3)(!x1 + !x2 + !x3)
with satisfying assignments {000, 001, 011, 101, 110}

This exact learner requires O(2^n) time which is exponential in number of
variables.

368.6 describes an algorithm (CDNF) to learn any boolean functions in time
polynomial in number of variables,its DNF size and CNF size. This together with
368.1 implies a polynomial time learning algorithm for NP and existence of
polynomial time learning algorithm for NP(3CNF) indicates NP does not have
polynomial time algorithms (and Expoential Time Hypothesis could be true). This
coincides with evidences towards divergence of P(Good) series and
superpolynomial size circuits for 3SAT implied in 53.16, 256, 265, 275, 318
(Majority Lemma for Hardness Amplification) and 355. This is a serious
contradiction - exact learning derandomizes both sides of P(Good) circuit. LHS
non-majority social choice is zero-error and equals 1 while RHS if NP also
converges to 1 and implies a polynomial time algorithm for NP while exact
learning forbids polynomial time algorithm for NP. Resolving this impasse
amounts to defining what is meant by exactness and zero-error in voting decision
and social choice (Perfection commonsensically enshrines zero-error decision
making for unbounded number of scenarios i.e variables while exact learning
assumes bounded set of variables)

References:
----------
368.1 Exact Learning implies Derandomization (i.e. P=BPP) and Mistake bound
learning - [KlivansKothariOliveira] -
http://www.cs.princeton.edu/~kothari/ccc13.pdf
368.2 KRW Conjecture, KW relations and Lowerbound for Depth of Boolean Circuit
Composition - [GavinskyMeirWeinsteinWigderson] -
https://video.ias.edu/sites/video/files/Meir.pdf
368.3 Generalization of Spira's Theorem - https://www.cs.utexas.edu/users/panni/
spira.pdf - Spira's theorem transforms every circuit of size S  to a circuit of
depth O(logS).
368.4 Depth Hierarchy Theorem - [RossmanServedioTan] - Theorem 1 -
https://arxiv.org/pdf/1504.03398v1.pdf - Circuit of depth <= d-1 with size <=
2^(n^(1/6(d-1))) agrees with a function of depth d on at most 0.5 +
n^Omega(-1/d) fraction of inputs.
368.5 Formula size of Majority function - [Valiant] -
http://www.sciencedirect.com/science/article/pii/0196677484900166?via%3Dihub
368.6 Exact Learning of Boolean Functions from Monotone Theory - [Nader H.
Bshouty] - http://www.cs.technion.ac.il/~bshouty/MyPapers/Exact-Learning-
Boolean-Functions-via-the-Monotone-Theory.ps - CDNF algorithm - Section 6 -
exactly learns any boolean function in time polynomial in number of variables,
size of DNF and CNF with polynomial membership and equivalence(counterexample)
queries.
368.7 BPEXP is believed to collapse to EXP - if P=BPP then BPEXP=EXP (but
converse BPEXP=EXP implying P=BPP is not known) -
https://www.ime.usp.br/~spschool2016/wp-content/uploads/2016/07/CarboniOliveiraI
```

gor.pdf
368.8 From [Klivans-Fortnow] result mentioned in 368.1, if class EXP is efficiently PAC learnable with membership queries, BPEXP is not in EXP contradicting BPEXP=EXP by homogeneous voter Condorcet Jury Theorem BPEXP circuit derandomization mentioned in 400. BPEXP=EXP implies hence that EXP is not efficiently PAC learnable.
368.9 Extended Depth Hierarchy Theorem - https://dl.acm.org/citation.cfm?id=3095799 - [Hastad-Rossman-Servedio-Tan] - For any d < logn/loglogn there is a function F computed by a read once formula of depth d but any circuit of depth d-1 and size O(2^O(n^(1/5d))) agrees with F by a fraction of 0.5 + O(n^(-1/5d)) inputs. This approximation by smaller depth circuit has applications in forecasting accuracy of previous Majority+VoterSAT boolean composition voting - as all Voters' decision SAT Circuit depth tends to infinity, accuracy of election forecasting tends to 50%.
368.10 Quasidemocracies - Banzhaf Power Index - https://gilkalai.files.wordpress.com/2018/08/kalai-icm2018.pptx - Quasi democracies are electorate in which each voter has decision SAT of n variables and as n tends to infinity, probability of forecasting the outcome of the election by any voter tends to zero as other voters vote at random. This needs to be contrasted with previous depth hierarchy ramification - number of variables per SAT versus depth of the CircuitSAT.

---
369. (FEATURE-DONE) An experimental special case k-CNF SAT Solver algorithm implementation - 31 January 2017

Algorithm:
---
*) Create a k-DNF from k-CNF input by negating literals in all clauses of k-CNF
*) Binary encode the clauses of k-DNF into set of binary strings of length n where n is number of variables
*) Compute the complement of the previous binary encoded set from set of 2^n binary strings
*) These are satisfying assignments
*) Presently requires all literals in each CNF clause
*) Implemented for simulation of a Majority Voting with Voter SATs

---
370. (THEORY) Majority Boolean Function, Sensitivity, Roth's estimate and 2-Coloring/Complement Functions of Sequences -
1 February 2017 - related to 24,338 - Important draft updates to https://arxiv.org/pdf/1106.4102v1

Boolean Majority Function finds the majority binary value of input binary string of length n. Any binary string of length n can be
considered as 2-coloring of bit position integers with corresponding complement functions for each color: f(x) is for 1s colored red and
g(x) is for 0s colored blue. For example, 1001011 is a 2-colored sequence with red (1) in bit positions 1,4,6,7 and blue (0) in bit positions
2,3,5. Complement functions for this bit position binary coloring are defined as:

f(1)=1
f(2)=4
f(3)=6
f(4)=7

and

g(1)=2
g(2)=3
g(3)=5

Natually, arithmetic progressions and monochromatic arithmetic progressions on

these bit positions can be defined. Discrepancy is the
 difference between number of colored integers in an arithmetic progression.
Roth's estimate which is the minimum(maximum(discrepancy))
for all possible colorings (i.e all possible binary strings) is >=
N^(1/4+epsilon). For majority function 2-coloring, this estimate is
the difference between number of 0s and 1s in the input to majority function
with probability equal to natural density of min(max()) arithmetic progression.
If majority function outputs 1, |f(x)| > |g(x)| and if 0, |g(x)| > |f(x)|. From
this an approximate estimate of number of 1s and 0s in majority function input
can be obtained (with probability equal to natural density):
      |f(x)| = number of 1s = [N + N^(1/4+epsilon)]/2 if majority function
outputs 1
      |g(x)| = number of 0s = [N - N^(1/4+epsilon)]/2 if majority function
outputs 1
and vice versa if majority function outputs 0. For previous example, majority is
1 and :
      |f(x)| = number of 1s = [7 + 7^0.25]/2 = 4.313288 ~ 4
      |g(x)| = number of 0s = [7 - 7^0.25]/2 = 2.6867 ~ 3
Sensitivity of Majority function is number of bits to be flipped to change the
outcome which is nothing but the discrepancy (|f(x)| - |g(x)|) wth probability
equal to natural density and is lowerbounded by Roth's estimate. Formally,
Sensitivity(Majority) > N^(1/4+epsilon). An important special case is when input
to majority is a binary string 2-colored by prime-composite complement functions
in bit positions i.e prime bit positions are 1 and composite bit positions are 0
and majority function outputs 0 or 1 (Approximate number of primes less than N =
pi(N) ~ N/logN from Prime Number Theorem).

---
371. (FEATURE-DONE) Reinforcement Learning - ThoughtNet based Recommender
Systems Implementation Update - 2 February 2017

---
*) Recursive Gloss Overlap classifier disambiguation has been updated to invoke
NLTK lesk algorithm function() with options to
choose between PyWSD lesk implementation and best_matching_synset() primitive
disambiguation native implementation
*) New usecase shopping cart example with book reviews has been added
*) Fixed a major bug in RecursiveGlossOverlap Classifier in definition graph
construction : Edges for all lemma names of previous level synsets were added
which bloated the graph and somewhat skewed core number of the graph. Changed it
to create edge only for one lemma name. All other Recursive Gloss Overlap
implementations in InterviewAlgorithm/ folder already have this fix. Regenerated
the hypergraph for shoppingcart3 example Logs for this have been added to
testlogs/

---
372. (FEATURE-DONE) Recursive Gloss Overlap classifier update - 3 February 2017

Uncommented NetworkX matplotlib graph plotting to get a graph for a sample
document definition graph. Logs and PNG file for this classification have been
added to testlogs/. Also did some experimentation with various documents and
classification based on core number has been reasonably accurate (relevance of
unsupervised core number based classification to the document's actual class
decreases with core number) and top percentage classes reflect the subject
matter of the document. Also best_matching_synset() lesk disambiguation
implemented in AsFer and NLTK lesk() functions are faster than PyWSD
simple_lesk().

---
373. (THEORY and FEATURE-DONE) CNFSAT solver update - polynomial time

approximation - 5 February 2017

Solves CNFSAT by a Polynomial Time Approximation scheme:
	- Encode each clause as a linear equation in n variables: missing variables and negated variables are 0, others are 1
	- Solve previous system of equations by least squares algorithm to fit a line
	- Variable value above 0.5 is set to 1 and less than 0.5 is set to 0
	- Rounded of assignment array satisfies the CNFSAT with high probability
Essentially each CNF is represented as a matrix A and solved by AX=B where X is the column vector of variables and B is identity column vector

---
374. (FEATURE-DONE) Java Spark Streaming - Generic Stream Receiver Implementation - 6 February 2017

Spark Streaming Java implementation to receive generic stream of unstructured text data from any URL and Sockets has been added to NeuronRain AsFer java/bigdata_analytics/. It is based on Spart 2.1.0 + Hadoop 2.7 single node cluster. Following are the setup prerequisites:

*) Oracle Java 8 compiler and runtime and PATH set to it
*) export CLASSPATH=/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/spark-streaming_2.11-2.1.0.jar:.:/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/scala-library-2.11.8.jar:/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/spark-sql_2.11-2.1.0.jar:/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/spark-core_2.11-2.1.0.jar:/usr/lib/jvm/java-8-oracle/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
*) Compiled as: javac SparkGenericStreaming.java
) JAR packaging of SparkGenericStreaming.class : jar -cvf sparkgenericstreaming.jar *class
*) Example Spark submit commandline for 2 local threads (2 cores) and data streamed from twitter :
/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/bin/spark-submit --class SparkGenericStreaming --master local[2] sparkgenericstreaming.jar "https://twitter.com/search?f=tweets&vertical=news&q=Chennai&src=typd"

Spark Streaming Java implementation complements Streaming Abstract Generator Python native abstraction implemented in AsFer. Advantage of Spark Streaming is it can process realtime streamed data from many supported datasources viz.,Kinesis,Kafka etc., This implementation is generic, customized for arbitrary streaming data.

---
375. (FEATURE-DONE) Java Spark Streaming for Generic data - Jsoup ETL integration - 7 February 2017
---
*) Jsoup HTML parser has been imported into SparkGenericStreaming.java and a very basic Extract-Transform-Load is
performed on the URL passed in with a special boolean flag to use Jsoup
*) Jsoup connects to URL and does a RESTful GET of the HTML, extracts text from HTML and stores the text in Spark RDD
*) receive() is periodically invoked and word count is computed
*) foreachRDD() is invoked on the DStream to iterate through all words in DStream.
*) forEach() invokes a lambda function to print the word text
*) Logs for this have been added to spark_streaming/testlogs/

spark-submit commandline requires additional classpath to jsoup .jar with --jars

option as below:
/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/bin/spark-submit --class SparkGenericStreaming --master local[2] --jars /home/shrinivaasanka/jsoup/jsoup-1.10.2.jar sparkgenericstreaming.jar "https://twitter.com/search?f=tweets&vertical=news&q=Chennai&src=typd" 2>&1 > testlogs/SparkGenericStreaming.out.7February2017

---
376. (THEORY and FEATURE-DONE) Approximate SAT solver update - 8 February 2017 - related to 373

Updated CNFSATSolver.py to create set of random 3CNFs and verify if the least squares assignment is satisfied. Presently it creates randomly concatenated CNFs of 10 variables and 10 clauses in an infinite loop and prints a percentage of satisfied 3CNFs so far by least square heuristic. Logs for this percentage after few hundred iterations has been committed to testlogs/. It shows an intriguing convergence at 80% (i.e 80% of 3CNFs are satisfied) after few hundred iterations of random CNFs. A caveat has to be mentioned here on hardness of approximation and PCP theorem: MaxSAT and Clique do not have Polynomial Time Approximation Scheme unless P=NP. PTAS usually creates a solution within 1+/-epsilon distance from optimal. MaxSAT problem tries to maximize number of clauses satisified in a CNF. But this least square SAT solver is exact SAT solver and the percentage of 80% found in testlogs/ implies 80% of CNFs were satisfied. This could heuristically imply 80% of clauses per CNF were satisfied on the average. Though this percentage is specific to 10 clauses * 10 variables, similar asymptotic convergence to ~79% was observed for 5 clauses * 5 variables. Assuming this implies epsilon to be 0.2 in general (average 20% distance from optimal for all CNFs), could imply a PTAS (because solving system of equations by Least Squares or Gaussian Elimination is polynomial in number of clauses) for MaxSAT which satisfies 80% clauses per CNF on average and P=NP.(Related: Unique Games Conjecture)

References:

376.1 PCP theorem and Hardness of Approximation - http://www.cs.jhu.edu/~scheideler/courses/600.471_S05/lecture_9.pdf
376.2 PCP theorem and Hardness of Approximation - http://pages.cs.wisc.edu/~dieter/Courses/2008s-CS880/Scribes/lecture17.pdf

---
377. (FEATURE-DONE) Java Spark Streaming Generic Receiver update - 9 February 2017
---
*) Objectified the SparkGenericStreaming more by moving the SparkConf and JavaStreamingContext into
class private static data.
*) new method SparkGenericStreamingMain() instantiates the Spark Context and returns the JavaPairDStream<String,Integer> wordCounts object
*) main() static method instantiates SparkGenericStreaming class and receives wordCounts JavaPairDStream from SparkGenericStreamingMain()
method
*) Spark JavaStreamingContext output operations are invoked by start() and print actions by lambda expressions.
*) SparkConf and JavaStreamingContext variables have to be static because Not Serializable exceptions are thrown otherwise.
*) sparkgenericstreaming.jar has been repackaged by recompilation.

---
378. (FEATURE-DONE) Spark Streaming Java Update - Java Bean DataFrame(Dataset)

creation, Hive Metastore support and persistence of
DataFrame to Hive and as a Parquet file - 10 February 2017

---
*) New JavaSparkSingletonInstance class for singleton Spark Context has been added.
*) New Java Bean Word.java has been added for creating DataFrame from JavaRDD iterable
*) DataFrame has been persisted to filesystem as .parquet file in overwrite mode of DataFrameWriter
*) DataFrame has also been persisted to Hive MetaStore db which Spark supports (Shark SQL) with saveAsTable()
*) SparkGenericStreaming has been rewritten to do transformations on words JavaRDD iterable with VoidFunction() in
2-level nested iterations of foreachRDD() and forEach()
*) map() transformation for each row in JavaRDD throws a null pointer exception and inner call() is never invoked.
*) This has been remedied by replacing map() with a forEach() and storing the rows in each RDD in an ArrayList
*) With Hive MetaStore + Shark integration, Streaming_AbstractGenerator.py abstraction now has access to Java Spark Streaming
because it already supports Hive via thrift and NeuronRain now has ability to analyze any realtime or batch unstructured data.
*) Primitive ETL to just scrape words in URL text sufficiently tracks trending topics in social media.
*) Hive metastore db, Spark warehouse data have been committed and Logs for this has been added to testlogs/
*) sparkgenericstreaming.jar has been repackaged with recompilation
*) Updated spark-submit commandline: /home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/bin/spark-submit --class SparkGenericStreaming --master local[2] --jars /home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/scala-reflect-2.11.8.jar,/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/spark-catalyst_2.11-2.1.0.jar,/home/shrinivaasanka/jsoup/jsoup-1.10.2.jar sparkgenericstreaming.jar "http://www.facebook.com"
*) Updated CLASSPATH: export CLASSPATH=/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/scala-reflect-2.11.8.jar:/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/spark-catalyst_2.11-2.1.0.jar:/home/shrinivaasanka/jsoup/jsoup-1.10.2.jar:/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/spark-streaming_2.11-2.1.0.jar:.:/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/scala-library-2.11.8.jar:/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/spark-sql_2.11-2.1.0.jar:/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/jars/spark-core_2.11-2.1.0.jar:/usr/lib/jvm/java-8-oracle/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

---
---
379. (THEORY) Machine Translation, Recursive Gloss Overlap Graphs in different Natural Languages and Graph Homomorphisms - 13 February 2017
- related to 178

Machine Translation is equivalent to graph homomorphism between 2 recursive gloss overlap graphs of a text in 2 natural languages where
homomorphism f for translation is defined as:

if (u,v) is in E(G1) then (f(u),f(v)) is in E(G2)

for recursive gloss overlap graphs G1 and G2 for languages L1 (e.g. English) and L2 (e.g Tamizh) constructed from respective ontologies.
In previous definition, u and v are two words in natural language L1 and f(u) and f(v) are two words in natural language L2.
This relaxes the isomorphism requirement mentioned in 178. Set of all digraphs G such that there is a homomorphism from G to a graph H is
denoted by L(H). Guessing the set of digraphs G for a digraph H implies finding set of all translations for a text from a set of natural
languages to another natural language. This guessing problem is known by

Dichotomy Conjecture to be either in P or NP-Complete.

---
380. (FEATURE-DONE) Approximate SAT solver update - 14 February 2017

---
*) Changed number of variables and clauses to 14 and 14 with some additional debug statements.
*) After 4000+ iterations ratio of random 3SAT instances satisfied is: ~70%

---
381. (FEATURE-DONE) Java Spark Streaming with NetCat WebServer update and some analysis on HiveServer2+Spark Integration - 18 February 2017

---
*) Rewrote SparkGenericStreaming.java mapreduce functions with Spark 2.1.0 + Java 8 lambda functions
*) Enabled Netcat Socket streaming boolean flag instead of URL socket with Jsoup
*) With above changes (and no HiveServer2), socket streaming receiver works and Hive saveAsTable()/Parquet file saving works.
*) HiveServer2 was enabled in Spark Conf by adding hive-site.xml in spark/conf directory. (Two example hive-site.xml(s) have been
committed to repository.)
*) Spark 2.1.0 was started with hive-site.xml to connect to HiveServer2 2.1.1. This resulted in 2 out of heapspace errors as documented
in testlogs/ with exception "Error in instantiating HiveSessionState".
*) Such OOM errors with HiveServer2 and Spark have been reported earlier in Apache JIRA. Instead of TTransport, FrameTransport was also
tried which caused "Frame size exceeds maximum frame size" errors. SASL was also disabled in hive-site.xml.
*) Reason for such OOM errors in HiveServer2 seems to be heavy memory footprint of HiveServer2 and Spark together and CPU usage reaches 100%
for both cores. Might require a high-end server with large RAM size so that heapspace is set to atleast 8GB. HiveServer2 is known to be
memory intensive application. Also Spark only supports Hive 1.2.1 and not Hive 2.1.1 which could be the reason.
*) Increasing Java Heap space in spark config file (spark.executor.memory and spark.driver.memory) also did not have effect.
*) Setting HADOOP_HEAPSIZE also did not have effect. OOM error occurs while instantiating HiveSessionState and bootstrapping Hive databases
(get_all_databases). Isolated beeline CLI connection works though with HiveServer2.
*) Hence presently Spark Streaming works with Spark provided metastore_db Hive Support and Parquet file support.
*) testlogs/ have stream processing logs for NetCat (NC) webserver.

---
382. (FEATURE-DONE) Spark Version Upgrade for Recursive Gloss Overlap Graph Intrinsic merit and Discrete Hyperbolic Factorization Cloud Implementations - 21 February 2017

---
*) Re-executed Interview Algorithm and Factorization Spark Cloud implementations with Spark 2.1.0 and Cython optimization for Recursive
Gloss Overlap Graph construction
*) Earlier these were benchmarked with Spark 1.5.x
*) Added debug statements in python-src/DiscreteHyperbolicFactorizationUpperbound_Bitonic_Spark.py and file locations updated. Logs
committed to python-src/testlogs and python-src/InterviewAlgorithm/testlogs

---
383. (THEORY) Recursive Gloss Overlap Graphs, Intrinsic Merit and Korner Graph Entropy - related to 348 - 22 February 2017

Graph representation of text and deriving a quantitative intrinsic merit from recursive gloss overlap definition graph by projecting
WordNet onto the document has been described previously. Spectral Graph Theory qualitatively analyzes the graph in terms of eigenvalues of
its Laplacian. Quantitative Graph Theory provides various statistical tools to quantify a graph in terms of its complexity. Measuring
meaningfulness of a text has been translated to the problem finding easy paths between concepts and keywords in the text. This is computational linguistic Symbol Grounding problem of learning from dictionary definitions (refer 216.3).
Korner's Graph Entropy defines the entropy of a
graph G as follows:

KE(G) = minimum(I(X,Y))

where X is a randomly chosen vertex in V(G) and Y is an independent set of G (set of vertices with no edges between them). I(X,Y) is the
probabilistic mutual information of two random variable X and Y defined as:

I(X,Y) = double_summation(p(x,y)log(p(x,y)/p(x)p(y)))

where p(x) is probability of X=x, p(y) is probability of Y=y and p(x,y) is a joint probability distribution of X=x and Y=y.

An alternative definition of Korner Entropy of a graph G is :

KE(G) = minimum [- sum_v_in_V(G) {1/|V(G)| * log(Pr[v in Y])}]

i.e minimum of summations over product of probability of a randomly chosen vertex v in V(G) and log likelihood probability of v in an independent set Y of G.

Mutual information is the generalization of Shannon information entropy when two dependent random variables are involved. Korner's entropy quantitatively captures the complexity and merit of a graph representation of text quite effectively because mutual information is the amount of knowledge one random variable has about the other. This is quite useful for text meaningfulness - two words with high mutual information are quite likely to be closely related. More precisely, from definitions of Korner Entropy, mutual information of a word vertex and an independent set of word vertices in recursive gloss overlap graph is the "extent of relatedness" between a word and an independent set of words it is part of. When a word is in an independent set of word vertices, there exists no relation among them - no word vertex is in dictionary gloss definition of other word vertices. By summing up log likelihoods for all such vertices, a measure of "meaninglessness" is obtained. Dual of this meaninglessness is meaningfulness (document ranking by meaninglessness is inversion of ranking by meaningfulness) obtained by (-) sign and minimum(). Maximum Korner Entropy is log(|V|) for complete graphs and minimum is zero for empty graphs. Any document graph would have entropy merit value between 0 and log|V|. Computing Korner Entropy is NP-hard since finding stable independent sets is NP-hard.

References:

383.1 Korner Entropy - [JaikumarRadhaKrishnan] - http://www.tcs.tifr.res.in/~jaikumar/Papers/EntropyAndCounting.pdf
383.2 Entropy of Graphs - [MowshowitzDehmer] - www.mdpi.com/1099-4300/14/3/559/pdf
383.3 Entropy Centrality of Graphs - [Qiao-Shan-Zhou] - http://www.mdpi.com/1099-4300/19/11/614/pdf - influence of a vertex is defined in terms of Shannon Entropy (-plogp) and Subgraph Degree Centrality
383.4 Entropy measures for networks - [Anand-Bianconi] - https://arxiv.org/pdf/0907.1514.pdf - Relations between Shannon Entropy, Gibbs Entropy and Von Neumann Entropy of networks
383.5 Graph Energy and its relation to average degree k - http://iopscience.iop.org/article/10.1088/1742-6596/604/1/012024/pdf - Energy of

a Graph is sum of absolute values of its adjaceny matrix eigenvalues. Quoted excepts: "...When the network is very sparse, graph energy E will be monotonically increasing with k. With slow increase of intensity, energy E will be still proportional to k but with appearing of some weak fluctuations. However, while the network is becoming highly dense, the graph energy will drop sharply. And the value of k that corresponds to the largest graph energy is about 0.8N...". This threshold relation between average degree and energy of graph has parallels to Bianconi-Barabasi least energy intrinsic fitness which implies low energy vertices attract large number of links. Similarly, Graph Energy which is cumulative estimate for all vertices, decreases when the graph has lot of edges which is possible only if average degree of the vertices is high. High average degree by Bianconi-Barabasi model implies least energy and high intrinsic fitness.
383.6 Graph Energy and its meaning - https://arxiv.org/pdf/1704.00779.pdf

---
384. (THEORY) Neural Tensor Networks Reasoning and Recursive Gloss Overlap merit scoring - 23 February 2017

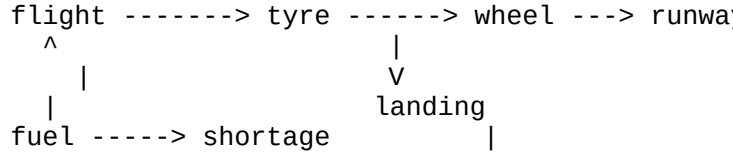
Neural Tensor Networks described in 384.1 generalizes a linear neuron into a multidimensional tensor and assigns a score for relation
between two entities e1 and e2. This is described in the context of an ontology (e.g WordNet) where a relation between two words w1 and w2
is assigned a score based on various distance measures. Recursive Gloss Overlap graph which is a gloss definition graph has a single relation
between any two words - w1 "is in dictionary/gloss definition of" w2. Present implementation does not quantify the extent of relatedness
between two words connected by an edge in Recursive Gloss Overlap graph. This definition graph can be considered as a Neural Tensor Network
and each edge can be scored as a neural tensor relation between two word vertices. Cumulative sum total score of all edges in this graph is a quantitative intrinsic merit measure of the meaningfulness and intrinsic merit. Thus any document's definition graph is a set of tensor neurons which together decide the merit of the text. Together with Korner Entropy complexity measure, Sum of Tensor Neuron scores effectively weight the text.

References:

384.1 Neural Tensor Networks - [SocherManningNg] - http://nlp.stanford.edu/pubs/SocherChenManningNg_NIPS2013.pdf

---
385. (THEORY) Recursive Lambda Function Growth, Random Walks on Recursive Gloss Overlap graph and Text Comprehension - 25 February 2017 - related to 46, 47, 216, 230 and 384

As mentioned in previous section, two word vertices x1 and x2 are connected by an edge weight determined by tensor neuron (x1 R x2) for some relation R. Human comprehension of text can be approximated by set of compositional lambda functions created from converging random walks or DFS Tree on the recursive gloss overlap graph. For example, following definition graph for a text "Flight landed on runway because
of fuel shortage" is created from thesaurus/sdictionary/gloss definitions of an ontology:



V
gear

Example Random walks on previous graph:
---
flight ---> tyre ----> wheel ---> landing ---> gear
flight ----> fuel ----> shortage

If Tensor Neurons are applied for each edge, this graph is a weighted directed graph where weight is determined by Tensor Neuron. If each relation is a lambda function following the Recursive Lambda Function Compositional Growth defined in 216, every random walk can be mapped to a lambda function composition tree. For example, following random walk:

fuel ----> flight ---> tyre ----> wheel ---> landing ---> gear

is tensor neuron labelled with lambda functions:

fuel --(requiredby)---> flight----(has)---> tyre ---(has)---> wheel ---(does) ---> landing ---(requires)---> gear

where lambda functions for each edge with word vertices as parameters are:

```
f1 = requiredby(fuel, flight)
f2 = has(flight, tyre)
f3 = has(tyre, wheel)
f4 = does(wheel, landing)
f5 = requires(landing, gear)
```

Composition is done left associative creating following tree recursively:

```
f5(landing,gear)
f4(wheel, f5(landing, gear))
f3(tyre, f4(wheel, f5(landing, gear)))
f2(flight, f3(has, f4(wheel, f5(landing, gear))))
f1(fuel, f2(flight, f3(tyre, f4(wheel, f5(landing, gear)))))
```

Previous composition tree is done for each Markov Chain Random Walk on the Definition Graph. Difference between composition in 216 and previous is: Recursive Lambda Function Compositional Growth is done on a flat sentence in 216, while it is applied to a random walk on definition graph here. Because neuron tensors are equivalent to a lambda function per edge, each function fi can be made to return a relevance merit score which is fed to next layer - in essence each random walk lambda function composition tree is a multilayer tensor neuron. Similar process can be applied to paths in a DFS tree also. Inorder traversal of the composition tree yields an equivalent expanded text of original text (In above example it is "fuel requiredby flight has tyre has wheel does landing requires gear") . Each random walk results in a tree-structured multilayer tensor neuron. Equivalently, each lambda function tensor could return a pictorial meaning and each composition "operates" on the tensor arguments to return a new animated version to next level up the tree (i.e meaning is represented as visuals). Ontology required for this is an ImageNet than a WordNet.

Advantage of Random Walks is it mimicks the randomness of deep learning in human text comprehension. Typical visual comprehension reads a passage atleast once scanning the keywords and inferring grammatical connectives between keywords. After keywords are known, they have to be related to infer meaning. An ontology provides paths between keywords and how they are related. After all edges and paths are known, collective meaning has to be evaluated by traversing the graph. Meaning is accrued over time by traversing the relations between keywords compositionally. This traversal is a random walk on the definition graph. Previous lambda function growth for each random walk on graph builds meaning recursively. Covering time of random walk is the number of steps required to visit all vertices. Mixing time of random walk is the number of steps required to reach stationary distribution (i.e any further random walk is no different from previous walks). These two measures have been widely studied for weighted

directed graphs and are good estimators of time required to deep-learn and comprehend a text. Trivial DFS search is quite static and stationary distribution in Mixing time of random walk is the state when a text has been fully understood and any further reading does not add more meaning. Mixing time in undirected graph is proportional to Isoperimetric number or Cheeger's constant. Isoperimetry measures amount of bottleneck in a graph. For directed weighted graph, Conductance is the equivalent isoperimetric measure. When isoperimetry is high, random walk converges quickly implying faster text comprehension. For a partition of the vertex set of a graph (S, T), conductance of the cut Φ(S,T) = u ∈ S,v ∈ T sum(a(u,v)) / minimum (sum(w ∈ S dw), sum(w ∈ T dw)) where dw = for all (w,z) sum(a(w,z)) and a(s,t) is the weight of an edge i.e tensor neuron lambda function score. PageRank of the Recusive Gloss Overlap graph is converging random walk - PageRank along with Core Number of definition graph is already known to classify texts in unsupervised setting by ranking the vertices.

References:

385.1 Random Walks Survey - http://www.cs.elte.hu/~lovasz/erdos.pdf
385.2 Random Walks on Weighted Directed Graphs - http://www.cs.yale.edu/homes/spielman/eigs/lect7.pdf

---
386. (FEATURE-DONE) Java Spark Generic Streaming and Streaming Abstract Generator Integration - 27 February 2017
---
*) Enabled URLSocket in Spark Generic Streaming and crawled an example facebook and twitter
streaming data stored into Parquet files and Spark metastore
*) Added Spark Streaming Data Source and Parquet Data Storage to Streaming_AbstractGenerator.py
and updated iterator to read DataFrames from word.parquet Spark Streaming storage.
*) Spark DataFrames are iterated by accessing word column ordinal and yielded to any application of
interest.
*) As an example Streaming HyperLogLogCounter implementation has been updated to read from Spark
Parquet streaming data storage and cardinality is computed
*) Example Spark Generic Streaming logs with URLSockets and HyperLogLogCounter with Spark Parquet
data storage have been added to testlogs in python-src and java-src/bigdata_analytics/spark_streaming/testlogs
*) Thus an end-to-end Java+Spark+Python streaming ETSL pipeline has been implemented where:
- Extract is done by JSoup crawl
- Transform is done by Spark RDD transformations
- Store is done by Spark Hive and Parquet storage
- Load is done by Streaming Abstract Generator iterator/facade pattern
*) Streaming_HyperLogLogCounter.py instantiates SparkSession from pyspark.sql. Usage of SparkSession requires executing this through spark-submit (Usual python <file> commandline doesnot work) as below:
/home/shrinivaasanka/spark-2.1.0-bin-hadoop2.7/bin/spark-submit Streaming_HyperLogLogCounter.py 2>&1 > testlogs/Streaming_HyperLogLogCounter.out.SparkStreaming.27February2017

---
387. (FEATURE-DONE) Streaming Algorithms Implementations for Spark Streaming Parquet Data Storage - 28 February 2017

---
*) All Streaming_<algorithm> implementations have been updated to have Spark Streaming
Parquet file storage as input generators.
*) New Streaming_SocialNetworkAnalyis.py file has been added to do sentiment analysis for
Streaming Data from Spark Parquet file storage
*) Streaming_AbstractGenerator.py has been updated to filter each Spark Parquet DataFrame to remove
grammatical connectives and extract only keywords from streamed content
*) Logs for all Streaming_<algorithm>.py and Streaming_SocialNetworkAnalysis.py executed against
Spark Streaming Parquet data have been committed to testlogs/
*) Streaming_SocialNetworkAnalysis.py presently iterates through word stream generated by
Streaming_AbstractGenerator.py and does Markov Random Fields Clique potential sentiment analysis.
*) Streaming_AbstractGenerator.py Spark Parquet __iter__() clause presently returns word stream which
can be optionally made to return sentences or phrases of specific length
*) All python streaming implementations instantiating Streaming_AbstractGenerator.py have to be executed with $SPARK_DIR/bin/spark-submit

---
388. (FEATURE-DONE) Major rewrite of BackPropagation Implementation for arbitrary number of Neuron input variables layer - 2 March 2017
---
*) BackPropagation algorithm code has been rewritten for arbitrary number of input layer, hidden layer and output layer variables.
*) Some Partial Differential Equations functions have been merged into one function.
*) This was executed with 6 variable input layer neuron, 6 variable hidden layer and 6 variable output layer with 2*6*6=72 weights
for each of 72 activations.
*) Logs for this have been committed to testlogs/
*) For 100000 iterations, error tapers to ~10^-29 as below

Error after Backpropagation- iteration : 99998
1.05827458078e-29
Layers in this iteration:
###############
Input Layer:
###############
[0.23, 0.11, 0.05, 0.046, 0.003, 0.1]
###############
Hidden middle Layer:
###############
[0.07918765738490302, 0.17639329000539292, 0.3059104560477687,
0.06819897810162112, 0.13596389556531566, 0.18092367420130773]
###############
Output Layer:
###############
[0.29999999999999977, 0.53, 0.11000000000000039, 0.09000000000000002,
0.010000000000004578, 0.2099999999999999]
Weights updated in this iteration:
[0.048788506679748, 0.11426171324845727, 0.24603459114714846,
0.0795780558315572, 0.17784209754093078, 0.18901665972998824,
0.11262698024008909, 0.20899193850252168, 0.5190303819914409,
0.3056462222980023, 0.5043023226568699, 0.6597581902164429, 0.16317511417689112,
0.6856596163043989, 1.0387023103855846, 0.2606362555325616, 0.9902995852922012,

```
1.0606233996394616, 0.03792106334721366, 0.08894920940249851,
0.19690317303590538, 0.06956852060077344, 0.15685074966243606,
0.1617685764906898, 0.07393365494416165, 0.11785754559950794,
0.32914500297950156, 0.2691055311488234, 0.4255090307780099, 0.5588219323805761,
0.17422407739745155, 0.7118075853635217, 0.335920938970278, 0.08421750448989332,
0.18955175416917805, 0.2131459459234997, 0.18585916895619456,
0.2871421205369935, 0.4097382856269691, 0.09927665626493291, 0.2820455537957722,
0.44170499922267087, 0.4181874495051114, 0.6343547541074734, 0.5019659876526656,
0.6341824150669009, 0.9311854573920668, 0.415739542966714, 0.8116648622103757,
0.7642211240783323, -0.27787045024304763, -0.03796350343576473, -
0.06897746410924578, -0.12106324365840128, 0.09688102161356603,
0.0833653488478167, 0.025844281679456543, 0.01924251728820474,
0.1166534752079898, 0.22544113767193752, 0.2565459528164421, 0.2598246679566054,
-0.2448286735461752, 0.48545168615654455, -0.12033616818940464, -
0.17996884551601777, 0.09161370830913909, 0.16523481156548922,
0.22079944846849042, 0.4246703906819317, 0.05813513760785996,
0.49187786590191596]
Recomputing Neural Network after backpropagation weight update
Error after Backpropagation- iteration : 99999
1.05827458078e-29
Layers in this iteration:
###############
Input Layer:
###############
[0.23, 0.11, 0.05, 0.046, 0.003, 0.1]
###############
Hidden middle Layer:
###############
[0.07918765738490302, 0.17639329000539292, 0.3059104560477687,
0.06819897810162112, 0.13596389556531566, 0.18092367420130773]
###############
--------------------------------------------------------------------------------
---------------------------------

--------------------------------------------------------------------------------
-----------------------------
389. (FEATURE-DONE) Convolution Network Final Neuron Layer Integrated with
BackPropagation Implementation- 3 March 2017
--------------------------------------------------------------------------------
-----------------------------
*) New python module has been added to repository to invoke BackPropagation
algorithm
implementation in final neuron layer of convolution network.
*) This creates a multilayer perceptron from maxpooling layer variables as input
layer and
backpropagates for few iterations to do weight updates.
*) Maxpooling layer is a matrix of 5*5=25 variables. With 25 variable input
layer, backpropagation
is done on 25*25*2=1250 activation edges with weights(25 inputs * 25 hidden + 25
hidden * 25 outputs).
*) Logs for this have been added to testlogs/
*) Presently expected output layer has been hardcoded to some constant

--------------------------------------------------------------------------------
-------------------------------------------------------------
390. (THEORY) Prime-Composite Complementation/2-coloring, Riemann Zeta Function
and Sum of Eigenvalues - Elementary Analysis - 6 March 2017
- related to 19, 24, 319, 323, 338, 370
--------------------------------------------------------------------------------
-------------------------------------------------------------
Special case of complementation/2-coloring is prime-composite coloring where set
of natural numbers is 2-colored as primes and composites.
Here terminology of complementation and 2-coloring is used interchangeably. But
complementation is a generic notion for functions over
```

reals too. Riemann Zeta Function and truth of Riemann Hypothesis imply a pattern in distribution of primes and hence pattern in prime-composite coloring. Thus Riemann Zeta Function is a 2-coloring scheme for set of natural numbers.

Let q^s + q^(1-s) = v be an eigenvalue of a graph where s is a complex exponent (s=a+ib). It can be written as:

q^2s + q = vq^s

q^2(a+ib) + q = vq^(a+ib)

q^2a*q^2ib + q = vq^a*q^ib

q^2a(e^i2blogq) + q = vq^a[e^iblogq] by rewriting q^ib = e^iblogq

q^2a[cos(2blogq) + isin(2blogq)] + q = vq^a[cos(blogq) + isin(blogq)]

Equating Re() and Im():

q^2a[cos(2blogq)] + q = vq^a[cos(blogq)]

q^2a[sin(2blogq)] = vq^a[sin(blogq)]

Ratio of Re() and Im() on both sides:

$$\frac{q^{2a}[\sin(2b\log q)]}{q^{2a}[\cos(2b\log q)] + q} = \frac{vq^{a}[\sin(b\log q)]}{vq^{a}[\cos(b\log q)]}$$

$$\frac{q^{2a}[2\sin(b\log q)\cos(b\log q)]}{q^{2a}[\cos(2b\log q)] + q} = \frac{[\sin(b\log q)]}{[\cos(b\log q)]}$$

q^2a[2*cos(blogq)*cos(blogq)] = q^2a[cos(2blogq)] + q

q^2a[2*cos(blogq)*cos(blogq)] = q^2a[cos(blogq)*cos(blogq) - sin(blogq)*sin(blogq)] + q

q^2a[cos(blogq)*cos(blogq) + sin(blogq)*sin(blogq)] = q

q^2a = q => a = 0.5

When s=0.5 with no imaginary part and q+1 is regularity of a graph, v = 2*sqrt(q) and corresponding graph is a q+1 regular Ramanujan Graph.

[This is already derived in:
https://sites.google.com/site/kuja27/RamanujanGraphsRiemannZetaFunctionAndIharaZetaFunction.pdf,
https://sites.google.com/site/kuja27/RZFAndIZF_25October2014.pdf,
http://sourceforge.net/p/asfer/code/HEAD/tree/python-src/ComplFunction_DHF_PVsNP_Misc_Notes.pdf]

Let H = {G1,G2,G3,...} be an infinite set of graphs with eigenvalues 1^s + 1^(1-s), 2^s + 2^(1-s), 3^s + 3^(1-s),....

Sum of these eigenvalues for infinite set of graphs has following relation to Riemann Zeta Function for s=w+1:
Sum_q=1_to_inf(q^s + q^(1-s)) = [1 + 1/2^w + 1/3^w + ...] + [1 + 2^(w+1) + 3^(w+1) + ...] = RiemannZetaFunction + [1 + 2^(w+1) + 3^(w+1) + ...]

Sum_q=1_to_inf(q^s + q^(1-s)) - [1 + 2^(w+1) + 3^(w+1) + ...] = RiemannZetaFunction

Thus Riemann Zeta Function is written as difference of sum of eigenvalues for an infinite set of graphs and another infinite exponential series.
Hermitian Real Symmetric matrices have real eigenvalues which is true for undirected graphs. For directed graphs, eigenvalues are complex numbers. Because of this condition, elements of set H have to be directed graphs. Relation between eigenvalues of two matrices M1 and M2 and eigenvalues of their sum M1+M2 is expressed by [Weyl] inequalities and [Knutson-Tao] proof of Horn's conjecture. Sum of eigenvalues of M1+M2 is equal to sum of eigen values of M1 + sum of eigen values of M2 (Trace=sum of eigen values=sum of diagonal elements). The set H has one-to-one mapping with set of natural numbers and is 2-colored as

prime and composite graphs based on index (each graph is monochromatic). Sum of adjacency matrices of H is also a directed graph (if all Gi have same number of vertices) denoted as G(H).

References:

390.1 Complex Analysis - [Lars Ahlfors] - Page 30 - Poles and Zeros, Page 213 - Product Development, Riemann Zeta Function and Prime Numbers
390.2 Honeycombs and Sums of Hermitian Matrices - http://www.ams.org/notices/200102/fea-knutson.pdf

---
391. (FEATURE-DONE) DeepLearning (ConvolutionNetwork-BackPropagation) JPEG files with Python Imaging Library (PILlow) - 6 March 2017
---
*) Added import of PIL Image library to open any image file and convert into a matrix in Integer mode
*) This matrix is input to Convolution and BackPropagation code
*) Logs of Convolution and BackPropagation for sample images have been committed to testlogs/

---
392. (FEATURE-DONE) DeepLearning (ConvolutionNetwork-BackPropagation) tobit() update - 7 March 2017
---
*) Updated python-src/DeepLearning_ConvolutionNetwork_BackPropagation.py tobit() function to map each pixel to a decimal value < 1
*) Increased BackPropagation iteration to 100.
*) Logs for this have been committed to testlogs/. With this final neurons for max pooling layer are sensitive to changes in image pixels

---
393. (FEATURE-DONE) DeepLearning (ConvolutionNetwork-BackPropagation) image_to_bitmatrix() update - 8 March 2017
---
*) Refactored the tobit() function - moved it to a new directory image_pattern_mining and reimplemented it with a new image_to_bitmatrix() function in ImageToBitMatrix.py by enumerating through the pixel rows and applying map(tobit)
*) This removed the opaqueness related 255 appearing for all pixels in bitmap.
*) Each pixel is multiplied by a fraction. Mapping it to binary digits 0,1 based on [0-127],[128-255] intervals for pixel values
does not work well with BackPropagation of MaxPooling in Convolution Network. Instead multiples of a small fraction makes the
MaxPooling layer quite receptive to changes in image pixel values.
*) Added few handwriting recognition example images for numbers 1(two fonts),2 and 8.
*) Logs for this have been committed to testlogs/

Following are Maxpooling Neurons

Final Layer of Inference from Max Pooling Layer - BackPropagation on Max Pooling Layer Neurons
Inference from Max Pooling Layer - Image: [0.1452950826564989, 0.15731736449115244, 0.16984767785665972]
Inference from Max Pooling Layer - Image: [0.14920888863234688, 0.16193600589418844, 0.17528092024601222]
Inference from Max Pooling Layer - Image: [0.13197897244871004,

```
0.1417448800619895, 0.15171424525284838]
Inference from Max Pooling Layer - Image: [0.14007345008310818,
0.15118749472569865, 0.16267822705412283]
Inference from Max Pooling Layer - Image: [0.07847441088145349,
0.0807616625716103, 0.08303607840796547]
Inference from Max Pooling Layer - Image: [0.08983732685398557,
0.09365789540391084, 0.09744765711798722]
Inference from Max Pooling Layer - Image: [0.0903328782140764,
0.09421672093823663, 0.09806798136073994]
Inference from Max Pooling Layer - Image: [0.08761036574911765,
0.09113121170496671, 0.09462405037074482]
('Inference from Max Pooling Layer - Example 11:', [0.1619896774094416,
0.1615169723530323, 0.15009965578039716])
('Inference from Max Pooling Layer - Example 12:', [0.14303879829783814,
0.12068614560924568, 0.07211705852669059])
('Inference from Max Pooling Layer - Example 21:', [0.1932801269861012,
0.11587916253160746, -0.007889615456070018])
('Inference from Max Pooling Layer - Example 22:', [0.17605214289097237,
0.19326380725531284, 0.2114152778184319])
('Inference from Max Pooling Layer - Example 3:', [0.062097695970929206,
0.062097695970929206, 0.062097695970929206])
('Inference from Max Pooling Layer - Example 41:', [0.2976324086103368,
0.32891470561393965, 0.34986397693929844])
('Inference from Max Pooling Layer - Example 42:', [0.22669904395788065,
0.26336097548290116, 0.3065443950445386])
('Inference from Max Pooling Layer - Example 51:', [0.11935155170571034,
0.12477637373255022, 0.12911872514904305])
('Inference from Max Pooling Layer - Example 52:', [0.12675869649313706,
0.12051203261825084, 0.10578392639393523])
----------------------------------------------------------
Previous logs are grep-ed from testlogs/. Handwritten Numeral recognitions (with
high noise) are done in following Maxpooling inferences:
[Number 1 - Font 1] Inference from Max Pooling Layer - Image:
[0.07847441088145349, 0.0807616625716103, 0.08303607840796547]
[Number 1 - Font 2] Inference from Max Pooling Layer - Image:
[0.08983732685398557, 0.09365789540391084, 0.09744765711798722]
[Number 2] Inference from Max Pooling Layer - Image: [0.0903328782140764,
0.09421672093823663, 0.09806798136073994]
[Number 8] Inference from Max Pooling Layer - Image: [0.08761036574911765,
0.09113121170496671, 0.09462405037074482]

Contrasting this with bitmap numerals inscribed in 2 dimensional arrays (with
low noise), similar elements have close enough values:
[Number 0 - Font 1] ('Inference from Max Pooling Layer - Example 11:',
[0.1619896774094416, 0.1615169723530323, 0.15009965578039716])
[Number 0 - Font 2] ('Inference from Max Pooling Layer - Example 12:',
[0.14303879829783814, 0.12068614560924568, 0.07211705852669059])
[Number 8 - Font 1] ('Inference from Max Pooling Layer - Example 21:',
[0.1932801269861012, 0.11587916253160746, -0.007889615456070018])
[Number 8 - Font 2] ('Inference from Max Pooling Layer - Example 22:',
[0.17605214289097237, 0.19326380725531284, 0.2114152778184319])
[Null pattern] ('Inference from Max Pooling Layer - Example 3:',
[0.062097695970929206, 0.062097695970929206, 0.062097695970929206])
[Letter X - Font 1] [('Inference from Max Pooling Layer - Example 41:',
[0.2976324086103368, 0.32891470561393965, 0.34986397693929844])
[Letter X - Font 2] ('Inference from Max Pooling Layer - Example 42:',
[0.22669904395788065, 0.26336097548290116, 0.3065443950445386])
[Number 1 - Font 1] ('Inference from Max Pooling Layer - Example 51:',
[0.11935155170571034, 0.12477637373255022, 0.12911872514904305])
[Number 1 - Font 2] ('Inference from Max Pooling Layer - Example 52:',
[0.12675869649313706, 0.12051203261825084, 0.10578392639393523])

----------------------------------------------------------------------------
-----------------------------------
```

394. (THEORY) Coloring Real Numbers and Complement Functions - 8 March 2017 - related to 323, 338

So far equivalence of 2-coloring schemes and integer valued complement functions have been mentioned. Most generic problem in this class
is to find coloring schemes (or) functions which "complement" real and complex n-dimensional surfaces(i.e sets of values of a function f and
its complement g create a disjoint set cover of the real/complex planes). Ramsey theory pertains to coloring integer sequences. 394.1 defines
the problem of coloring the real line as minimum number of colors required to color the real line such that no two points within some specific distance (which in turn is an element of a distance set) are of same color. In Graph theory terms, this reduces to chromatic number of a
graph where two vertices a and b are adjacent iff distance between a and b on real line is in distance set. This is stricter requirement
of coloring compared to complementation and complement graphs representation mentioned in 338. For example, following complementation of [1,2]:

```
      f(0) = [1.0,1.11]
            g(0) = [1.12,1.23]
      f(1) = [1.24,1.30]
            g(1) = [1.31,1.68]
      f(2) = [1.69,2.0]
```

has colored regions of length [0.11,0.11,0.06,0.37,0.31] and can be 2-colored with f and g. If the distance set is {0.01,0.02} two
points 1.32 and 1.33 are 0.01 apart but have same color (belong to same function g). Similarly points 1.69 and 1.71 are 0.02 apart but
have same color f. Thus complementation over reals relaxes the coloring conditions significantly. Coloring real line is a special case of complementation with distance set requirement while Coloring integers is equivalent to complementation. Rather, distance set for real complementation is of size 1 containing length of largest monochromatic streak - in previous example it is {0.37}.

References:

394.1 Coloring Real Line - [EggletonErdosSkilton] - http://ac.els-cdn.com/0095895685900395/1-s2.0-0095895685900395-main.pdf?_tid=3b53ab92-03ec-11e7-870e-00000aab0f01&acdnat=1488970060_784d7a37e58d4662a8e474c62a6c8ae7
394.2 Coloring Reals - http://webbuild.knu.ac.kr/~trj/HN.pdf

---

395. (FEATURE-DONE) Deep Learning Recurrent Neural Network Gated Recurrent Unit (RNN GRU) Implementation - 9 March 2017

---

*) This commit implements Gated Recurrent Unit algorithm (most recent advance in deep learning - published in 2014) which is a
simplification of RNN LSTM by reducing number of gates (input, cell, forget, output are mapped to cell, reset, update)
*) Logs for this have been uploaded to testlogs/ showing convergence of gates and state.

---

396. (FEATURE-DONE) Recursive Lambda Function Growth + Tensor Neuron (NTN) Intrinsic Merit - 13 March 2017 - related to 384, 385

---

*) This commit rewrites the Recursive Lambda Function Growth implementation by adding a new lambda function growth
function.

*) This new function creates a definition graph with Recursive Gloss Overlap algorithm from a text and computes all pairs shortest paths in the graph
*) For each such shortest path edges, an AVL lambda function composition balanced tree is grown (i.e every path is a lambda function composition tree).
*) Tensor Neuron Lambda Function is computed for each subtree root by following relation evaluated on a stack:

function(operand1, operand2)

where operand1, operand2 and function are successively popped from AVL composition tree stack representation
*) Tensor Neuron triplet (operand1, function, operand2) is evaluated presently as maximum Wu-Palmer similarity of Synsets for cartesian product of operand1 and operand2. But it can be augmented with any arbitrary weighting scheme. Cartesian product of Synsets for 2 entities is the simplest 2-dimensional tensor neuron.
*) Sum of tensor neuron weights for all random walks composition tree evaluation is printed as merit of the text.
*) Logs for this have been committed to testlogs

---

397. (FEATURE-DONE) Korner Entropy Intrinsic Merit for Recursive Gloss Overlap graphs - 14 March 2017

---

*) Implemented Korner Entropy intrinsic merit for Recursive Gloss Overlap graph.
*) function korner_entropy() finds maximal independent sets for all vertices and chooses the minimum entropy such that a vertex is in a maximal independent set.
*) Logs for this have been committed to testlogs/

Previous two intrinsic merit measures - tensor neuron network and korner entropy - sufficiently quantify the complexity of a document's inner purport. Neuron Tensors transform the ontology subgraph of a text into a new kind of multilayered graph perceptron. Infact it generalizes traditional multilayered trellis perceptrons for which backpropagation is applied to find weights per activation edge. There is no known backpropagation algorithm for generic graph of perceptrons. Neuron Tensors per word-word edge are equated to a lambda function and compositional tree of these neurons for any walk on the text-graph is an alternative deep learning technique to backpropagation. Korner Entropy is a measure of disconnectedness (total probability of vertices in independent sets). Presently all pairs shortest paths is used in place of random walks - set of all pairwise path tensor neuron lambda compositions is an upperbound on set of all random walks.

---

398. (FEATURE-DONE) Software Analytics update - 15 March 2017

---

*) Software Analytics Deep Learning code has been updated to include all Deep Learning implementations: RNN LSTM, RNN GRU, BackPropagation and ConvolutionNetwork+BackPropagation
*) logs for this have been committed to software_analytics/testlogs/

---

399. (USECASE) NeuronRain Usecases - Software (Log) Analytics and VIRGO kernel_analytics config - 16 March 2017

Software Analytics Deep Learning implementation in python-src/software_analytics/ sources its input variables - CPU%, Memory%, TimeDuration% for a process from logs. VIRGO kernel_analytics module reads machine learnt config variables from kernel_analytics.conf and exports them kernelwide. An example single layer perceptron usecase is:

```
      If the CPU% is > 75% and Memory% > 75% and TimeDuration% > 75% then
            do a kernelwide   overload signal
      else
            do nothing
```

Sigmoid function for this usecase is:
sigma(4/9 * x1 + 4/9 * x2 + 4/9 * x3) where x1 = CPU%, x2 = Memory% and x3 = TimeDuration%
which is 1 when all variables reach 75%.

Spark Streaming Log Analyzer ETLs htop data to .parquet files and Software Analytics code reads it, deep-learns above neural network and writes a key-value variable OverloadSignal=1 to VIRGO kernel_analytics.conf. VIRGO kernel_analytics driver reads it and exports a kernelwide variable kernel_overload=1. Any interested kernel driver takes suitable action based on it (automatic shutdown, reboot, quiesce intensive applications etc.,)

An inverse of this usecase is to find the weights for x1,x2,x3 when system crashes. This requires a gradient descent or backpropagation with multilayered perceptron.

---

400. (THEORY) Mechanism Design, Condorcet Jury Theorem, Derandomization, Gibbard-Satterthwaite Theorem and Designing a Voting Mechanism
- 27 March 2017 , 30 March 2017, 31 March 2017, 18 April 2017 - related to 368 and all other P(Good) circuit related sections

---

Gibbard-Satterthwaite theorem prevents any Social Choice Function defined on atleast 3 elements from being non-dictatorial. Mechanism Design is the reverse game theory which tries to circumvent this limitation and designs a mechanism for required result for social goodness. With respect to P(Good) series and circuit, design of a voting mechanism for arbitrary number of voters which overcomes limitation of Gibbard-Satterthwaite Theorem would imply convergence of P(Good) series. This is related to exact learning of boolean social choice functions with zero-error in 368.6. Open question: Does exact learning of a voter decision function imply a voting mechanism design?

A simple example of strategic/tactical voting and Gibbard-Satterthwaite theorem is given in http://rangevoting.org/IncentToExagg.html. Here by changing the ranking preferences (i.e voter switch loyalties by manipulation, tactical and strategic voting, bribery, bounded rationality and erroneous decision etc.,) result of an election with atleast 3 candidates can be changed. Thus majority voting is vulnerable to error. Gibbard-Satterthwaite Theorem applies to all voting systems including secret ballot because even discreet voter can be psychologically influenced to change stance against inherent desire.

This implies for more than 3 candidates, RHS of P(Good) series for majority voting can never converge because there is always a non-zero error either by bounded rationality or manipulation. By pigeonhole principle either some or all of voters contributing to the P(Good) summation must have decided incorrectly or manipulated with non-zero probability. But this requires voter decision function (i.e social choice function) to be non-boolean (Candidates are indexed as 0,1,2,...). Exact learning of boolean functions applies to only 2 candidates setting where candidates are indexed as 0 and 1. Also, LHS of P(Good) series which is a (or quite close to a dictatorship by Friedgut-Kalai-Nisan quantitative version of Gibbard-Satterthwaite theorem) dictatorship social choice function, would be strategyproof (resilient to error and manipulative voting) and thus may have greater goodness probability (which is a conditional probability on chosen one's goodness) than RHS in more than 3 candidates setting.

Condorcet Jury Theorem mentioned in references below, is exactly the P(Good) binomial series summation and has been studied in political science and social choice theory. This theorem implies majority decision is better and its goodness probability tends to 1 than individual decision if probability of correct decision making for each voter is > 0.5. In the opposite case majority decision goodness worsens if individual decision goodness probability is < 0.5. Thus majority voting is both good and bad depending on individual voter decision correctness. In terms of BP* complexity definitions, if each voter has a BP* social choice function (boolean or non-boolean) with p > 0.5, accuracy of majority voting tends to 100% correctness i.e Composition Majn(BPX1, BPX2,...) for some complexity class BPXi for each voter implying derandomization (this also implies BPX=X if Majn(BPX1,BPX2,...) composition is in BPX). Proof of BPP ! = P would imply there is atleast one voter with p < 0.5. Condorcet theorem ignores dependency and unequal error scenarios but still holds good by central limit theorem (because each voter decision is a random variable +1 or -1 and sum of these random variables for all voters tends to gaussian by CLT) and law of large numbers.

Corollary: From 129 and 368, depth of Majority voting circuit composition can be unbounded and if all voter social choice functions have same number of variables and number of voters is exponential in number of variables, Majority voting circuit composition with error is in BPEXP than PH. By Condorcet Jury Theorem, if all voters tending to infinity decide correctly with probability > 0.5, then BPEXP derandomizes to EXP => BPEXP=EXP. Non-boolean social choice functions for more than 2 candidates are usually preference profiles per voter (list of ranked candidates per voter) and topranked candidate is voted for by the voter.

If Majority Voting Circuit is in BPP, which is quite possible if number of voters is polynomial in number of variables and circuit is of polynomial size and because from Adleman's Theorem (and Bennet-Gill Theorem), BPP is in P/poly, BPP derandomizes to P for infinite electorate with decision correctness probability > 0.5 for all voters. Thus homogeneous electorate of high decision correctness (> 0.5) implies both BPEXP=EXP and P=BPP.

Locality Sensitive Hashing accumulates similar voters in a bucket chain. The social choice function is the distance function. Circuit/Algorithm for Multiway contest social choice (generalization of Majority boolean function) is:

*) Locality Sensitive Hashing (LSH) for clustering similar voters who vote for same candidate

*) Sorting the LSH and find the maximum size cluster (bucket chain)

References:

400.1 Mechanism Design, Convicting Innocent, Arrow and GibbardSatterthwaite Theorems - https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-254-game-theory-with-engineering-applications-spring-2010/lecture-notes/MIT6_254S10_lec21.pdf

400.2 Electoral Fraud, Gibbard-Satterthwaite Theorem and its recent extensions on manipulative and strategic voting, Cake cutting Multi Agent Resource Allocation Protocols - https://www.illc.uva.nl/COMSOC/theses/phd-schend.pdf

400.3 Condorcet Jury Theorem in Political Science - http://www.stat.berkeley.edu/~mossel/teach/SocialChoiceNetworks10/ScribeAug31.pdf - Asymptotic closed form and convergence of P(Good) binomial summation when probability of good decision > 0.5 uniformly for all voters was already solved by Condorcet 2 centuries ago :

"... Condorcet’s Jury theorem applies to the following hypothetical situation: suppose that there is some decision to be made between two alternatives + or −. Assume that one of the two decisions is ‘correct,’ but we do not know which. Further, suppose there are n individuals in a population, and the population as a whole needs to come to a decision. One reasonable method is a majority vote. So, each individual has a vote Xi , taking the value either +1 or −1 in accordance with his or her opinion, and then the group decision is either + or − depending on whether Sn = Pn i=1 Xi is positive or negative. Theorem 1.1. (Condorcet’s Theorem) [4] If the individual votes Xi , i = 1, . . .

, n are independent of one another, and each voter makes the correct decision with probability p > 1 2 , then as n → ∞, the probability of the group coming to a correct decision by majority vote tends to 1. ..."
400.4 Strategic Voting and Mechanism Design - [VinayakTripathi] - https://www.princeton.edu/~smorris/pdfs/PhD/Tripathi.pdf
400.5 Condorcet Jury Theorem and OCR - Section 2 - http://math.unipa.it/~grim/Jlamlouisa.PDF
400.6 Condorcet Jury Theorem graph plots - http://www.statisticalconsultants.co.nz/blog/condorcets-jury-theorem.html
400.7 Probabilistic Aspects of Voting - Course Notes - www.maths.bath.ac.uk/~ak257/talks/Uugnaa.pdf - Condorcet Jury Theorem, Bertrand Ballots Theorem and Gibbard Random Dictatorship Theorem
400.8 Law of Large Numbers and Weighted Majority - [Olle Haggstrom, Gil Kalai, Elchanan Mossel] - https://arxiv.org/abs/math/0406509v1
400.9 Thirteen Theorems in search of truth - http://www.socsci.uci.edu/~bgrofman/69%20Grofman-Owen-Feld-13%20theorems%20in%20search%20of%20truth.pdf - Condorcet Jury Theorem for Heterogeneous voters (voters with unequal decision correctness probability - Poisson-binomial distribution) - same as Condorcet Jury Theorem for Homogeneous voters (voters with equal decision correctness probability) when correctness probabilities are a Gaussian and > 0.5. In other words, large majority is more correct than small majority.
400.10 Margulis-Russo Formula and Condorcet Jury Theorem - http://www.cs.tau.ac.il/~amnon/Classes/2016-PRG/Analysis-Of-Boolean-Functions.pdf - pages 224-225 - Majority function is defined in Erdos-Renyi Random graph model with each edge having probability p. Maj(N,G) for a random graph G with probability of edge being p = True of G has atleast N/2 = |V|*(|V| - 1)/4 edges. Majority voting can be defined as a random graph with each voter being an edge with decision correctness probability p i.e Probability of Existence of an edge increases with correctness of decision by voter. Margulis-Russo formula describes the sharp threshold phenomenon for boolean functions and rate of change of P(Maj(G))=1 with respect to p, specifically for majority boolean function. Thus it is an alternative spectacle to view P(Good) majority voting binomial coefficient summation and Condorcet Jury Theorem. For majority function, sharp threshold occurs at p >= 0.5.
400.11 Percolation, Sharp Threshold in Majority, Condorcet Jury Theorem and later results, Influence, Pivotality - http://www.cs.tau.ac.il/~safra/PapersAndTalks/muligil.old.pdf
400.12 Complexity Inclusion Graph - https://www.math.ucdavis.edu/~greg/zoology/diagram.xml - BPEXP=EXP implies a significant collapse of the complexity class separation edges.

---
401. (FEATURE-DONE) LSH Index implementation for NeuronRain AsFer Text and Crawled Web Documents Intrinsic Merit Ranking - 4 April 2017
---
*) This commit adds a Locality Sensitive Hashing based inverted index for scrapy crawled HTML website pages.
*) It is a derivative of LSH based similarity clustering implemented in python-src/LocalitySensitiveHashing.py
*) It has to be mentioned here that ThoughtNet is also an inverted index for documents with additional features (i.e it is
classifier based than raw string similarity)
*) LSHIndex uses Redis Distributed Key Value Persistent Store as hashtable backend to store index.
*) LSHIndex is not string to text inverted index, but hashed string to text inverted index (something like a digital fingerprint)
*) Presently two types of hashes exist: primitive ordinal and MD5 hash
*) Only one hash table is implemented on Redis with multiple hash functions with random choice.
*) Presently 50 hash functions with 50 entries hash index has been implemented.

---
402. (FEATURE-DONE) ThoughNet Index implementation for NeuronRain AsFer Text and Crawled Web Documents Intrinsic Merit Ranking - 4 April 2017
---
*) This commit implements an inverted index on top of ThoughtNet
*) ThoughtNet is a hypergraph stored in file and on Neo4j Graph Database
*) JSON loads ThoughtNet edges and hypergraph files and queries by classes returned by Recursive Gloss Overlap classifier
*) Querying Neo4j is also better, but file representation of ThoughtNet is quite succinct and scalable because it just numerically
encodes each edge (= a web crawled HTML document)
*) file representation of ThoughtNet views the hypergraph as stack vertices interconnected by document id(s) similar to a source code
versioning system. Having similar view on a graph database doesn't look straightforward

---
403. (FEATURE-DONE) Initial commits for Indexless Hyperball Recursive Web Crawler - 7 April 2017
---
*) An indexless crawler which tries to find a url matching a query string in a recursive crawl creating a hyperball of certain radius
*) This is similar to Stanley Milgram, Kleinberg Lattice Small World Experiments and recent Hyperball algorithm for Facebook graph which
concluded that Facebook has 3.74 degrees of freedom.
*) Starts with a random url on world wide web.
*) Success probability depends on average number of hops over large number of queries.
*) This is a Randomized Monte Carlo Crawler and does a Depth First Search from an epicentre pivot url to start with
*) Hyperball References:
	1) http://webgraph.di.unimi.it/docs/it/unimi/dsi/webgraph/algo/HyperBall.html
	2) Facebook Hyperball - https://arxiv.org/pdf/1308.2144.pdf

---
404. (THEORY) Indexless Web Search, World Wide Web Graph Property Testing, On-demand Indexing - related to 403 - 9 April 2017

Almost all known web search engines rely on a pre-built index created from already crawled WWW graph. Is a pre-built index necessary for
web search? Can the small world property of world wide web graph (i.e property of graph such that easy short paths exist between any pairs
of vertices as proved in Stanley Milgram experiment, Kleinberg Lattice), be applied to find a url matching a search query starting from any
vertex? Facebook graph has almost a billion user vertices and on an average any person is connected to any other within 3.74 path lengths.
Can this be generalized to world wide web? Traditional book index maps word queries to pages having the words. If entire book is translated
into a huge definition graph by Recursive Gloss Overlap graph, then instead of looking-up index, starting at a random word vertex, any other
word should be easily reachable if graph has small world property. This realworld problem is already studied as Graph Property Testing which are algorithms to test if a graph has specific property(clique, colorability, edge and vertex queries etc.,). Hyperball indexless crawler in NeuronRain AsFer is an effort in this line of thought. Such an indexless crawler can "learn" an index online by repeated queries and build an index from cached results at runtime. Index is not prebuilt but grown ondemand. Assuming internet has such small world

degrees of freedom upperbounded by a constant, any query is constant time serviceable in average case complexity.

References:

404.1 Small World Property - [WattsStrogatz]
-http://www.nature.com/nature/journal/v393/n6684/full/393440a0.html

---
405. (FEATURE-DONE) Commit - NeuronRain AsFer-CPython Extensions VIRGO64 system call invocations - 20 April 2017
---
*) VIRGO64 system calls are invoked from Python code by CPython extensions.
*) Separate folder cpython_extensions64/ has been created for 64bit VIRGO kernel
*) Requires Python 64-bit version
*) Logs for VIRGO64 memory and filesystem systemcalls-to-drivers have been committed to testlogs
*) With this Complete Application Layer-SystemCalls-Drivers request routing for python applications deployed on VIRGO64 works.

---
406. (FEATURE-DONE) Commits - NeuronRain AsFer Boost C++-Python - VIRGO64 System calls + Drivers invocation - 26 April 2017
---
*) VIRGO64 system calls are invoked from C++ by Boost::Python extensions
*) Kernel Logs for VIRGO KMemCache, Clone and FileSystem Calls Boost::python invocations have been committed to testlogs/
*) Boost version used is 1.64.0
*) setup.py has been updated with library_dirs config variable
*) virgofstest.txt for filesystem calls has been committed to testlogs/

---
407. (FEATURE-DONE) Commits - 28 April 2017 - NeuronRain AsFer-Boost::Python-C++ - VIRGO64 systemcalls and drivers invocations
---
*) VIRGO64 clone, kmemcache and filesystem system calls were invoked from Python-C++ boost extensions again and reproducibly work
without kernel panics.
*) Logs, persisted disk file written by filesystem system calls and rebuilt boost-python C++ extensions have been committed to testlogs/

---
408. (THEORY) Condorcet Jury Theorem, Collaborative Filtering, Epistemological Democracy, Network Voting in WWW Link Graph,
Majority Function and Correctness of Majority Vote Ranking - 16 May 2017, 23 May 2017

Let v1, v2 be two vertices being good and bad choices respectively. In link graph all incoming edges to a vertex v1 are
votes for that vertex. From Condorcet Jury Theorem if all incoming links occur with probability p > 0.5, then probability that v1 is chosen tends to 1 for large indegree. For 2 candidates c1, c2 and voters v1, v2, v3, ... vn, edges (votes) to c1 occur with probability p and edges(votes) to v2 occur with probability 1-p. Thus voting graph is a random bipartite graph and can be generalized to multiple candidates. Candidates Vertex set C={c1, c2, ...,cn} can have more than one indegree and zero outdegree. Voters Vertex set V={v1,

v2, ..., vn} can have only one outdegree and zero indegree. If C intersection V != emptyset, then graph is general directed graph and can have cycles. Each voter vertex vy either votes to some other vertex vx or receives vote from vx. vy votes for vx with probability p. vy votes for vertices other than vx with cumulative probability 1-p. Vertices voted by vy are ranked by goodness (i.e goodness(vx) > goodness(others)). If vy votes for vx then vy decides with decision correctness probability p.

More generically, vertex vx receives votes from vertices vy2,vy2,vy3,...,vyn with decision correctness probability p. Vertices vy1,vy2,vy3,...,vyn vote for vertices other than vx with decision correctness probability 1-p. Each vertex in WWW link graph has this 2 level tree structure with varying probability pi. From Condorcet Jury Theorem, hyperlinks to vx from vy1,vy2,...,vyn are 100% correct group decision if p > 0.5. Equal p for all votes to a vertex is the homogeneous voter assumption of Condorcet Jury Theorem. If the vertices are ranked by some algorithm (HITS Hub-Authority, PageRank etc.,) then correctness of rank is measured by Condorcet Jury Theorem. It is apt to mention here that probability p is different from weights of directed graph edges in PageRank Markov Random Walk iteration (where weight of outgoing edge/vote is decided by the ratio 1/outdegree) in the sense that: 1/outdegree is the static percentage of vote to an adjacent vertex in a non-random graph while p in network voting is a dynamic measure for decision correctness of a vote to a vertex in a random graph. For heterogeneous voter assumption, more recent theorems which generalize Condorcet Jury Theorem are required.

Collaborative Filtering in Recommender Systems is the most generalized way of Majority Voting where a matrix of users to their preferences for items is used as a training data for a new user to recommend an item. For example, if 9 out of 10 users up vote an item, 11th user is recommended that item and viceversa for down votes. Above analysis is an alternative direction for defining value judgement correctness already discussed in references 408.1 and 408.2. Correlated Votes and their effects on Condorcet Jury Theorem are analyzed by [KrishnaLadha].

References:

408.1 Condorcet Jury Theorem and the truth on the web - http://voxpublica.no/2017/03/condorcets-jury-theorem-and-the-truth-on-the-web
408.2 Webometrics, Goodness of PageRank and Condorcet Jury Theorem - [MastertonOlssonAngere] - https://link.springer.com/article/10.1007/s11192-016-1837-1 - PageRank is very good in finding truth i.e it is close to objective judgement with certain empirical assumptions despite being subjective ranking measure. Here objective judgement is any value judgement which is not majority voting or subjective. Computational linguistic text graph analysis which reckons the psychological aspects of text comprehension is an objective intrinsic merit judgement. Ideally majority vote should coincide with objective judgement absence of which implies error in voting or objective judgement or both (family of BP* complexity classes).
408.3 Generalized Condorcet Jury Theorem, Free Speech, Correlated Voting (dependence of voters) - https://www.jstor.org/stable/2111584

---
409. (FEATURE-DONE) Commits - 17 May 2017 - NeuronRain AsFer-VIRGO64 Boost-C++-Python invocations
---
*) Boost Python C++ VIRGO64 system calls were tested repeatedly in a loop
*) All three system call subsystems - virgo_clone, virgo_kmemcache, virgo_filesystem - work well without any kernel panics
*) But a strange thing was observed: virgo filesystem system calls were not appending to disk file but when run as
"strace -f python asferpythonextensions.py" disk file is written to (a coincidence or strace doing something special)

---
410. (THEORY) Objective and Subjective Value Judgement, Text Ranking and Condorcet Jury Theorem - 29 May 2017 and 4 June 2017 - Related to 202, 384, 385 and all Recursive Lambda Function Growth and Recursive Gloss Overlap algorithms sections in this document
---
Objective merit or Intrinsic merit of a document is the measure of meaningfulness or information contained in a document. Subjective merit is how the document is perceived - Reality Versus Perception - Both should ideally coincide but do not.

Following are few real-world examples in addition to academic credentials example in 202:
*) Soccer player, Cricket player or a Tennis player is measured intrinsically by number of goals scored, number of runs/wickets or number of grandslams won respectively and not subjectively by extent of votes or fan following to them (incoming edges). Here reality and perception coincide often and an intrinsically best player by records is also most revered. Any deviation is because of human prejudice. Here intrinsic merit precedes social prestige.
*) Merits of students are judged by examinations (question-answering) and not by majority voting by faculty. Thus question-answering or interview is an algorithm to measure intrinsic merit objectively. Here again best student in terms of marks or grades is also the most favoured. Any deviation is human prejudice. Interview of a document is how relevant it is to a query measured by graph edit distance between recursive gloss overlap graphs of query and text. Here also intrinsic merit precedes social prestige.

Present ranking algorithms are mostly majority voting (perception) oriented which is just half truth. Other half is the reality and there are no known algorithms to objectively judge documents. Recursive Lambda Function Growth algorithm and its specialization Recursive Gloss Overlap graph are objective intrinsic merit algorithms filling this void and try to map above real-life examples to text document ranking.

But how to measure "number of goals etc.," of a document vis-a-vis the rest? This is where Computational Linguistic Text-to-Graph analysis fits in. Graph representation of a document with various quantitative and qualitative graph complexity measures differentiates and grades the texts by intrinsic complexity. This combines two fields: Computational Linguistics which is founded on Psychoanalysis and Graph Theory. Former pertains to mental picture created by reading a document and latter is how complex that picture is. Recursive Lambda Function Growth envisages an ImageNet which is a graph of related images of entities to create an animated movie representation of a text stored as a subgraph of ImageNet. ImageNet (pictorial WordNet) is not yet available.

Condorcet Jury Theorem formalizes above intuition and bridges two worlds - objective reality and subjective perception. A best performing student by objective examination/interview assessment is also the most voted by infinite faculty if correctness of decision > 0.5 for all faculty voters i.e. Objective Intrinsic Merit = Subjective Perception based Majority Voting Merit.

Crucial Observation is: Merit creates Centrality (Social prestige) and not the opposite - Prestige can not create merit but can only be a measure of merit. Otherwise this is an anachronism - merit does not exist yet but prestige exists and thus prestige preceding merit.

Alternatively, interview of document is simply the rank of the document in terms of intrinsic graph complexity merit of it versus the rest of the text documents. Thus there are 2 aspects of interview:
*) Ranking text document by intrinsic graph complexity/entropy merit
*) Relevance of the text to a query - graph edit distance

Here again another question can be raised: Why is a graph representation of text an apt objective intrinsic merit measure? Because, by "Circuits of Mind" and "Mind grows circuits/lambda functions" theories (cited in previous sections), biological neurons are connected as a graph and information from sensory perception is transmitted through these neurons. Threshold TC circuits theoretically formalize neural networks. Better neuroimaging techniques for quantifying the potentials created in brain by cerebral representation of a text could be ideal relevance measures and can be substituted in Tensor Neuron Model of Recursive Lambda Function Growth algorithm.

There is special case of intrinsic merit: For example, reading a story creates a mental picture of it as a graph of events. Objectively judging it could vary from one human to the other making it subjective. But still the absolute objective merit can be defined for such a special case by applying EventNet to this problem - whole set of events are laid out as events with partakers with cause-effect edges amongst them.

References:

410.1 The Meaning of Meaning - http://courses.media.mit.edu/2004spring/mas966/Ogden%20Richards%201923.pdf
410.2 WordNet and Word2Vec - https://yaledatascience.github.io/2017/03/17/nnnlp.html - In word2vec words from text are represented in a vector space and contextually related words are close enough in proximity on vector space. Word2Vec is a recent neural network model of word relations similar to Neural Tensor Network (NTN). NTN defines a relation between two words as a tensor neuron and thus complete WordNet can be defined as a graph with Tensor Neuron as edge potentials. This is the motivation for invoking Neural Tensor WordNet as an objective intrinsic merit indicator as it approximates human brain neurological text comprehension and visualization.
410.3 Random Walks on WordNet - http://anthology.aclweb.org/N/N15/N15-1165.pdf - Recursive Lambda Function Growth algorithm does something similar to this. It finds all random walks on Recursive Gloss Overlap graph (described in 385) and constructs a lambda function composition tree for each such walk and assigns a Neural Tensor Potential to each edge of these trees.
410.4 ImageNet - Pictorial WordNet - http://www.image-net.org/ - image net is recently available. In the context of Recursive Lambda Function Growth, lambda functions inferred from ImageNet than WordNet/ConceptNet are pictures and composition of the lambda functions translates to animating the pictures based on PoS or tensor relation. For example the composition for "It rained heavily today", rained(It, heavily(today)), should theoretically compose/superpose/animate images for day, heaviness, rain from leaves to root of the tree.

---
411. (FEATURE-DONE) AngularJS - Tornado GUI-REST WebService - Commits - 1 June 2017

NeuronRain AngularJS RESTful client for Tornado webserver and others
---
---
*) AngularJS support has been added to NeuronRain GUI-WebServer with a new webserver_rest_ui/NeuronRain_AngularJS_REST_WebServer.py which reads and renders angularjs Model-View-Controller templates
*) New AngularJS Model, View and Controller script-html templates have been added to angularjs directory
*) LSHIndex.py has been updated to have commandline arguments for index queries
*) Hyperball Crawler pivot epicentre url has been changed and an example query "Chennai" was found to be matching within few hops.

*) NeuronRain_REST_WebServer.py has been updated to print the logs for the script execution to browser console with subprocess POpen() , communicate() and poll() python facilities

---
412. (THEORY) Graph Neural Networks, Tensor Neurons and Recursive Lambda Function Growth Intrinsic Merit - 4 June 2017 and
8 June 2017 - related to 410

Definition Recursive Gloss Overlap Graph with word-word edges as Neuron Tensor Network relations having potentials was defined previously. Recursive Lambda Function Growth computes all random walks of the definition graph and creates a lambda function composition tree for each such random walk. For each subtree f of the recursive lambda function growth with children w1 and w2 composed potential for f is defined as:

p(f(w1,w2)) = p(w1) * p(w2) for some operator *

Rather than mere summation of potentials of edges following minmax criterion extracts the most meaningful random walk:
*) Minimum Neuron Tensor Potential edge of each random walk is found - path minimum potential
*) Maximum of all minimum path potentials extracts a random walk lambda composition tree which is the most probable inferred meaning of the document - Max(Min(potentials)) - this makes sense because maximum of minimum potential implies minimum possible neural activation.

Event Related Potentials (ERP) mentioned in 202 are unusual spikes in EEG of brain (N400 dataset) when unrelated words are read. Complement of ERP for related words as dataset could be an ideal estimator for Neuron Tensor relatedness potential between two words in definition graph.

Graph Neural Networks are recent models of neural network which generalize to a graph. In a Graph Neural Network, potential of each vertex is a function of potentials of all adjacent vertices and neuron tensor potentials of incoming edges from them. Neuron Tensor Network edge relations for each word vertices pair of Recursive Gloss Overlap Graph can be mapped to edges of a Graph Neural Network. Thus a text is mapped to not just a graph but to a Graph Neural Network with Tensor Neuron Edges, an ideal choice for human text comprehension simulation. Recursive Gloss Overlap Graph with Neuron Tensor word-word edges combines two concepts into one - Graph Tensor Neuron Network (GTNN).

Data on websites can be classified into 3 categories: 1) text 2) voice 3) images and videos. Intrinsic merit which is a measure of creativity presently focuses only on text. Analyzing the intrinsic merit of voice and visuals is a separate field in itself - Fourier Analysis of Waveforms and Discrete Fourier Transforms.

Ranking text by Graph Tensor Neuron Network intrinsic merit potential can be done in multiple ways:
- Rank Vertices and Edges by potential
- Rank the random walk lambda function composition tree by potential
- Rank by Korner Entropy of the Graph Tensor Neuron Network and other qualitative and quantitative graph complexity metrics.
and so on.

Example lambda composition tree on a random walk of recursive gloss overlap graph:

```
f1 = requiredby(fuel, flight)
f2 = has(flight, tyre)
f3 = has(tyre, wheel)
f4 = does(wheel, landing)
f5 = requires(landing, gear)
f1(fuel, f2(flight, f3(tyre, f4(wheel, f5(landing, gear)))))
```

Previous example random walk lambda composition tree has been constructed in 384 and 385. Potentials for Tensor Neuron Relations have to be predetermined by a dataset if such exists similar to N400 EEG dataset. Tensor Neuron Relations in Recursive Gloss Overlap Definition Graph are grammatical connectives mostly. In this example, "requiredby" is a relation. Cumulative potential of each such random walk lambda composition tree has to be computed.

Difference in Graph Tensor Neuron Network mapping of a text is: Each lambda composition tree is evaluated as a Graph Neural Network - each subtree is evaluated and passed on to higher level - and potential at the root is returned as the merit. Mixing time in markov chain andom walk is the number of steps before stationary distribution is attained. Thus maximum potential returned at the root of (a lambda composition graph tensor neuron network of (any converging stationary random walk of (a recursive gloss overlap definition graph of (a text)))) is a measure of intrinsic merit.

Intuition for Graph Tensor Neuron Network is below:
*) Random walk on recursive gloss overlap graph simulates randomness in cognitive cerebral text comprehension.
*) Tensor Neuron relatedness potential between two word vertices in definition graph quantifies relevance and meaningfulness
*) Lambda composition tree-graph neural network for each random walk on definition graph simulates how meaning is recursively understood bottom-up with randomness involved.

References:

412.1 Graph Neural Networks -
http://repository.hkbu.edu.hk/cgi/viewcontent.cgi?article=1000&context=vprd_ja

---
413. (FEATURE-DONE) Recursive Lambda Function Growth - Graph Tensor Neuron Network Implementation - Commits - 9 June 2017

(*) Recursive Lambda Function Growth implementation has been updated to compute Graph Tensor Neuron Network intrinsic merit
(*) Each lambda function composition tree of a random walk on recursive gloss overlap graph is evaluated as a Graph Neural Network having Tensor Neuron potentials for word-word edges.
(*) Tensor neuron potential for relation edges in Graph Neural Network has been hardcoded at present and requires a dataset for grammatical
connective relations similar to EEG dataset for brain spikes in electric potentials.

---
414. (FEATURE-DONE) Graph Tensor Neuron Network - implementation update - Commits - 10 June 2017
---
*) Changed the subtree graph tensor neuron network computation

*) Children of each subtree are the Tensor Neuron inputs to the subtree root Each subtree is evaluated as a graph neural network with weights for each neural input to the subtree root.

*) WordNet similarity is computed between each child and subtree root and is presently assumed as Tensor Neuron
relation potential for the lack of better metric to measure word-word EEG potential.

If a dataset for tensor neuron potential is available, it has to to be looked-up and numeric
potential has to be returned from here.

*) Finally a neuron activation function (simple 1-dimensional tensor) is computed and returned to the subtree root for next level.

*) logs for the this have been committed to testlogs. Presently graph tensor neuron network intrinsic merit is a very small decimal
because of the decimal values of similarity and tensor neurons, but quite receptive to small changes.

---
415. (FEATURE-DONE) Script for Querying Index (LSH and ThoughtNet) and Ranking the results with Recursive Lambda Function Growth
- Commits - 13 June 2017
---
*) New python script QueryIndexAndRank.py has been committed for retrieving results from Index matching a query and rank them
*) It queries both Locality Sensitive Hashing and ThoughtNet indices
*) Functions of classes have been parametrized for invocation across modules
*) Both LSH and ThoughtNet indices have been recreated
*) Ranking is done by Recursive Lambda Function Growth algorithm which is a superset of Recursive Gloss Overlap
*) This script was tested via NeuronRain Tornado RESTful GUI

---
416. (FEATURE-DONE) Graph Tensor Neuron Network Intrinsic Merit and QueryIndexAndRank update - Commits - 15 June 2017
---
*) Graph Tensor Neuron Network Intrinsic Merit computation has been changed - Intrinsic merits of all random walks are summed up than multiplying to get a tangible merit value
*) Some bugs fixed and debug prints have been changed
*) logs have been committed to testlogs/

---
417. (FEATURE-DONE) Hyperball crawler update - Commits - 19 June 2017

Integrated Recursive Lambda Function Growth Intrinsic Merit Score(Graph Tensor Neuron Network and Korner Entropy) into Hyperball crawler.

---
418. (THEORY) Thought Experiment of Generic Intrinsic Merit and Condorcet Jury Theorem - 21 June 2017

Intrinsic merit so far is restricted to text documents. Subjective ranking measures are like mirrors which reflect the real merit of the
candidate. Each vote in the majority voting is similar to an image of the votee projected on to the voter mirror. Some mirrors reflect well
while others don't. Efficiency of a voter mirror is equivalent to decision correctness of a voter. Present web ranking algorithms measure votee by the perception mirror images, which is indirect. Intrinsic merit breaks mirror images and relies only on what the votee really is. But then isn't intrinisic merit a mirror image too? It is not because no human perception or image based voting is involved to ascertain merit. Only the graph complexity of the text-

graph and its graph neural network is sufficient. Merit is a universal requirement. Examples in 410 and 202 motivate it. Can merit be applied to entities beyond voice, visuals and text? Most probably yes. Intelligence of human beings are ranked by intelligence quotients which is also an intrinsic merit measure (though IQ tests are disputed by psychological studies). Hence judging people by intrinsic merit is beyond just IQ scores. An algorithm to classify people as "Good" and "Bad" could be a breakthrough in machine intelligence. Hypothetical people classifier could be as below:
(*) Peruse academic records and translate to score
(*) Peruse work records and translate to score
(*) Translate awards received to score
(*) Translate brain imageing data to score (EEG and fMRI)
(*) Translate IQ or EQ to score
Above is not a formal algorithm but tries to intuit how it may look like. Human Resource Analytics are done as above

Objective Intrinsic Merit = Vote by an algorithm
Subjective Ranking = Votes by people translated into rank by an algorithm
(present ranking algorithms rely on incoming links or votes to a text which are created by human beings)

Thus Generic Intrinsic Merit removes human element completely while assessing merit (closer to AI).Goodness when it applies to human beings is not just a record based score and usually has moral and ethical elements in it. People classifier based on intrinsic merit has to be intrusive and invasive similar to HR analytics example earlier.

(*)Question: Why should intrinsic merit be judged only in this way?
(*)Answer: This is not the only possible objective intrinsic merit judgement. There could be other ways too. Disclaimer is intrinsic merit assumes cerebral representation of sensory reception (words, texts, visuals, voices etc.,) and its complexity to be the closest to ideal judgement.

(*)Question: Wouldn't cerebral representation vary from person to person and thus be subjective?
(*)Answer: Yes, but there are standardized event related potential datasets gathered from multiple neuroscience experiments on human subjects. Such ERP data are similar for most brains. Variation in potential occurs because cerebral cortex and its sulci&gyri vary from person to person. It has been found that cortex and complexity of gray matter determine intelligence and grasping ability. Intrinsic merit should therefore be based on best brain potential data.

(*)Question: Isn't perception based ranking enough? Why is such an intrusive objective merit required?
(*)Answer: Yes and No. Perception majority voting based ranking is accurate only if all voters have decision correctness probability > 0.5 from Condorcet Jury Theorem. PageRank works well in most cases because incoming edges vote mostly with >50% correctness. This correctness is accumulated by a Markov Chain Random Walk recursively - vote from a good vertex to another vertex implies voted vertex is good (Bonacich Power Centrality) and so on. Initial goodness is based on weight of an edge. Markov iteration stabilizes the goodness. Probability that goodness of stationary Markov distribution < 0.5 can be obtained by a tail bound and should be exponentially meagre.

It was mentioned earlier that Fourier analysis of visuals and voice is the measure of intrinsic merit. Following definition of generic intrinsic merit further strengthens it:
Generic intrinsic merit of an entity - text, visual or voice - is the complexity of cerebral representation potential as received by sensory perception. For texts it is the graph complexity measure. For voice and visuals it could ERPs activated on seeing or hearing and based on EEG data.

References:

418.1 Event Related Potentials for attractive facial recognition - https://labs.la.utexas.edu/langloislab/research/event-related-potential-erp/
418.2 Event Related Potentials - http://cognitrn.psych.indiana.edu/busey/eegseminar/pdfs/Event-Related%2520PotentialsIntro.pdf - ERPs are cortical potentials of interoperating neurons in response to a cognitive stimulus.

---

419. (THEORY) Goodness of Link Graph Majority Voting and Complex Plane representation of merit - 22 June 2017 and 23 June 2017 - related to 418

Following is an example link graph in world wide web with weights for each edge:

```
y1 - x1 : 0.25
y1 - x2 : 0.25
y1 - x3 : 0.25
y1 - x4 : 0.25
```

implying y1 votes to x1,x2,x3,x4 with weight 0.25 each.

Following is the corresponding initial decision correctness (goodness) graph with goodness for each edge:

```
y1 - x1 : 0.25
y1 - x2 : 0.25
y1 - x3 : 0.375
y1 - x4 : 0.125
```

implying y1 votes to x1 with correctness 0.25, x2 with correctness 0.25, x3 with correctness 0.375 and x4 with correctness 0.125.

Markov iteration on the decision correctness graph (similar to random walk on link graph) tends to a stationary distribution after certain number of random walks. Cumulative goodness of voter v1 is defined as:

```
summation(weight(edge) * goodness(edge))
```

For previous example cumulative goodness of voter y1 is:

```
0.25*0.25 + 0.375*0.25 + 0.25*0.25 + 0.25*0.25 =
1/32 + 3/32 + 2/32 + 2/32 = 0.25
```

If all votes are 100% correctly decided by y1, correctness graph has 1 for all edge weights. Thus cumulative goodness of y1 is:

```
0.25 * 1 + 0.25 * 1 + 0.25 * 1 + 0.25 * 1 = 1 = 100%
```

After markov iteration attains stationary distribution, cumulative goodness of all vertices also becomes stationary. Atleast one incoming vertex y voting to another vertex x having cumulative goodness < 0.5 at the end of markov iteration implies group decision is a failure and does not concur with the real merit of a vertex x.

Following are the votes received by a vertex x1 from voters y1,y2,y3,y4 having respective cumulative goodness:

```
y1 - x1: 0.75
y2 - x1: 0.3 (voter with cumulative goodness < 0.5)
y3 - x1: 0.8
y4 - x1: 0.9
```

What is the probability in average case that atleast one voting vertex has cumulative goodness < 0.5? Link graph voting is different from normal voting - it is peer to peer and each voter apportions vote to the neighbouring candidates - single voter can vote multiple candidates simultaneously.

Pr[atleast one adjacent vertex y to a candidate vertex x has cumulative goodness < 0.5 after markov iteration] = 1 - Pr[goodness > 0.5]

From Markov inequality tail bound, Pr[cumulative goodness > 0.5] <= mean/0.5
Pr[cumulative goodness < 0.5] = 1 - Pr[goodness > 0.5] >= 1 - mean/0.5
where mean is the average cumulative goodness of all vertices in link graph.

When mean is 0.5 (uniform distribution), Pr[cumulative goodness < 0.5] >= 1 - 0.5/0.5 = 0
For any other values of mean < 0.5, Pr[atleast one adjacent vertex y to a candidate vertex x has cumulative goodness < 0.5 after markov iteration] = 1 - mean/0.5 is always greater than zero.

In discrete random variable case, if goodness takes n discrete values between 0 and 1 then, mean = summation(x*p(x)) where p(x) = 1/n is:
n(n+1)/2*n*n = 0.5 + 0.5/n which tends to 0.5 when n tends to infinity.
Thus mean cumulative goodness is 0.5 in uniform distribution.

Objective and Subjective rankings can be represented on a complex plane with following notation:
(intrinsic merit=m) + i(perception=p) = m + ip
where p is a function of m. Any text,visual or voice can be plotted on complex plane in previous notation. Ideally m should be equal to p. This creates a special case when m is fixed and p varies for an entity which usually happens with perception based majority voting. Such set of complex numbers with fixed merit and varying perception can be thought of as zeros of a complex valued function. Famous special case is Riemann Zeta Function with merit=0.5 and varying perception imaginary parts.

Riemann Hypothesis if true implies Re=0.5 for non trivial zeros of Riemann Zeta Function. If a set of complex number rankings as described previously have Re=0.5 or fixed real part and varying imaginary parts equalling non trivial zeros of Riemann Zeta Function, then the rankings show pattern in prime numbers an unusual connection.

Any k-coloring of a sequence of text and audio-visuals(AVs) denoted by integers is a classifier. 2-coloring or complementation is a special case of such a classifier. Conversely, any classifier is a coloring scheme for sequence of text and audio-visuals(AVs). For example following bit pattern is a 2-coloring/complement function - 0 and 1 are colors and f and g are complement functions illustrated as dotted indentations:
11110000111100001110000111100011111100000

```
----  ----  ---    ---   ------          function f:
   .....    ----   ----   ---      -----  complement g:
```

Vapnik-Chervonenkis Shattering or VC Shattering is defined as:
Let H be a set of sets and C be a set. C is shattered by H if H intersection C = powerset of C = 2^C. Intuitively, H classifies C with labels from the set H. Largest cardinality of C shattered by H is the VC dimension.

2-coloring/complementation is a classifier on real line or integer sequences. 2 colors/binary encodings/complementations create a set H of size equal to powerset of real line of integer sequence. Largest cardinality of real line or integer sequence that can be shattered by H (H intersection C = 2^C) is infinite. Thus 2-color/complementation classifier of infinite stream of sequences has VC Dimension Infinity.

References:

419.1 Probability and Statistics with Reliability, Queuing and Computer Science Applications - Pages 223-225 - [Kishor Shridharbhai Trivedi-Duke University]

---
420. (THEORY) Human Resource Analytics, Interview Algorithm and Intrinsic Merit - related to 314, 359 and 365 - 26 June 2017 and 27 June 2017

Stability of Interview TQBF circuit which is the theoretical formalism of

Interview Algorithm Intrinsic Merit has been analyzed earlier as opposed to stability of Majority Voting. Extending the notion of merit from WWW to humans is the most obvious consequence. Usual interview procedure in industry is to screen resumes, shortlist them and interview for few rounds and make an offer. Does this tradition measure merit flawlessly? No. Error in interview and its stability has been analyzed in 359 and 365. Is there a way to circumvent this error and remove human element completely? Error in interview process applies to examination system in academics too. Real-life interviews and examinations have duration of few hours and rely on question-answering. Can few hours measure years of accrued merit? This question is reduced to sampling problem. If merit is a scatter-plot of feature vector points on a metric space V of n-dimensions, a sample is a set of subspaces S which approximate V. Contraction Mapping (and Banach Fixed Point Theorem) maps a space to a subspace and there is always a unique fixed point in this map (f(x)=x for some x in V). An intrinsic merit analyzer can be thought of as a contraction map on the metric space of merit feature vectors of an individual. This contraction map must create a sample subspace in such a way that all merit feature vectors are near-perfectly measured. This involves 2 difficult problems:
(*) Construct a complete merit metric feature vector space for an individual based on past records(work and academic).
(*) Construct a contraction map(s) which contracts this merit vector space into a sample subspace. This contraction map is then translated into an interview TQBF function. Objective Question-Answers (multiple choice) are relatively easy to construct and evaluate than descriptive, subjective question-answers.

Real life interviews typically have following usecase (e.g IT industry) - a slightly modified version of previous TQBF construction:
(*) suitability for a requirement
(*) technical discussions (question-answering on programming/projects etc.,)
(*) experience

Intrinsic Merit of Collection of Humans are measured in economics literature by many indices like GDP,Human Development Index(HDI),Inequality adjusted HDI,Gender Inequality Index,Purchasing Power Parity(PPP) etc.,Human Development Index defined as geometric mean of standard of life,income,education indices is used to rank countries by their development.Intrinsic Merit of sportspersons are measured by Intrinsic Performance Rating(IPR) measures e.g Elo rating in Chess is used to rank players. Translating the previous example, one possible IPR for interview is the following adapted from HDI (not necessarily perfect):

IPR(candidate)=geometric_mean(IPR(interview)*IPR(education)*IPR(experience))

From 359 and 365, NoiseSensitivity(InterviewTQBF) which is the dual of Stability is defined as:
NoiseSensitivity(InterviewTQBF) = 0.5-0.5*Stability(InterviewTQBF) = 0.5-0.5*a^2*n^2/4

Error in interview is equal to NoiseSensitivity of interview TQBF. What is the probability of interview process failing? By Markov tail bound:
Pr[Error in interview process > e] <= mean/e where mean is the average error or NoiseSensitivity of interview. Therefore:
Pr[Error in interview process > e] <= NoiseSensitivity(InterviewTQBF)/e

While suitability can be easiliy quantified for error and interview TQBF having previous error bound, experience is a new variable in real-life HR analytics. Experience of a candidate is total duration in industry (academics,private initiatives inclusive). Experience itself is indirectly caused by past interviews. For example, a person having 30 years experience from 10 companies was meritorious in past 10 interviews and it is counterintuitive if 11th finds no merit (this contradiction makes objective merit subjective and in a sense this is also an error in interview process - a flawed TQBF). Thus experience is a function of merit and gradually makes future interviews redundant - an example below:
Experience = f(number of job/academic/private hops, intrinsic merit at

each previous hop, experience per hop)

Interview algorithm being a TQBF satisfiability problem is PSPACE-complete (=IP=AP). There are existing question-answering systems like IBM watson (answer-questioning) which beat humans in Jeopardy with least error, a contest like Turing test, Question-Answering using WordNet as Word Sense Disambiguator and older expert-system based Q&A software. But there does not seem to exist a theoretical decision tree parallel for QBF similar to boolean decision trees except DPLL evaluations. An error in interview QBF can also be formalised as a Lambda-Tolerant Randomized Decision Tree having access to pseudorandom bits and making errors with an upperbound while evaluating decision tree.

TQBF formulation of question-answering is far stringent than traditional interviews. Existential and Universal quantifiers simulate "there exists an answer or a counter-question for all questions" instead of "there exists an answer for a question". In proving lowerbounds for games like Chess, Go etc., TQBF is evaluated as game-tree (alpha-beta pruning) - "for all moves there exists a countermove" where each level choices in the tree alternate between 2 opponents for bounded (polynomial) or unbounded (exponential) number of rounds.

Intrinsic merit metric space of feature vector points can be construed as a Hypergraph with feature vector points as vertices and edges spanning multiple of these points which are related as edges. Hypergraph edge connects more than two vertices. Transversal or Hitting Set of a Hypergraph is the subset S of the vertices X which have non-empty intersection with all hyperedges. Transversal graph is a subgraph of this hypergraph consisting of all possible minimal transversals(a minimal transversal has no other traversals as subsets). Transversal graph is a "summary" of the larger graph and thus is a Contraction Map which creates a gist of the merit metric space.

Similar notion of transversal hypergraph can be applied to Recursive Gloss Overlap Definition Graph too (considering it as a hypergraph), for text summarization - subset of word vertices which intersect all hyperedges. Hypergraph Transversal Problem is known to be in co-NP.

Like usual text documents, candidate resumes can be represented as either a Recursive Gloss Overlap graph or a ThoughtNet hypergraph:
*) Graph: Resume text is mapped to a graph by Recursive Gloss Overlap algorithm. Core number based classifier brings out the best in the graph - candidate experience domains in resume that are closely related
*) Hypergraph: Resumes are stored in ThoughtNet as Hypergraph index. Querying results in similar resumes. ThoughtNet internally invokes Recursive Gloss Overlap core number classifier.

References:

420.1 P,NP and examinations - https://terrytao.wordpress.com/2009/08/01/pnp-relativisation-and-multiple-choice-exams/
420.2 Shrink Map and Contraction Map - [Topology - James Munkres] - pages 181-182
420.3 Banach Fixed Point Theorem - https://en.wikipedia.org/wiki/Banach_fixed-point_theorem
420.4 Davis-Putnam-Logemann-Loveland (DPLL) decision tree solver algorithm for QBF - http://personalpages.manchester.ac.uk/student/joshua.dawes/notes/qbf.pdf
420.5 Parallel algorithm for Hypergraph Transversals - https://people.mpi-inf.mpg.de/~elbassio/pub/COCOON05.pdf
420.6 Efficient algorithm for Hypergraph Transversals - http://jgaa.info/accepted/2005/KavvadiasStavropoulos2005.9.2.pdf
420.7 Compendium of Intrinsic Performance Ratings in Chess - https://www.cse.buffalo.edu/~regan/papers/pdf/Reg12IPRs.pdf
420.8 Human Development Index (New) - https://poseidon01.ssrn.com/delivery.php?ID=615100008007024087109082112068031068002059093093031010095070006067125121072104018099058027096058051040011088102003094012124014108058062055076070124088071097101123070065059085125012119018080072084029012007002119065081015116009119108104075

076102112012027&EXT=pdf
420.9 Social Progress Index -
https://www.socialprogressindex.com/assets/downloads/resources/en/English-2017-Social-Progress-Index-Methodology-Report_embargo-until-June-21-2017.pdf -
Comprehensive ranking of countries based on multitude of social indicators
(Basic Needs, Wellbeing, Opportunities)

---
421. (FEATURE-DONE) Text Summarization from Recursive Gloss Overlap Graph Core - Commits - 28 June 2017

(*) Added a new function to create a summary from text - This function creates the core of a Recursive Gloss Overlap graph
with certain core number and writes out a text sentence for each edge in core subgraph obtaining least common ancestor(hypernym) relation
(*) logs for this have been committed to python-src/testlogs

Summarization from k-core(s) of a Recursive Gloss Overlap graph captures the most crucial areas of the text because document belongs to the class/word vertex with high core numbers.

---
422. (FEATURE-DONE) Text Summarization from Recursive Gloss Overlap Graph Core - Commits - 30 June 2017

(*) New clause added to classify the text and match the prominent core number word vertices in the text and only add those sentences to summary.
(*) This is an alternative to k-core subgraph traversal and creating text programmatically. It is based on the heuristic that a summary should
capture the essence/classes the text belongs to.
(*) Prominent core number classes are shaved off from the sorted core number list returned by RGO classifier and top percentile is used. Summary is limited in size relative to the original text.

---
423. (THEORY) Dense Subgraph Problem and Mining patterns in Graph Representation of Text - 2 July 2017

Recursive Gloss Overlap and Recursive Lambda Function Growth algorithms create graphs from text documents. Unsupervised classification from
prominent vertices of the definition graph and Text summarization at present depend on finding k-core subgraphs. In general this is an NP hard
problem to find Dense Subgraphs of a Graph where density of a subgraph S of a graph G is defined as:

$$d(S) = |\text{Edges of } S| \;/\; |\text{Vertices of } S|$$

There are polynomial time maxflow based algorithms and approximations to find dense subgraphs of a graph.

References:

423.1 Goldberg Algorithm, Charikar Algorithm and k-Cliques Densest Subgraph Algorithm for Dense Subgraph Discovery - people.seas.harvard.edu/~babis/dsd.pdf
423.2 Network Structure and Minimum Degree - [Social Networks - Seidman] - https://ucilnica.fri.uni-lj.si/pluginfile.php/1212/course/section/1202/Seidman%20-%20Network%20structure%20and%20minimum%20degree%2C%201983.pdf - k-cores as measure of connectedness in a network G which are maximal connected subgraphs of G whose vertices have degree atleast k.

---
424. (THEORY) Pseudorandom non-majority choice and Bounded Electorate Majority Choice - 3 July 2017 - related to 53.7

Bounded Electorate Majority Choice:

Odd Electorate with 3 voters is the minimum possible majority voting setting which vouchsafes a clear winner. This is a composition (both formula and circuit) of NC majority voting function with Voter SATs. This composition can be written as a unified formula by substituting the voter SAT formulas in formula for Majority function which has formula of size O(n^5.3). This combined formula can be converted to a 3-CNF by Tseitin transformation. This specific 3 voter example is NP-complete. Similar substitution for infinite electorate majority could be undecidable.

Pseudorandom Non-majority Choice:

Majority social choice has only one level of error probability i.e goodness for each voter while Non-majority pseudorandom social choice invoking a PRG has 2 levels of probabilities - 1) Probability of choosing a voter SAT at random 2) Goodness probability of chosen voter SAT and thus conditional.

Let number of voter decision functions with goodness xi = m(xi). Total number of voters N = m(x1)+m(x2)+m(x3)+...+m(xn)
Effective goodness of a PRG choice is the mean:
1/N * summation(xi*m(xi)) = (x1*m(x1) + x2*m(x2) + x3*m(x3) + ... + xn*m(xn)) / N
When all voter functions have goodness 1 then PRG choice has effective goodness 1.

Probability of choosing a voter of Goodness xi = m(xi)/N
Conditional goodness probability of a PRG chosen voter SAT = Pr[choosing voter SAT of goodness xi] * Pr[goodness of SAT] = [m(xi)/N]*xi

When m(xi)*xi/N = 1, Goodness of PRG choice is 1. This can happen only if m(xi)*xi = N => m(xi) = N and xi = 1 i.e all voters have equal goodness 1 because smaller values of xi require m(xi) > N , a contradiction. This is a BPP/BPNC/RNC/RP algorithm despite the goodness being 1 because Pseudorandom bits have to be created by a PRG.

When goodness is 1 for both LHS PRG choice and RHS Bounded Electorate Majority Choice, PRG choice is a BPP/BPNC/RNC/RP algorithm to NP-complete Majority Choice for finite voters => NP is in BPP/BPNC/RNC/RP. But BPP is in P/poly. Therefore NP is in P/poly if NP is in BPP. From Karp-Lipton theorem if NP is in P/poly, PH collapses to Sigma(p,2) and NP in P/poly implies AM=MA by [Arvind, Vikraman; Köbler, Johannes; Schöning, Uwe; Schuler, Rainer (1995)]. This independently leads to a similar bound mentioned earlier for Percolation Boolean Voter Functions for Non-majority social choice which have 100% Noise stability (LHS is a P/poly percolation circuit with 100% goodness while RHS is NP-complete finite electorate).

NP in P/poly also implies PH is in P/poly. It is not known if this implies PH-complete problems exist (because PH collapsing to second level causes every k-QBFSAT problem in PH to reduce to 2-QBFSAT sigma(p,2) and thus PH-hardness is proved).

PH-completeness proof outline:

If NP is in BPP, NP is in P/poly because BPP is in P/poly.
=> If NP is in P/poly, PH is in Sigma(p,2) from Karp-Lipton-Sipser Collapse Theorem
=> If NP is in P/poly, PH collapses to P/poly

=> There are complete problems in each level k of the Polynomial Hierarchy(correspond to a k-depth QBFSAT).
=> There is a complete problem in Sigma(p,2) corresponding to 2-QBFSAT.
=> All problems in Sigma(p,2) can be reduced to this Sigma(p,2)-complete 2-QBFSAT problem.
=> All problems in PH collapse to Sigma(p,2) if NP is in P/poly
=> All problems in PH can be reduced to this Sigma(p,2)-complete 2-QBFSAT problem.
=> Sigma(p,2)-complete problem is thus a PH-Complete problem

References:

424.1 Proof of Karp-Lipton-Sipser collapse theorem - PH in P/poly - http://www.cse.iitm.ac.in/~jayalal/teaching/CS6840/2012/lecture14.pdf

---
425. (FEATURE-DONE) Updates to Text Summarization from Dense Subgraph of Recursive Gloss Overlap graph of a Text - Commits - 4 July 2017

(*) import matplotlib commented in RGO classifier
(*) New clause added to Text Summarization: This finds the common path between 2 word vertices synsets by inheriting an existing WordNet code for shortest wordnet path distance and constructs sentences by a sliding window for each successive pair of intermediate vertices in this common path. This creates a deeper profund summary than least_common_hypernyms().Sentences thus created are sorted based on a relevance score to the actual text - size of intersection similar to Lesk WSD. Only top relevant sentences can be chosen as required.
(*) Logs for this have been committed to testlogs/.

It has to be noted that, dense subgraph traversal and writing unguided summary sans training data mimicks human recursive comprehension and looks non-conventional for human reading. Presently only hypernym (IS A) relation is used for connectives.
More comprehensive humane-looking summary can be created by Meronyms (HAS A) and Holonyms(IS PART OF) in WordNet API. ConceptNet 5 which is a better semantic framework for finding relations between concepts could do well in this context. But presently there is no python API for ConceptNet5 and data has to be queried as RESTful JSON objects.

---
426. (FEATURE-DONE) Updates to Text Summarization - Commits - 5 July 2017

(*) Changed the class-sentence matching clause by increasing the percentile of prominent classes
(*) Changed relevance_to_text() by comparing each sentence chosen by prominent class match to the sentences in text - Ratcliff-Obershelp Longest Common Subsequence matching difflib library function is invoked for this similarity. Sentences are chosen also based on relevance_to_text() scoring
(*) Percentage of summary to the size of the text has been printed as a ratio. Logs have been committed to testlogs/

---
427. (FEATURE-DONE) Updates to Text Summarization - choosing sentences matching class labels - 6 July 2017

(*) Some experimentation on choosing relevant sentences to be added to summary was performed
(*) relevance_to_text() invocation has been changed as per algorithm below:
for each prominent dense subgraph k-core class label
find the synset definition of class label
for each sentence
invoke relevance_to_text() similarity function between sentence and the class label definition
and add to summary if the relevance ratio > 0.41 and if not already in summary
(*) Ratio 0.41 was arrived at heuristically:
- For relevance ratio 0.1, summary ratio was 0.65
- For relevance ratio 0.2, summary ratio was 0.48
- For relevance ratio 0.3, summary ratio was 0.35
- For relevance ratio 0.4, summary ratio was 0.21
- For relevance ratio 0.5, summary ratio was 0.02
There is a drastic dip in size of summary if relevance threshold is increased beyond 0.4. Because of this 0.41 has been
hardcoded.
(*) Logs for this have been added to testlogs/
(*) Summary generated is human readable - subset chosen from actual text.

---
428. (FEATURE-DONE) ConceptNet 5.4 Python RESTful API implementation - lookup, search and association - 7 July 2017

(*) This commit implements the RESTful Python requests HTTP API for querying ConceptNet 5.4 dataset
(*) Three functions for looking up a concept, searching a concept and finding similar associated concepts have been
implemented with 3 endpoints (as per the documentation in https://github.com/commonsense/conceptnet5/wiki/API/2349f2bbd1d7fb726b3bbdc14cbeb18f0a40ef18)
(*) ConceptNet is quite different from WordNet in representation as JSON dictionary as against graph in WordNet.
(*) ConceptNet 5.4 endpoints have been invoked instead of 5.
(*) If necessary ConceptNet can replace all WordNet invocations in a later point in time. But such a replacement is non-trivial and would
probably require almost all Recursive Gloss Overlap related code to be rewritten from scratch. But that depends on how ConceptNet 5
weighs against WordNet in measuring semantic meaningfulness.

---
429. (THEORY) Contradictions in bounds between finite and infinite electorate - 8,10,11,12,13 July 2017

Lowerbounds by equating the goodness of Non-majority and Majority social choice thus far mentioned in drafts in this document (subject to errors) are based on following assumptions:
(*) There are two possible paths to social choice - Non-majority and Majority
(*) Each social choice belongs to a computational complexity class
(*) Goodness of a social choice is the measure of error-free-ness of the choice made in either paths i.e decision correctness (e.g noise stability, sensitivity,error in BP* algorithm etc.,)
(*) RHS Majority voting has been assumed to abide by Homegeneous version of Condorcet Jury Theorem convergence and divergence of group decision goodness as a function of individual voter goodness while LHS is either a pseudorandom choice or an interview TQBF algorithm
(*) Goodness of either choice majority or non-majority must be equal
(*) Either LHS or RHS has to be a complete problem for a complexity class

C.

(*) Traditional literature on boolean majority functions and circuits assumes that input to majority is readily available which is equivalent to SAT oracle access to Majority function where SAT oracle could belong to any complexity class (2-SAT, 3-SAT, k-QBFSAT, etc.,) => Majority voting is in P^NP,P^PH,P^EXP etc.,

(*) Previous Oracle access based proofs have been intentionally circumvented by replacing Oracles with Boolean Function/Circuit compositions which have strong Communication Complexity basis (KW relations and depth of a circuit composition mentioned in 368)

(*) Assuming all above, LHS is an algorithm for RHS complete problem (or viceversa) creating a lowerbound.

(*) All bounds derived in this draft assuming above do not follow conventional lowerbound techniques e.g Circuit Lower Bounds. Equal goodness assumption implies - "both algorithms solve same problem - one is more efficient than the other".

(*) Most importantly drafts in this document are just analyses of various social choice functions, their complexities and contradictions irrespective of attaining lowerbounds.

Equality of Goodness of PRG choice and Majority voting for finite electorate of size 3:

Let x1,x2,x3 be the goodness of 3 voter SATs. PRG choice randomly chooses one of the 3 voters while majority voting is the usual CJT Majority+SAT composition.

Goodness of PRG choice:

$= (x1*m(x1) + x2*m(x2) + x3*m(x3)) / 3$

When all 3 have equal goodness 1, effective goodness is:

$= (1*1 + 1*1 + 1*1)/3 = 1$

Goodness of Majority choice:

This is the bounded version of CJT binomial series summation. When x1=x2=x3=1:

$= (3C2(1)^2(0)^1 + 3C3(1)^3(0)^0) = (0 + 1) = 1$

Possible Lowerbounds described previously for Unbounded and Bounded electorate lead to contradictions as below:

(*) In infinite voter case, homogeneous voter CJT circuit in BPP is derandomized to P if CJT converges => P=BPP.

(*) In finite voter case, NP-complete RHS has BPP algorithm in LHS (NP in BPP) implying NP in P/poly and collapse of PH => NP in P/poly and PH in P/poly.

Contradiction 1 :

If BPP=P (unbounded CJT) and NP is in BPP (bounded) then NP is in P=BPP => P=NP. This conflicts with P != NP implied by :

(*) the Majority Hardness Lemma (318),

(*) Polytime learning of NP implying NP does not have polytime algorithms (368) and

(*) HLMN PARITYSAT counterexample (53.15)

but concurs with :

(*) high percentage of random k-SATs satisified in Approximate CNF SAT solver by least squares (376).

But can infinite voter CJT circuit be in BPP and thus in P/poly? Infinite voter CJT circuit is of polynomial size if it is polynomial in number of Voter SAT variables and exponential if it is exponential in number of variables. Latter happens only if all voters have dissimilar SAT variables while common variables across voter SATs make it exponential. This is described in example of 53.8. Condorcet Jury Theorem convergence in homogeneous voters case implies all voters are similar (e.g have similar voting SATs) thus ruling out polynomial size case

i.e Unbounded CJT circuit can not both be in BPP and converging. This contradiction stems from the assumption - BPP derandomizes to P. LHS could be in RP too. RP is contained in NP. (Can BPP derandomize to NP i.e BPP in NP?). Another assumption is all voters have 3-SAT choice functions. If all voter SATs are 2-SATs (in P), infinite and finite voter cases are not equatable and there is no contradiction. Known result: If NP is in BPP, NP=RP - this applies to bounded voting case above. Thus BPP=P possibility is removed by exponential sized CJT circuit and BPEXP=EXP is still possible when CJT converges. Contradiction 1 is avoided.

Contradiction 2 :

If NP is in BPP and thus in P/poly (irrespective of BPP=P), similar conflicts arise. An assumption made in composition of 3-SAT voters and Majority function for bounded electorate in 424 is resultant composition is also in NP (depth lowerbound for this composition can be obtained by KW relations in 368). This assumption could be false because composition of NP 3-SAT with non-uniform NC1 majority function could be harder than mere NP - because this is equivalent to replacing oracle with a circuit composition in a P^NP algorithm. P with NP oracle adds an additional quantifier and places it in second level of polynomial hierarchy (inclusion in https://www.cse.buffalo.edu/~regan/papers/ComplexityPoster.jpg shows there are problems in Sigma(p,2) /\ Pi(p,2) which also have BPP algorithms and P^NP is contained in Sigma(p,2) /\ Pi(p,2)). Thus a PRG choice and converging bounded electorate voting of equal goodness need not imply NP in BPP. This composition may not be a complete problem. Stricter oracle definition of Majority+SAT composition is NC1(L) where Majority is computable by non-uniform circuits or BWBP with oracle access to gates belonging to a class L. For 3-SAT voter oracles, CJT majority voting circuit is in NC1(NP) (is this contained in P^NP?). Subject to equal goodness, composition equivalent of NC1(NP) is in BPP and not NP is in BPP.

Reference 429.3 suggests there exists a random oracle A relative to which NC^A is in P^A. P having NP oracle access is known as class delta(p,2)=P^NP. This answers if NC1^NP is in P^NP in the affirmative. There is a known delta(p,2)-complete problem mentioned in reference 429.5. If bounded electorate NC1^NP problem is complete for its class, then equal goodness of PRG choice and bounded majority voting implies NC1^NP is in BPP. This in turn implies NP is in BPP (if NC1^NP strictly contains NP) and thus NP is in P/poly again leading to contradiction in the outset.

Replacing oracles with compositions and applying depth lowerbounds for Majority+VoterSAT circuit composition as below for m voters with n variables per voter SAT (from 368):

D(Maj + Voter) = CommunicationComplexity(R(Maj,Voter)) >= 5.3*logm + n - O(m*logm/n)

does away with the hassles of relativization and directly gives the depth lowerbound of the majority voting circuit composition. Size of this composition is the alternately phrased KRW conjecture in 429.7.

Example delta(p,2)-complete problem mentioned in 429.5:

While number of queries to NP oracle <= k
{
(*) Query a 3SAT oracle for a satisfying assignment for a 3CNF Voter SAT
(*) Store the queried satisfying assignment in a linked list in sorted order - this is linear per insertion i.e traverse the list and compare the two adjacent nodes. E.g. {1,5}, {1,3,5}, {1,3,5,6}, {1,3,5,6,7}... upto k-th query and lexicographic ordering is preserved by a decimal encoding of SAT oracle query string.
}
Output 1 if last element in the list ends with binary 1 digit.

This algorithm makes k queries to an NP oracle and has O(k^2) time complexity

and thus in delta(p,2). Hardness follows by reducing any polynomial time algorithm with NP oracle to above - reasonably straightforward because oracle queries are memoized and can be looked-up in polynomial time.

Mapping above delta(p,2) complete problem to proving completeness of NC1^NP majority voting is non-trivial. Majority function can make n queries to n voter SAT oracles and atleast n/2 + 1 queries should return 1 or 0 to compute majority in NC1. Beyond this, NC reducibility has to be proved by Many-one/Turing reductions or Logspace reductions of any other problem in NC1^NP to this. Integer multiplication,powering and division have been proved to be equivalent (NC reducible to each other) mentioned in reference 429.8. NC reducibility through oracle NC gates have been proved in reference 429.9. Majority function is known not to be complete for NC1 under AC0 many-instances-to-one reductions (reference 429.10). This could probably imply that NC1^NP is not complete too, thereby avoiding contradiction 2.

But this also implies that any Majority^A for a random oracle A is not complete if Majority is not complete for NC1. Hence it has to be proved or disproved if a problem is not NC1 complete, it is not NC1^A complete for an oracle A. Lemma 3.3.9 in reference 429.9 [RuzzoGreenlawHoover] describes an Oracle PRAM or an NC oracle circuit M' to another NC circuit M. M has O(n^c) size/processors and depth/time of O(logn) and M makes at most O(n^c) oracle queries to M' (each node can query). But these oracle queries are made simultaneously in parallel time of O(logn). Thus replacing calls to M' by M' itself increases depth by O(logn) and size by O(n^c) order of magnitude i.e new circuit without oracle M' has time O((logn)^2) and size O(n^2c). Similar oracle replacement could be done for Majority^A circuits too. If oracle gates to Majority are replaced by circuit for A itself which has size s and depth d and Majority makes at most O(n^c) calls to oracle A, new Majority circuit without oracle has depth/time O(d*logn) and size/ processors O(s*n^c). This new circuit need not be in NC1. From Spira's Theorem a circuit of size O(s*n^c) can be transformed into a circuit of depth O(log(s*n^c))=O(log(s) + c*log(n)). Following cases arise after replacing oracle gates with a new circuit of size s.

Case 1 - s = 2^n:

This makes the new majority circuit sans oracle to be of depth O(n*log2+c*log(n)) which is not polylog depth and polynomial size and thus lies outside NC - this new circuit problem could be complete for a different class. This line of reasoning coincides with previous formulations of majority voting based on depth bounds by Communication Complexity (KRW Conjecture) and Direct Connect circuit families of unbounded depth and exponential size. If there is a problem B in NC1, Majority^A and B^A can be in totally different depth hierarchy classes by depth hierarchy theorem. This could be a complete problem in different class (e.g, PH-complete, EXP-complete etc.,) and can be equated under equal goodness assumption with a non-majority social choice.

Case 2 - s = n^c:

This new majority circuit has depth O(2*c*log(n)) and size O(n^2c) and is obviously in NC1 and computes majority. But majority is not complete for NC1 and thus majority voting itself is not a complete problem and cannot be equated under equal goodness assumption with a non-majority choice.

References:

429.1 Definition of Homogenenous Voter - http://www.uni-saarland.de/fak1/fr12/csle/publications/2006-03_condorcet.pdf - "...Now assume that a chamber consists of three homogenous judges. Homogenous means that the decision-making quality of each single judges is described by identical parameters r..." - identical parameters are translatable to identical variables though SAT could be different for each
429.2 Counting Classes and Fine Structure between NC and L - [Samir Datta ,

Meena Mahajan , B V Raghavendra Rao , Michael Thomas , Heribert Vollmer] - http://www.imsc.res.in/~meena/papers/fine-struct-nc.pdf - Definition of NC circuits with Oracle gates. Definitions 8 and 10 and Remark 9 - Majority NC1 circuit with NP oracle for all homogeneous voters belongs to Boolean Hierarchy and specifically is in NC1 hierarchy.
429.3 For a random oracle A, NC^A is strictly contained in P^A - https://complexityzoo.uwaterloo.ca/Complexity_Zoo:N, https://complexityzoo.uwaterloo.ca/Zooref#mil92 - [PeterBroMiltersen]
429.4 Delta(p,2) - P has NP oracle - https://complexityzoo.uwaterloo.ca/Complexity_Zoo:D#delta2p
429.5 Delta(p,2) complete problem - https://complexityzoo.uwaterloo.ca/Zooref#kre88 - [Krentel] - Given a Boolean formula, does the lexicographically last satisfying assignment end with 1?
429.6 Spira theorem - any formula of leaf size s can be transformed into a formula of depth log(s) - https://www.math.ucsd.edu/~sbuss/CourseWeb/Math267_1992WS/wholecourse.pdf
429.7 Size of a circuit composition - alternative form of KRW conjecture - http://www.math.ias.edu/~avi/PUBLICATIONS/GavinskyMeWeWi2016.pdf - "...This suggests that information complexity may be the “right” tool to study the KRW conjecture. In particular, since in the setting of KW relations, the information cost is analogous to the formula size, the “correct” way to state the KRW conjecture may be using formula size: L(g*f) ≈ L(g)·L(f)..." - for Majority+VoterSAT composition size is conjectured as O(n^5.3 * s) where s is the size of VoterSAT formula. Information complexity of a composition of a function g:{0,1}^m->{0,1} and a universal relation Un:{0,1}^n->{0,1} for majority voting is: a Voter SAT g of m variables is composed with a Majority universal relation of n variables. Majority is a universal relation because for 2 input strings to majority function drawn at random can be checked if they differ in a bit position. Above depth bound is the amount of mutual information "leaked" while computing the composition together by Alice and Bob. Alice gets a m*n matrix X and a string a in ginverse(0) and Bob gets a m*n matrix Y and a string b in ginverse(1) and both accept if a and b differ in a bit position and X and Y have row mismatch and reject else. ginverse(0) is equivalent to a rejecting assignment to SAT and ginverse(1) is equivalent to an accepting assignment to SAT because g() is the Voter SAT.
429.8 Log Depth circuits for Division and Related - [BeameCookHoover] - https://pdfs.semanticscholar.org/29c6/f0ade6de6c926538be6420b61ee9ad71165e.pdf
429.9 NC reducibility - reducing one NC instance to another - https://homes.cs.washington.edu/~ruzzo/papers/limits.pdf - replace NC oracle gates by multiplication of equivalent number of processors and depth
429.10 Catechism on open problems - Interview of Eric Allender - https://books.google.co.in/books?id=7z3VCgAAQBAJ&pg=PA37&lpg=PA37&dq=NC+reducibility+majority+function&source=bl&ots=WOzlJFRbpt&sig=OmFqckehlrTfyaeTOjscJ423dUk&hl=en&sa=X&ved=0ahUKEwjQ_O-Y1YPVAhVGMI8KHbJCDbAQ6AEIIzAA#v=onepage&q=NC%20reducibility%20majority%20function&f=false - "...Majority function is not complete for NC1 under AC0 many-one reductions..."
429.11 Parallel Computation and the NC hierarchy relativized - [Christopher Wilson] - https://link.springer.com/chapter/10.1007/3-540-16486-3_111 - containments of NC classes relative to oracle - "NC^A hierarchy is seen to be in P^A for any oracle A. Also, nondeterministic log-space relative to ... there exists an oracle A, such that NC1^A in NC2^A ... in P^A"

---

430. (THEORY) Theoretical Formalism for an Electronic Voting Machine based on Locality Sensitive Hashing - 13 and 14 July 2017 - related to 265,275,376 and 319

Locality Sensitive Hashing and their relevance to Multiway Contests have been described in 319. Voters voting for same candidate are clustered together in a bucket chain of tabulation hashing. This is akin to a naive electronic voting machine which increments counters of a candidate for each vote cast for

h(im/er). An LSH voting machine queries voter SAT oracles sequentially(or parallelly if there is a parallel LSH implementation), receives the candidate index, finds the candidate index key in LSH and appends the voter id to the bucket chain for the candidate. Sequential version of LSH voting machine is in delta(p,2)=P^NP because LSH is in P and oracle queries are made to voter 3SATs. LSH algorithms usually search for nearest neighbours and find similar items with high probability. For LSH based Electronic Voting Machine this rho parameter as defined in 319 is log(1/p1)/log(1/p2) where p1=Pr(h(x)=h(y)) if x and y had voted for same candidate and p2=Pr(h(x)=h(y)) if x and y had voted for different candidates. Simplest voting machine is an array of candidates and voters increment an array element for a candidate index which is exact and errorfree. But LSH voting generalizes the notion of voting as: "x and y vote for same candidate" is generalized to "x and y are similar or have similar liking". For example, a web search engine lists URL results for a query and all these URLs hash/vote to same query bucket in LSH parlance. Thus notion of exact candidate is replaced by an abstract similarity probability. Definition of rho thus allows error and its bounds are derived in references 430.2 and 430.3.

Non-boolean social choice functions which a voter computes to obtain a candidate index have been described previously. Previous LSH or array of counters based voting machine has following standard operating procedure:

(*) receives voters in a streaming sequence,

(*) each voter solves a non-boolean SAT (which could be in an arbitrary complexity class) having oracle access to it

(*) oracle returns a candidate index and counter is incremented

(*) LSH based voting has a special step - it compares two voters for similarity i.e if their oracle queries return same candidate index hash them to same bucket

(*) Multipartisan SAT Oracle internally has to implement the following:

- Iterate through all candidates
- Find the maximum number of clauses that can be satisfied by each candidate and quantify it as score which is NP-hard MAXSAT problem.
- Sort the scores for the candidates and return the topranked candidate.

(*) Sort the LSH by length of buckets

(*) MAXSAT has been used in lieu of Exact SAT for grading the candidates based on number of clauses satisfied. Exact SAT would return either 1 or 0 only and doesn't compute percentage of clauses satisfied.

(*) Random k-CNF SAT Solver implemented in 276 approximates by solving system of equations by least squares.

(*) Boolean Majority function is a special case of multipartisan voting: Restrict the number of candidates to 2 indexed as 0 and 1 and apply LSH.

(*) Voter SAT Oracles could be Constraint Satisfaction Problem(CSP) Solvers too which allow reals. This has been described in 265. Approximation of CSPs could be NP-hard if UGC is true.

Circuit Value Problem finds if a circuit encoding evaluates to 1 or 0 for an input assignment. This is equivalent to the following prover-verifier protocol:

(*) Voter has a Constraint Satisfaction Problem (boolean or real)

(*) Candidate has an assignment to the variables of voter CSP (Prover)

(*) Voter verifies the assignment to CSP (Verifier)

This kind of Circuit Value Problem (CVP) formulation has been avoided throughout drafts in document for Majority Voting. This is because, equating goodness and Condorcet Jury Theorem application requires quantifying the decision correctness (or) accuracy of Voter SAT/CSP which is not feasible to compute in CVP. Class PP generalizes BPP by removing bounds on error. From Toda's Theorem, PH is contained in P^PP and P^#P. Goodness in LSH voting is exactly the rho parameter and if each voter SAT oracle in LSH with unbounded error is in PP (extent to which a voter is misclassified in a candidate bucket), LSH voting itself is in P^PP and thus subsumes PH. Another aspect of LSH voting is majority circuit fan-in is replaced by size of buckets.

References:

430.1 Locality Sensitive Hashing - [Alex Andoni] - http://web.mit.edu/andoni/www/LSH/index.html
430.2 Lowerbounds for Locality Sensitive Hashing - [MotwaniAssafPanigrahi] - http://theory.stanford.edu/~rinap/papers/lshlb.pdf
430.3 Optimal lowerbounds for Locality Sensitive Hashing - [ODonnell] - https://www.cs.cmu.edu/~odonnell/papers/lsh.pdf
430.4 Reflections on Trusting Trust - Trojan horses in code - program that prints itself - Quines - Godel,Escher,Bach: An Eternal Golden Braid - Fixed Points in Turing Computable Functions - [Ken Thompson] - http://www.ece.cmu.edu/~ganger/712.fall02/papers/p761-thompson.pdf
430.5 Problems with Electronic Voting Machines (e.g DRE machines), VVPAT and Trusting source code - [Bruce Schneier] - https://www.schneier.com/blog/archives/2004/11/the_problem_wit.html
430.6 B-Cryptographic counters protocol augmented with Public Key Infrastructure for Incrementing Counters in Tabulation Authority - https://crypto.stanford.edu/pbc/notes/crypto/voting.html - Previous LSH based theoretical voting machine requires such a secure increment protocol for incrementing tabulation hashing buckets. Anonymity requires shuffling the buckets.
430.7 PP is as hard as polynomial time hierarchy - [Toda] - http://epubs.siam.org/doi/abs/10.1137/0220053?journalCode=smjcat and [SanjeevArora-BoazBarak] - http://theory.cs.princeton.edu/complexity/book.pdf
430.8 Beyond Locality Sensitive Hashing - [Alexandr Andoni,Piotr Indyk, Huy L. Nguy˜ên, Ilya Razenshteyn] - http://www.mit.edu/~andoni/papers/subLSH.pdf - Two level LSH partitioning by hash tables - Outer hashing creates buckets for the hash table and Inner hashing creates hash table for each buckets around a centroid.

---
431. (FEATURE-DONE and THEORY) Approximate CNF SAT Solver Update - percentage of clauses satisfied - 18 July 2017

(*) This commit prints percentage of clauses satisfied for each random CNF 3SAT formula for both 0 and 1 evaluations
(*) logs for ~1000 iterations of random 3SAT have been committed to testlogs/
(*) Interestingly, even failing formula assignments by least squares satisfy more than 75% of clauses of the formula
(*) Percentage of formulas satisfied by least squares heuristic is ~70%
(*) This is probably a demonstration of complement form of Lovasz Local Lemma (LLL) for MAXSAT which is:
if events occur independently with a certain probability, there is a small non-zero probability that none of them will occur
(*) Dual of LLL for this MAXSAT approximation is:
if each clauses are satisfied with a certain probability, there is a large probability that all of them are satisfied

References:

431.1 Lovasz Local Lemma - https://en.wikipedia.org/wiki/Lovász_local_lemma

---
432. (THEORY) MAXSAT ranking of text documents, Approximate CNF SAT solver and Lovasz Local Lemma - related to 431 -
19 July 2017

Considering a set of text documents, ranking function r(X,A), is the subjective perception measure of a text X
with access to a perception oracle A:
      r(X,A) = subjective rank of X as perceived by an oracle A
In the context of web link graphs of HTML documents, every adjacent vertex of a node N is the perceiver of N and for two

adjacent vertices A and B of N:
      r(X,A) = r(X,B) or r(X,A) != r(X,B)
Perception ranking of a node need not be decided just by adjacent vertices at the end of random walk markov iterations. For example, there could be vertices which are not adjacent but yet could have indirect unaccounted for perception. An example of this are the readers of a website who do not link to it but yet have a perception. This perception is not reckoned in subjective link graph ranking.

Ranking of texts can be thought of as MAXSAT problem: Each reader of a text has a CNF (subjective) or all readers have fixed CNF (Intrinsic Objective Merit) which they apply on the qualitative and quantitative attributes of a text. CNF for measuring merit conjoins clauses for measuring meaningfulness of the text. Each literal in this CNF is a merit variable e.g Graph complexity measures like Korner Entropy , Graph Tensor Neuron Network Intrinsic Merit, Connectivity etc., Rank of the text is the percentage of clauses satisfied. This CNF based ranking unifies all subcategories of rankings.

Lovasz Local Lemma:
Let a1,a2,a3,...,an be set of events. If each event occurs with probability < p, and dependency digraph of these events have outdegree at most d, then the probability of non-occurrence of all these events is non-zero:
      Pr[/\ !ai] > 0 , if ep(d+1) < 1 for e=2.7128...

CNF formula can be translated into a graph where each vertex corresponds to a clause and there is an edge between any 2 clausevertices if they have a common literal (negated or unnegated). In approximate least-square SAT solver, let d+1 be the number of clauses per CNF. Then at most d other clauses can have literal overlap and thus CNF clause dependency graph can have maximum outdegree of d. Each event ai corresponds to not satisfying clause i in CNF.

From LLL, If p < 1/[e(d+1)] is the probability of an event (not satisfying a clause) :
Probability that clause is satisfied: 1-p > (1-1/[e(d+1)])
Probability that all of the clauses are satisfied > (1-1/[e(d+1)])^(d+1) (for d+1 clauses)
Probability that none of the clauses are satisfied < 1-(1-1/[e(d+1)])^(d+1)
In previous example d+1 = 14 and thus probability that none of the clauses are satisified for any random CNF is:
= 1-(1-1/[e(14)])^(14) < ~31.11%
and probability of all the clauses being satisified for any random CNF is:
(1-1/[e(14)])^(14) > ~68.89%

Experimental iterations of SAT solver converge as below (10000 CNFs):
Percentage of CNFs satisfied so far: 71.14
Average Percentage of Clauses per CNF satisfied: 97.505

This coincides with LLL lowerbounds above and far exceeds it because outdegree (common literals) is less and average percentage of CNFs satisfied is 71% after 10000 iterations. It has to be noted that average percentage of clauses satisfied per random CNF is 97% which implies that very few clauses fail per CNF.

---
433. (FEATURE-DONE) Approximate CNF SAT Solver update - Commits 1 - 19 July 2017

(*) Average percentage of clauses satisfied per CNF is printed at the end of 10000 iterations
(*) AsFer Design Document updated for Lovasz Local Lemma analysis of least squares CNF SAT solver
(*) logs for least square assignment of 10000 random CNFs committed to testlogs/

---
434. (FEATURE-DONE) Recursive Gloss Overlap Graph Classifier update - Commits 2 - 19 July 2017

(*) printed Betweenness Centrality (BC) of the definition graph of a text
(*) BC of a node is the ratio of number of (s-t) shortest paths going through the node to all pairs shortest paths and
thus measures how central a node is to the graph and thus is another basis for classifying a text apart from dense subgraphs.

---
435. (FEATURE-DONE) Recursive Gloss Overlap graph classifier update - commits 1 - 20 July 2017

(*) Following centrality measures for the definition graph have been printed in sorted order and top vertices are compared:
1) Betweenness Centrality
2) Closeness Centrality
3) Degree Centrality
4) Eigenvector Centrality (PageRank)
5) Core numbers Centrality (k-core dense subgraphs)
(*) While first 3 centrality measures are reasonably equal in terms of ranking central vertices, PageRank centrality and k-core centrality are
slightly different
(*) But all 5 centrality measures reasonably capture the central purport of the text i.e the text on
"Chennai Metropolitan Area Expansion to 8848 sqkm" is classified into "Area" which is the topranked central vertex

---
436. (FEATURE-DONE) Approximate SAT Solver update - commits 2 - 20 July 2017

(*) Increased number of clauses and variables to 16 and iterated for ~3100 random CNF formulae.
(*) Following are the MaxSAT percentages after ~3100 iterations:
Percentage of CNFs satisfied so far: 69.1020276794
Average Percentage of Clauses per CNF satisfied: 97.6122465401
(*) Previous values are quite close to 14 clauses and variables iterations done previously.

Lovasz Local Lemma bound for 16 clauses all having common literals:
= (1-1/[e(16)])^(16) > ~68.9233868%
and average percentage of clauses satisfied after 3100 iterations far exceeds this bound at 97.61%

Commercially available SAT solvers (exact NP-complete decision tree evaluation based) usually scale to millions of clauses and variables.Though
this approximate polynomial time SAT solver does not have similar advantage, yet the huge percentage of clauses satisfied in repetitive iterations does seem to have a lurking theoretical reason. Infact this is symmetry breaking solver having a sharp threshold phase transition which tries to translate a system of equations over reals in [0,1] to discrete boolean CNF formulae(value below 0.5 is 0 and above 0.5 is 1). Hardness of Approximation and UGC prohibit polytime approximations unless P=NP. 98% satisfied clauses could imply 100/98=(1 + 0.020408163) approximation where epsilon=0.020408163. This has to be corroborated for millions of clauses and variables in least-squares and requires high performance computing.

Random number generator used in generating random CNF clauses is based on

/dev/random and hardware generated entropy. This CNF SAT solver depends on linux randomness to simulate pseudorandomness and create permutations of CNFs. Accuracy of previous convergence figures therefore might depend on if the CNFs are pseudorandom (e.g k-wise independent for k variables - when k increases to infinity independence may have infinitesimal probability).

References:

436.1 Linux Pseudorandom Generator - https://eprint.iacr.org/2012/251.pdf

---
437. (FEATURE-DONE) Approximate SAT Solver update - 24 July 2017

(*) Increased number of variables and clauses to (20,20).
(*) Changed SciPy lstsq() to NumPy lstsq() because NumPy has a faster lstsq() LAPACK implementation as against xGELSS in SciPy
(*) Logs for 100 random CNFs have been committed to testlogs/ and Percentage CNFs and Clauses satisfied are as below:
Percentage of CNFs satisfied so far: 58.4158415842
Average Percentage of Clauses per CNF satisfied: 97.1782178218

---
438. (THEORY) Tight Hamiltonian Cycles and Independent Sets in ThoughtNet Hypergraph for Social Networks - related to 40,222,229,295,351,418, 420 and other sections on ThoughtNet - 25 July 2017 and 28 July 2017

Theoretical description of ThoughtNet Hypergraph as basis for evocative thought and text analysis is mentioned previously. Hypergraph with 3 vertices per edge (3-uniform) and number of vertices n > n0, and minimum degree > n/2 + epsilon*n is guaranteed to have atleast one tight hamiltonian cycle from [Rodl,Rucinski,Szemeredi] theorem. Tight Hamiltonian Cycles have set of maximum vertex overlapping hyperedges. If complete universal knowledge is represented as 3-uniform ThoughtNet (infinite), this ensures all the concepts in universe are connected in a tight hamiltonian. Social media networks are also generalizable to hypergraphs (though tradition is to model as graphs having network flow). This can be simulated by ThoughtNet index - streamed social media tweets/posts/shares/likes are updated in ThoughtNet (non-planar) and each class stack vertex (e.g genre of people) could be part of multiple hyperedges (e.g events involving people) and each hyperedge can span multiple class stack vertices. Independent sets in this ThoughtNet Hypergraph are set of vertices e.g people who do not directly know each other. Centrality measures (e.g PageRank if there is one for hypergraph) on this hypergraph would elicit the social prestige/perception of individuals. It is worth quantifying how a vertex A is perceived by vertex B with no edge between them. This is a special case of centrality and gossip information flow - ratio of number of shortest paths between two vertices s,t in an independent set / number of all pair shortest paths. This estimates how two unknown vertices indirectly know each other.

Ranking people in social network can not be done in the same way as documents are ranked because of intrinsic merit anachronism paradox motivated by real world examples in earlier sections - merit precedes prestige while perception/prestige is a measure of merit. Bianconi-Barabasi network model deserves a mention here in which each node has an intrinsic fitness parameter in addition to incoming links. Examples in the reference 438.2 below explain how intrinsic fitness/merit are applied to search engine ranking algorithms. Earlier network models relied on temporal Preferential Attachment alone i.e node having high adjacency at time x are likely to have higher adjacency at time x+deltax. This is a transition from older First-Mover-Advantage (first starter always wins), Rich-Get-Richer to Fit-Get-Richer paradigm. Fitness of a social network profile (twitter/facebook etc.,) is a function of credentials mentioned in the

profile and how fitness is quantified is specific to internal algorithm.

Reference:

438.1 Tight Hamiltonian Cycles in 3-uniform Hypergraphs - [Rodl-Rucinski-Szemeredi] theorem - https://web.cs.wpi.edu/~gsarkozy/Cikkek/Rio08.pdf
438.2 Bianconi-Barabasi Social Network Model - https://en.wikipedia.org/wiki/Bianconi%E2%80%93Barab%C3%A1si_model - "...In 1999, Albert-László Barabási requested his student Bianconi to investigate evolving networks where nodes have a fitness parameter. Barabási was interested in finding out how Google, a latecomer in the search engine market, became a top player. Google’s toppling of previous top search engines went against Barabási's BA model, which states that first mover has an advantage. In the scale-free network if a node appears first it will be most connected because it had the longest time to attract links. Bianconi's work showed that when fitness parameter is present, the "early bird" is not always the winner.[10] Bianconi and Barabási's research showed that fitness is what creates or breaks the hub. Google's superior PageRank algorithm helped them to beat other top players. Later on Facebook came and dethroned Google as Internet's most linked website. In all these cases fitness mattered which was first showed in Bianconi and Barabási's research. In 2001, Ginestra Bianconi and Albert-László Barabási published the model in the Europhysics Letters.[11] In another paper,[12] substituting fitness for energy, nodes for energy level and links for particles, Bianconi and Barabási was able to map the fitness model with Bose gas...."
438.3 Experience versus Talent shapes the web - http://www.pnas.org/content/105/37/13724.full - following simplistic theory provides intuition for it:

Let M be the natural ability/merit/fitness of a vertex and E be the experience.

Rate of change of experience of a social profile vertex is proportional to its intrinsic fitness and present experience.

Previous assumption is a heuristic on real-life experiences - intrinsic fitness is a constant while experience changes with time.

Thus at any time point, experiential learning depends on how naturally meritorious a person is and how much past experience is

helpful in increasing experiential learning. (Might have some theoretical basis in Mistake bound in computational learning theory.

Following formalises how a boolean function is learnt incrementally from a dataset over time by making mistakes and correcting them)

dE/dt = kME
=> dE = kME.dt
=> dE/E = kM.dt
=> log E = kMt + c
=> E = e^(kMt+c)
=> E = e^c * e^(kMt)
=> Experience is exponentially related to talent/merit/fitness
=> People with high natural ability amass same experience in less time compared to people with low merit

At time t=0, E = e^c which is equal to natural ability/merit/fitness M of a social profile vertex in Bianconi-Barabasi network model.
Therefore, E = M*e^(kMt). At t=0, experience equals merit and grows exponentially over time. But how to measure the fitness of a social profile for a human in terms of energy? What is the energy of a human social profile? Low energy syntactically implies low entropy and less chaotic nature of a profile. But real-life social profiles with high entropy do attract huge links. Previous derivation assumes natural talent is defined in terms of numeric quantification of credentials (marks/grades/awards/publications/IQ/achievements) which is insufficient. For human resource analytics, sampling experience at each tenure change (academic/work) gets a snap of experience at that time. Usually experience is measured linearly in time in academics/industries often overlooking merit, but previous equation changes that notion and implies experience exponentially increases with time as a function of intrinsic merit.

Also previous differential equation is a very naive definition of experience vs talent - adding more variables could make it reflect reality.
438.4 Bose-Einstein Condensation in Complex Networks - [Bianconi-Barabasi] - Physical Review Letters - http://barabasi.com/f/91.pdf
438.5 PAFit - Joint estimation of preferential attachment and node fitness in growing complex networks - [Thong Pham, Paul Sheridan & Hidetoshi Shimodaira] - https://www.nature.com/articles/srep32558 - factors determining node fitness in facebook graph - "...A directed edge in the network represents a post from one user to another user's wall. One might speculate that the following factors are important for a user to attract posts to his/her wall: a) How much information about his/her life that he/she publicises: his/her birthday, engagement, promotion, etc. b) How influential and/or authoritative his/her own posts are which call for further discussions from other people; and c) how responsive the user is in responding to existing wall posts. We then can hypothesize fitness ηi to be a combination of these three factors averaged over time ..."
438.6 3-Uniform Hypergraphs/Triple System and Langford Pairs - [Donald Knuth] - The Art of Computer Programming: Combinatorial Algorithms - Volume 4 - Pages 3,33 - Langford Pairs are combinatorial objects created by all possible ways of spacing each of 2n elements in a set having n pairs of identical twins so that each pair is spaced equal to its numeric value. In the context of ThoughtNet, langford pairs can be created by embedding any two pair of class stack vertices having identical number of hyperedges passing through them on a plane, separated by distance equal to number of edges traversing the two hypervertices. 3-Uniform Hypergraphs or Triple System can be represented by an equivalent bipartite graph, incidence matrix and a boolean formula.

---
439. (FEATURE-DONE) Scheduler Analytics for VIRGO Linux Kernel - Commits - 31 July 2017 and 1,3 August 2017

(*) As mentioned in NeuronRain FAQ at http://neuronrain-documentation.readthedocs.io/en/latest/, analytics driven linux kernel scheduler is the ideal application of machine learning within kernel. There are two options to go about:
(*) First is Application Layer Scheduling Analytics in which a python code for example, learns from linux kernel logs, /proc/<pid>/schedstat and /proc/<pid>/sched data of CFS and exports key-value pairs on how to prioritize processes' nice values in /etc/kernel_analytics.conf. This config file is read by VIRGO Linux kernel_analytics module and exported kernelwide. Existing kernel scheduler code might require slight changes to read these variables periodically and change process priorities. There have been past efforts like Scheduler Activations which delink userspace threads and kernel threads by multiplexing M user threads onto N kernel threads and focus is on scheduling the userspace threads on kernel threads. But Linux kernel does not have support for Scheduler Activation and has 1:1 user:kernel thread ratio i.e there is no application thread library having scheduling support. This option would be ideally suited for prioritizing userspace threads by deep-learning from process perf data.
(*) Second is Kernel Layer Scheduling Analytics in which kernel has to make upcalls to userspace machine learning functions. This is quite impractical because kernel overhead and latency of scheduling increases and upcalls are usually discouraged for heavy processing. Moreover this adds a theoretical circularity too - upcall is made on the context of present kernel thread and who decides the priority of the upcall? This requires complete kernel level machine learning implementation (e.g Kernel has an independent replicated implementation of String library and does not depend on userspace GCC libraries). Major hurdle to implement machine learning in kernel is the lack of support for C in machine learning as most of the packages are in C++/Java/Python - e.g. how would Apache Spark cloud processing fit with in kernel?. Computational complexity of this scheduling is O(n*m) where m is the time spent per upcall in process priority classification by machine learning and n is the number of processes. This implies every process would have significant waiting time in queue and

throughput takes a hit. Also this requires significant rewrite of kernel scheduler to setup upcalls, do userspace execve(), redirect standard error/output (fd_install) for logging userspace analytics etc., Userspace upcalls are already supported in VIRGO linux CPU/Memory/FileSystem kernel modules for executing user libraries from kernel. If upcall instead writes to /etc/kernel_analytics.conf file than communicating to kernel (using NetLink for example), this option is same as first one.
(*) Third option is to implement a separate kernel module with access to process waiting queue which loops in a kernel thread and does upcalls to userspace to classify the process priorities and notifies the scheduler on process priorities change asynchronously. But this option is same as first except it is done in kernel with upcalls and overkill while first is a downcall.

First option is chosen for time being because:
	- it is top-down
	- it does not interfere in existing scheduler code flow much
	- minimal code change is required in kernel scheduler e.g as a KConfig build parameter and can be #ifdef-ed
	- key-value pair variable mechanism is quite flexible and broadcast kernelwide
	- any interested module in kernel can make use of them not just scheduler
	- it is asynchronous while kernel layer scheduling is synchronous (scheduler has to wait to know priority for each process from userspace code)
	- second and third options are circuitous solutions compared to first one.
	- Languages other than C are not preferable in kernel (http://harmful.cat-v.org/software/c++/linus)
	- Key-Value pair protocol between userspace and kernelspace effectively tames language barriers between user and kernel spaces indirectly i.e this protocol is atleast as efficient as an inlined kernel code equivalent doing an upcall and computing the same pair
	- Key-Value storage can also exist on a cloud too not necessarily in /etc/kernel_analytics.conf file thus transforming VIRGO kernel into a practical dynamic cloud OS kernel
	- first option is more suited for realtime kernels

```
################################################################################
############
#Scheduler Analytics for Linux Kernel:
#==================================
# Following DeepLearning models learn from process perf variables - CPU, Memory,
Context switches, Number of Threads and Nice data
# for each process id retrieved by psutil process iterator - BackPropagation,
RecurrentLSTM
# RecurrentGRU and ConvolutionNetwork models learn software analytics neural
networks from psutil process performance info
# Key value pairs learnt from these can be read by Linux Kernel Scheduler or
anything else
# and suitably acted upon for changing the process priorities dynamically. Any
kernel module can make an upcall to this userspace
# executable and dump the key-value pairs in /etc/kernel_analytics.conf.
Presently the implementation is quite primitive and
# classifies the output layer of neural network into "Highest, Higher, High,
Normal, Medium, Low, Lower and Lowest" priority classes
# in the format: <pid#deeplearningalgorithm>=<scheduled_priority_class>. Number
of iterations has been set to 10 for all deep learning networks.
################################################################################
############
```

Following is a simulation of how the key-value pair for repriotization from Scheduler Analytics could affect the kernel scheduler:
	(*) Linux kernel scheduler has the notion of jiffies - HZ - set of clock ticks - for time slicing processes
	(*) Each priority class has a red-black binary search tree based queue - deletion is O(1) and insertion is O(logN) after rebalancing such that longest

root to leaf path is not more than twice longer than that of least root to leaf path

(*) On receipt of repriotization analytics, kernel_analytics exports it kernelwide, scheduler reads this <pid>=<new_priority> key-value pair and removes <pid> from <old_priority> red-black tree and inserts in <new_priority> red-black tree.

(*) Thus multiple red-black trees make a trade-off and rebalance periodically with no manual nice interventions - this deletion and insertion is O(logN)

(*) Every repriotization is therefore O(logN).

(*) In realtime bigdata processing of heavy load, priorities can change quite frequently. For m reprioritizations, scheduler incurs O(m*logN) time. But m can have maximum value equal to total number of processes and thus a multiple of N (number of processes per priority class) and thus effective worstcase cost is O(NlogN)

References:

439.1 Scheduler Activations - Effective Kernel Support for the user level management of parallelism - http://dl.acm.org/citation.cfm?id=121151
439.2 Controlling Kernel Thread Scheduling From Userspace - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.605.3571&rep=rep1&type=pdf

---

440. (THEORY) Social Networks, Money Flow Markets, Pricing Equilibrium, Centrality, Intrinsic versus Perceptive Valuations - 1 August 2017 and 11 August 2017

Flow of money in a network of buyers and sellers and pricing have been described in KingCobra design document - https://github.com/shrinivaasanka/kingcobra-github-code/blob/master/KingCobraDesignNotes.txt. Problem of relating intrinsic merit/fitness of an entity to perceptive judgements is essentially reducible to the problem of market equilibrium as below:
In a flow market digraph,there is a directed edge from a vertex b to vertex s of weight p where b is a buyer, s is a seller and p is the price offered by buyer b for s. There can be multiple b(i) edges incoming for a seller s offering different prices and multiple sellers s(i) can have incoming edges from a buyer b. Problem of determining fair price for seller s amounts to finding objective intrinsic merit/fitness of goods/services of s while incoming buyer edges are perceptive subjective market values (auction). Treating this as a link graph and applying Centrality measures like PageRank could identify most valued seller in terms of market perception. Thus the problem of intrinsic merit versus perception is ubiquitous and is not just restricted to internet(text, audio-visuals). Assuming Bose-Einstein Condensation in Flow Markets, fitness/merit is measured by least energy.Note of caution is the term "least energy" has multiple meanings depending on context and cannot be literally taken as least entropy. In flow markets, least energy seller could be one with best marketing abilities, quality, technical knowhow, qualifications etc.,

In money flow markets, each buyer i is allocated a good/service j of x(ij) units with utility u(ij) with price p(j). Objective is to maximimize the money weighted geometric mean of the product of utilities for all buyers. This has to satisfy Karush-Kuhn-Tucker conditions of convex program i.e this money weighted geometric mean is a convex function (line connecting two points on the graph of a function is always above the graph - epigraph is convex) on a convex set (line connecting any two points in the set is within the set) and optimized subject to KKT conditions. Another algorithm is to construct a goods/services - buyers bipartite flow network with a source and sink. This network has edges weighted by prices from source to goods/services vertices and edges weighted by money from buyers vertices to sink. Mincuts of this network determine price equilibrium by maxflow algorithm.

Previous example is mooted to invoke the striking parallels between buyer versus seller price equilibrium and intrinsic merit versus perception equilibrium - Intrinsic value of a good/service is immutable while buyers' price requirements for same good/service vary; Similarly intrinsic merit of an entity is immutable while perception of an entity is subjective. By replacing price with merit in Convex Program and Maxflow algorithms, equilibrium state between intrinsic merit and perceived merit of a social profile vertex can be found. But still this does not lead to absolute intrinsic merit.

Convex Program for Market Equilibrium is defined as below:
Maximize u(ij)^e(i) where buyer i has money e(i) at disposal
subject to:
Total happiness of a customer = Sigma(u(ij)*x(ij)) where u(ij) is the utility buyer i gets from x(ij) units of good j

Following maps a Market to Merit equilibrium:
Intrinsic merit of a (corresponds to Seller) vertex is defined by a feature vector of merit variables. Buyers are perceivers. Each perceiver has a pre-allocated total perceived merit which is divided amongst the perceived vertices. The Convex Program for Equilibrium between intrinsic merit and perceived merit is:
Maximize u(ij)^e(i) where perceiver i has total perceived merit e(i) at disposal and u(ij) is the utility perceiver i gets from merit variable j of a vertex
subject to:
Total happiness of a perceiver = utility of i = Sigma(u(ij)*x(ij)) where u(ij) is the utility perceiver i gets from x(ij) units of merit variable j of a vertex
and p(j) is the equilibrium perceived merit of a variable j
Solving the previous convex program results in an optimal merit vector (set of p(j)'s) and weights of merit variables (x(j)'s) which satisfies both perceiver and perceived.

This equilibrium is closely related to Nash Bargaining Problem which postulates existence of an agreement and disagreement point (a,b) in a set X in R^2 involving two players where a and b are utilities of the two players. Nash function of a point (a,b) is the maximum a*b. In the context of merit equilibrium, the equilibrium point (a,b) is in optimum distance between intrinsic merit and perceived merit and maximizes happiness utility of both perceiver and the perceived.

References:

440.1 Algorithmic Game Theory - Chapter 5 and Chapter 15 - Existence of Flow Markets Equilibrium and Nash Bargaining Problem - Fisher and Eisenberg-Gale Convex Program and Network MaxFlow algorithm - [Tim Roughgarden, Noam Nisan, Eva Tardos, Vijay Vazirani] - http://www.cambridge.org/download_file/909426
440.2 New Convex Program for Fisher Market Equilibria - Convex Program Duality, Fisher Markets, and Nash Social Welfare - [Richard Cole † Nikhil R. Devanur ‡ Vasilis Gkatzelis § Kamal Jain ¶ Tung Mai k Vijay V. Vazirani ∗∗ Sadra Yazdanbod ††] - https://arxiv.org/pdf/1609.06654.pdf - Previous Convex Geometric Mean Program is a Nash Social Welfare Function which is Spending Restricted. This article introduces an improved Convex Program with higher integrality gap. Integrality Gap is the measure of goodness of approximation and is based on Linear Program Relaxation. LP Relaxation waives the 0-1 integer programming to range of reals in [0,1]. Approximate CNFSAT Solver implemented and described earlier in this document is also a relaxation but for system of equations. Integrality Gap of CNF MAXSAT solver found experimentally for few instances is ~100/98 = 1.02 though still it requires further theoretical analysis to upperbound the error.
440.3 Triangle Inequality based Christofides 1/2-approximation Algorithm for Travelling Salesman Problem - Combinatorial Optimization - [Christos H.Papadimitriou,Kenneth Steiglitz] - Page 416

---
441. (FEATURE-DONE) Scheduler Analytics update - commits - 1 August 2017

(*) Updated AsFer Design Document
(*) Updated python-src/software_analytics/DeepLearning_SchedulerAnalytics.py to write in /etc/kernel_analytics.conf
(*) logs committed to testlogs/

---
442. (FEATURE-DONE) Graph Density (Regularity Lemma) and Bose-Einstein Fitness implementations - commits - 1 August 2017

(*) New intrinsic merit measures based on graph density (regularity lemma) and Bose-Einstein intrinsic fitness have been added as functions
and invoked
(*) logs have been committed to testlogs/

---
443. (THEORY) Mistake bound learning(MB), Experience and Intrinsic merit, Social Network Analytics - 2,3,11 August 2017 - related to 368, 438

Mistake bound learning, iteratively refines and converges from an initial hypothesis based on mistakes. PAC learning is also a kind of mistake bound learning. Previously described differential equation relation between experience and intrinsic merit/fitness/natural talent can be theoretically formalised as below:

Let M be the intrinsic merit (correctness) of a boolean function hypothesised in the form of conjunctions and disjunctions. From the derivation
of experience(E) versus merit(M) as function of time(t):

```
      dE/dt = kME
      => dE = kME.dt
      => dE/E = kM.dt
      => log E = kMt + c
      => E = e^(kMt+c)
      => E = M*e^(kMt)
```

If dE/dt is replaced by number of mistakes b1 made and corrected per time unit in MB model and E is replaced by total number of mistakes b2 made so far in hypothesis and corrected (experience is function of this in human context), learner's experiential learning converges in time t to some percentage error and number of mistakes done in time t , b2 = t*b1.

```
      => E = M*e^(kMt)
      => E = M*e^(kM*b2/b1)
```

Previous expression for experience depends both on intrinsic merit and mistakes made which is exactly required in human classification.

Traditionally mistake bounds are used only in learning boolean functions. Generalizing it to any functions and humans creates a generic definition of merit and experience irrespective of the entity involved. Assumption in the previous differential equation is delta increase in experience is proportional to existing experience and merit. Boolean function learning theory does not appear to have the concept of intrinsic merit or how learning accrued so far helps in further learning which is quite essential for classifying humans based on merit+experience in social networks and human resource analytics. In previous expression for mistake bounded experience, number of mistakes made b2 and b1 themselves could be a function of factors like IQ and EQ e.g b1 = f(IQ,EQ) and

b2 = g(IQ,EQ). Then experience becomes:
      => E = M*e^(kM*g(IQ,EQ)/f(IQ,EQ))
If M=h(IQ):
      => E = h(IQ)*e^(k*h(IQ)*g(IQ,EQ)/f(IQ,EQ))

It has to be mentioned here f is nothing but the first temporal derivative of g evaluated at time t i.e f = dg/dt.
      => E = h(IQ)*e^(k*h(IQ)*g(IQ,EQ)/g'(IQ,EQ))

Alternatively, f and g can be substituted directly if known and integrable:
      g'(IQ,EQ) = k*h(IQ)*g(IQ,EQ)
      g'(IQ,EQ) = k*h(IQ)*g(IQ,EQ)
      log g(IQ,EQ) = k*M*t + c
      g(IQ,EQ) = h(IQ)*e^(k*h(IQ)*t)

Curiously, function g which depends on EQ has no equivalent in RHS except time duration t and a constant k. This is probably because time is measured in terms of mistakes in the integral above and EQ is function of mistakes. Intrinsic merit function M does not involve EQ because mistakes alone are usually emotive in the context of social profiles (bounded rationality) and merit must be independent of emotions and time. Previous definition of experience can be used as a ranking metric for human social profiles as below for comparison of two humans after evaluating h, g and g' for some time duration subject to disclaimer that follows:
      g(IQ1,EQ1) = h(IQ1)*e^(k*h(IQ1)*t1)
      g(IQ2,EQ2) = h(IQ2)*e^(k*h(IQ2)*t2)
Disclaimer: Previous model of merit based on IQ and EQ are purely presumptive and disputable because of lack of well-defined Bose-Einstein "least energy" counterpart for human profiles.

This model is quite generic and should apply to any computable function including boolean learnt by mistake bounds. For boolean function learning, this model implies total number of mistakes corrected at time t ("experience") depends exponentially on learning time i.e if the mistake bound learning algorithm is polynomial time in number of variables, number of mistakes corrected is exponential in number of variables. For learning 3CNFs polynomial time mistake bound in RHS implies exponential number of mistakes have been corrected (function g) in LHS but an existing result implies if polynomial time learning is possible for 3CNFs there is no polynomial time algorithm for SAT (section 368) (But, isn't correcting exponential number of mistakes in polynomial time a nemesis of SETH?).

Mistakes in the social networks context are proportional to bounded rationality of the human social profile vertices of the social network. Bounded rationality in turn arises by inability of a human being to make 100% correct decision because of cognitive, psychological, emotional, time limitations. Decision taken is termed satisfactory than optimal and such suboptimal decision is called satisficing. Thus emotional quotient and bounded rationality are related. Measuring emotional quotient (intelligence+emotions) than intelligence quotient (intelligence alone) in social network profiles is equivalent to quantifying the sentiments of social network tweets/wallposts/likes etc.,

Thus following equivalence is conspicuous: Emotional Quotient (proportional to) Bounded Rationality (proportional to) Sentiment polarity of Social profile data e.g obtained by Sentiwordnet sentiment analysis. This is an approximate estimation of mistake-prone-ness and applies to decision errors of voters in Condorcet Jury Theorem group decision circuit too.

BackPropagation is a mistake bound deep-learning algorithm refining a perceptron by trial and error iterations. Previous differential equation for experiential learning is equivalent to a time-evolving recursive mistake correction tree described below:
At time x, tree reaches level l(x). Each node in the tree is a corrected mistake and each mistake corrected at level l(x) is subtree root for n corrected

mistakes in level l(x+1). Thus each level l(x) of the learning tree is the set of mistakes made at time x. In the language of NC-PRAM circuit families, each node is a PRAM processor correcting a mistake. After time duration x, this learning tree has 2^x-1 nodes correcting exponential number of mistakes. In NC notation, if number of mistake correcting processors=2^x=n, depth of the circuit is O(logn) equal to learning time x. This is a hypothetical parallel learning algorithm in non-uniform NC. This recursive mistake correction tree is nothing but a translation of differential equation dE/dt=kME. Outdegree of each vertex (number of future mistakes corrected by a PRAM processor) is determined by merit M, and delta increase in learning is number of mistakes corrected at level l(x) after time x = M*2^(l(x)). This should apply to backpropagation weight update iterations because each error term is a mistake. Perceptrons are in TC=AC=NC which is same as previous PRAM recursive learning.

Caveat is the previous recursive mistake correction tree is not traditional bottom-up evaluation circuit but is top-down based on heuristic: mistake corrected at time x helps correcting n number of mistakes at time x+deltax inspired by a differential equation for experiential learning.It makes sense to define "least energy/fitness" of a social network profile vertex to be equal to function g above thereby sufficiently weighing both merit and experiential learning. Social networks fall into two classes: Professional(e.g LinkedIn/Stackoverflow/Quora/Academic Citation Networks) and Personal(e.g Facebook/Twitter/Instagram/Whatsapp). Aforementioned definition of least energy as function of merit and experience applies only to Professional social networks. Stackoverflow is a special case of professional social profile - users up or down vote answers to a question which is fame-driven and not merit-driven. Least energy in personal social networks is totally unrelated to merit and experience and should more or less quantify definition of fitness in Facebook mentioned in 438.5 - profile having high nett positive emotive sentiment evaluated from wallposts/tweets/likes/replies/retweets. Personal social networks are replete with spam,chutzpah,trolls,expletives,emojis,satires etc., requiring parsing/filtering in Sentiment Analysis. Reference 443.4 analyzes the Merit Versus Fame problem in academic citations which belongs to Professional social network category. In essence ranking humans on intrinsic merit is two-fold: 1) Merit-Experience ranking 2) Personal Emotive ranking. These two rankings need not coincide and define intrinsic fitness/least energy of a person in two disjoint contexts. In literature, fitness/least energy of a social profile vertex is not a point value but a probability distribution i.e probability of a vertex/person having an intrinsic merit random variable value. Let us assume that merit is defined as a vector of n dimensions as:

M=<m1,m2,m3,...,mn> where each mi is the weight for a merit variable

For example, Probability distribution for merit/least energy/fitness can be defined as:

Pr(M=mi) = mi/|M| where |M| is the L1 norm of M

Sigma(Pr(M=mi)) = (m1 + m2 + m3 + ... mn) / |M| = 1

Continuous Probability distribution is the polynomial passing through above merit weight points.

In generic sense, previous recursive mistake correction tree need not be a binary tree but a graph: At any time point t, number of mistakes corrected by a social profile vertex thus far forms an evolving triangular region of experience E, and each corrected mistake vertex in this triangle causes some mistake to be corrected in a future trapezoidal region dE in time interval (t,t+dt) adjoining this triangle E weighted by intrinsic merit M. This intuitively captures the equation E = M*e^(kMt) (dE/dt = kME) and the paradigm "experience = learning from mistakes"

References:

443.1 Mistake bound learning - https://www.cs.utexas.edu/~klivans/lec1.ps - In this teacher/learner model, learner iteratively refines hypothesis based on mistakes pointed out by teacher and number of mistakes are polynomially bounded.
443.2 Learning quickly when irrelevant attributes abound - [Nick Littlestone] - http://citeseerx.ist.psu.edu/viewdoc/download?

doi=10.1.1.130.9013&rep=rep1&type=pdf
443.3 Bounded Rationality and Satisficing - https://en.wikipedia.org/wiki/Bounded_rationality
443.4 Multiplex Networks with Intrinsic Fitness: Modeling the Merit-Fame Interplay via Latent Layers - [Babak Fotouhi and Naghmeh Momeni] - https://arxiv.org/pdf/1506.04189.pdf - An example of intrinsic merit versus fame in Academeic Citations graph - "...We consider a growing directed multiplex network that comprises two layers. Each node is assigned an intrinsic fitness, which models its quality. The fitness of a node never changes. Each node belongs to two layers: a merit layer and a fame layer. In the former, fitness values are the sole drivers of the growth mechanism. In the fame layer, attachment is preferential, that is, the probability that a node receives a link from a newcomer is proportional to the total degree of that node, i.e., the sum of its degrees in both layers. For example, in the case of citation networks, the interpretation of the model is as follows. Two distinct types of citations can be discerned. The first type—the meritocratic type—is when a scholar reads a paper, and cites it because of its content (a citation which would be given regardless of the number of citations that paper already has). Another type of citation is what we call fame-driven. A paper can become trendy, or well-known in some literature (particularly true for seminal papers which initiate a new subfield), and many citations that it receives would be solely due to its fame—i.e., current number of citations,..."
443.5 Relation between Intrinsic Merit of two vertices and probability of creating link between them - Page 27 - Equation 3.1 - http://bura.brunel.ac.uk/bitstream/2438/10345/1/FulltextThesis.pdf - a new vertex is assumed to have an apriori fitness and probability of establishing connection between a new node and an existing node in social networks is weighted average of probability of new node creation and edge creation between new node and existing node. From this edge probability an a posteriori fitness distribution can be derived for all vertices after a stationary markov chain random walk.
443.6 First to Market is not Everything: an Analysis of Preferential Attachment with Fitness - [Christian Borgs, Jennifer Chayes, Constantinos Daskalakis ∗ Sebastien Roch †] - https://www.microsoft.com/en-us/research/wp-content/uploads/2016/11/First-to-Market-is-not-Everything-an-Analysis-of-Preferential-Attachment-with-Fitness.pdf - Polya Urn model of preferential attachment and Power law relation between degree of a vertex in networks and its intrinsic fitness - "...A by-product of our technique is a precise characterization of the vertex dynamics under preferential attachment with fitness. More specifically, if a vertex v has fitness f, then our analysis implies that its degree dv(t) at time t scales as dv(t) ~ t^(c*f) , (1) where c is a global constant determined by the fitness distribution. (The details are omitted from this extended abstract.) Hence, the logarithm of the degree of the vertices directly reflects their quality. We note in passing that this could suggest new directions in the design of ranking or recommendation algorithms...". Roughly this implies, fame in social networks is exponentially related to merit. Previous experience-versus-merit temporal identity strikingly coincides with this. Fame=dv(t)=t^(cf) can be rewritten as dv(t) = e^(cf*logt) which when constrasted against Experience E = M*e^(kMt), yields E=f*e^(cft) setting fitness f= Merit M and k=c. Similarity between degree of a vertex and experience by matching LHS of these 2 expressions is obvious i.e experience of vertex is proportional to its adjacency. This is intuitive because experience is gained after interacting with other vertices and correcting mistakes while judging them. Alternatively, Experiential Learning E can be defined in terms of time evolving degree of vertex :

log(dv(t)) = cf*logt
f=log(dv(t))/clog(t)
Since f=M, E = M*e^(kMt) = log(dv(t)) * e^(klog(dv(t))*t/clogt) / clogt

443.7 Degree distribution and Intrinsic Fitness - https://www.cs.upc.edu/~CSN/slides/08network_dynamics.pdf - "... Consider f (xi , xj) = (xi*xj)/x*M)^MM where x(M) is the largest value of x in the network. Then the mean degree of a node of fitness x is k(x) = nx/x(M)^2 * integral(0,∞) [yρ(y)dy] = N<x>*x/x(M)^2 ... " - Expected Degree of vertex and Fitness of vertex are thus related - Expectation of Fitness PDF is obtained by integral.

443.8 Ghadge et al Model of Intrinsic Fitness of a vertex - https://www.cs.umb.edu/~duc/publications/papers/ijpeds10.pdf - A statistical construction of power-law networks - [Shilpa Ghadgea, Timothy Killingback, Bala Sundaram and Duc A. Trana] - "... To motivate the definition of our procedure, we observe that in most real-world networks the nodes will have an attribute or an associated quantity that represents, in some way, the likelihood that other nodes in the network will connect to them. For example, in a citation network (see, e.g. [22]), the different nodes (i.e. papers) will have different propensities to attract links (i.e. citations). The various factors that contribute to the likelihood of a paper being cited could include the prominence of the author(s), the importance of the journal in which it is published, the apparent scientific merit of the work, the timeliness of the ideas contained in the paper, etc. Moreover, it is plausible that the overall quantity that determines the propensity of a paper to be cited depends essentially multiplicatively on such various factors. The multiplicative nature is likely in this case since if one or two of the factors happen to be very small, then the overall likelihood of a paper being cited is often also small, even when other factors are not small. The case of Mendel’s work on genetics constitutes an exemplar of this – an unknown author and an obscure journal were enough to bury a fundamentally important scientific paper. Motivated by this and the other examples, we consider it reasonable that in many complex networks each node will have associated to it a quantity, which represents the property of the node to attract links, and this quantity will be formed multiplicatively from a number of factors. That is, to any node i in the network there is associated a non-negative real number Fi , which is called the fitness of node i, and which is of the form Fi = product(fl) ,l=1 to L where each fl is non-negative and real. ..." - this multiplicative intrinsic merit is mapped to summation of logarithms of merit variables. Central Limit Theorem implies summation of random variables converge to a Gaussian irrespective of values of merit variables. Merit variables are dimesional projections of merit vector defined above. CLT also implies merit (social or linguistic) converges to a Gaussian as it were in real life - tails are entities with highest and lowest merit and modal is of maximum merit. Interestingly, this prohibits multimodal. It has to be noted here Recursive Gloss Overlap algorithm computes the intrinsic merit of a text as a product of graph complexity merit variables (Section 4.6 of https://arxiv.org/abs/1006.4458, Section 5.6 of https://tac.nist.gov//publications/2010/participant.papers/CMI_IIT.proceedings.pdf). Recursive Gloss Overlap graph intrinsic merit is indeed a mapping of social network intrinsic fitness to graph representation of texts - socially networked humans are replaced by hyperlinked texts.

443.9 Vertex Intrinsic Fitness in Social Networks - [Servedio,Caldarelli] - https://arxiv.org/pdf/cond-mat/0309659.pdf

443.10 Bibliography listed in Social Networking: Mining, Visualization and Security - [Mrutyunjaya Panda, Satchidanda Dehuri, Gi-Nam Wang] - https://books.google.co.in/books?id=5-W5BQAAQBAJ&pg=PA42&lpg=PA42&dq=intrinsic+fitness+social+network&source=bl&ots=KD2lkaUq9Q&sig=6WmEYptNmnz4X2Y4qrQHJIRpsHk&hl=en&sa=X&ved=0ahUKEwi177OJ7dHWAhUBtY8KHZ9hBPcQ6AEITDAH#v=onepage&q=intrinsic%20fitness%20social%20network&f=false

443.11 Vertex Fitness defined in terms of Unfitness - Network growth models: A behavioural basis for attachment proportional to fitness - [Michael Bell* , Supun Perera, Mahendrarajah Piraveenan, Michiel Bliemer, Tanya Latty, Chris Reid] - https://arxiv.org/pdf/1702.04046.pdf

443.12 A κ-deformed Model of Growing Complex Networks with Fitness - [Massimo Stella , Markus Brede] - https://arxiv.org/pdf/1404.3276.pdf - Generalizes the Bianconi-Barabasi Bose-Einstein Condensation Least Energy Intrinsic Fitness of a vertex - https://arxiv.org/pdf/1404.3276.pdf

443.13 Various Intrinsic Fitness Models - Barabasi-Albert, Barabasi-Bianconi, Ghadge Log Normal Fitness Attachment, Caldarelli Fitness - https://www.cs.umb.edu/~duc/www/research/publications/papers/bookchapter_fitness_models.pdf - Of these Ghadge LNFA multiplicative intrinsic fitness has close resemblance to Recursive Gloss Overlap Intrinsic Merit and Recursive Lambda Function Growth Graph Tensor Neuron Network Intrinsic Merit which are both multiplicative and additive.

---
444. (FEATURE-DONE) Scheduler Analytics Update - Commits - 10 August 2017

(*) There is a problem in defining expected priorities in BackPropagation Neural Network output layer. This is necessary for
weight update to happen iteratively and to quantify error between expected and actual neural network output
(*) Output layer of BackPropagation is an indicator of process priority learnt by the perceptron.
(*) Previously expected priorities were hardcoded. This has been changed as below:
	(*) Set of processes currently running are clustered and stored in an input JSON file
	(*) This JSON dictionary has key-value pairs mapping a process class to its apriori expected priority
	(*) Scheduler Analytics loads this JSON and looks up the expected priority for each process executable by its process class
	(*) Presently process class is just a substring of the executable name
	(*) BackPropagation does string match of this process class to executable name and finds the expected priority from JSON dict
	(*) BackPropagation iteration updates the weights of all 3 layers
(*) Defining apriori class of a process is subjective and instead of substring of executable, an independent clustering algorithm having a
suitable distance function can be used to write the JSON classification of processes. For example, unix process groups are numerical classes of
 processes.
(*) Previous generation of JSON classes for processes and backpropagation weight update iteration is the first phase of analytics.
(*) In the second phase, weight-updated BackPropagation Neural Network takes as input runtime process statistics (CPU, Memory and Context Switches)
to predict the actual priority of a process belonging to the apriori class.
(*) This is a depth 2 tree evaluation:
	Static - apriori expected class of a process and BackPropagation iteration
	Dynamic - fine grained priority of a process within the previous static class by evaluating the multilayer neural network

Presently, input variables are per process CPU percentage, Memory percentage and ratio of involuntary context switches to total context switches. Involuntary context switches are usually number of pre-emptions by the scheduler on time-slice expiry and Voluntary context switches are number of system call and I/O pre-emptions.

445.(FEATURE-DONE) Celestial Pattern Mining Update - specific to NeuronRain Research in SourceForge repositories - recreated Sequence Mined Rules - Commits - 22 August 2017

(*) Added a documentation text file
asfer-docs/NeuronRain_Research_Celestial_Pattern_Mining.txt describing how astronomical pattern
mining is done and code involved therein.

(*) In python-src/autogen_classifier_dataset/MaitreyaToEncHoroClassified.py upgraded Maitreya's Dreams (http://www.saravali.de/)
ephemeris text client version to 7.1.1

(*) Created separate string encoded astronomical data files for Earthquakes(USGS 100 year earthquake dataset) and
Storms(NOAA 100 year HURDAT2 dataset) Datasets from

python-src/autogen_classifier_dataset/MaitreyaToEncHoroClassified.py
(zodiacal does not have ascendant, while ascrelative is rotated string relative to ascendant. Ascendant is defined astronomically as the
 degree of the zodiac sign rising at the time of an event in eastern horizon)

(*) Executed python-src/SequenceMining.py on these 2 datasets. The results have been written to python-src/MinedClassAssociationRules.txt

(*) python-src/autogen_classifier_dataset/asfer_dataset_segregator.sh has updated asfer relative location

---
446. (FEATURE-DONE) Celestial Data Pattern Mining - Updates to autogen_classifier_dataset pre-processing and classification of datasets - 23 August 2017

(*) An 8 year old lurking bug in cpp-src/NaiveBayesClassifier.cpp has been resolved.It was causing a null class to be returned after
bayesian argmax() computation.
(*) Lot of new articles on earthquakes and hurricanes have been added as training dataset for NaiveBayesian Classifier.These articles also
have lot of research information on predicting hurricanes and earthquakes
(*) python-src/autogen_classifier_dataset/AsferClassifierPreproc.py has been updated to include new articles in training data for NaiveBayesian
(*) words.txt,word-frequency.txt,training-set.txt,topics.txt,test-set.txt have been re-created by executing
python-src/autogen_classifier_dataset/AsferClassifierPreproc.py
(*) Datasets are classified into
python-src/autogen_classifier_dataset/EventClassDataSet_Earthquakes.txt and
python-src/autogen_classifier_dataset/EventClassDataSet_Storms.txt by executing asfer_dataset_segregator.sh

---
447. (FEATURE-DONE) Updated Ephemeris Search Script - Commits - 24 August 2017

Updated mined class association rule parsing in
python-src/MaitreyaEncHoro_RuleSearch.py

---
448. (FEATURE-DONE) Approximate CNFSAT Solver update - SciPy Sparse - 28 August 2017

(*) imported scipy.sparse.linalg least squares function lsqr().
(*) Number of clauses and variables set to 18 each.
(*) After 673 iterations of Random 3CNFs following satisfied clauses data were observed:
Percentage of CNFs satisfied so far: 58.7537091988
Average Percentage of Clauses per CNF satisfied: 97.0161556215

From LLL, If p < 1/[e(d+1)] is the probability of an event (not satisfying a clause) :
Probability that clause is satisfied: 1-p > (1-1/[e(d+1)])
Probability that all of the clauses are satisfied > (1-1/[e(d+1)])^(d+1) (for d+1 clauses)
Probability that some of the clauses are satisfied < 1-(1-1/[e(d+1)])^(d+1)

If all clauses overlap:
In previous example d+1 = 18 and thus probability that all of the clauses are

satisified for any random CNF is :
(1-1/[e(18)])^(18) > ~68.96%
and probability that some but not all of the clauses being satisified for any random CNF is:
1-(1-1/[e(18)])^(18) < ~31.043%

If none of the clauses overlap:
d=0 and probability that all the clauses are satisfied (1-1/e) > ~63.21%
and probability that some but not all of the clauses being satisified for any random CNF is < ~36.79%

---
449. (THEORY) Analysis of Error Upperbound for Approximate CNFSAT Solver based on Lovasz Local Lemma clause dependency digraph - 29 August 2017 ,30 August 2017

Probability of a clause being not satisfied from Lovasz Local Lemma:
      p < 1/(e(d+1))
Probability of a clause being satisfied:
      1-p > 1 - 1/(e(d+1))
Probability of all n clauses being satisfied (ExactSAT):
      (1-p)^n > [1 - 1/(e(d+1))]^n
Probability of some or none of the clauses being satisfied:
      1-(1-p)^n < 1 - [1 - 1/(e(d+1))]^n
If there are no overlaps, d=0:
      Probability of all n clauses being satisfied = (1-p)^n > [1 - 1/e]^n = [1 - 1/e]^n
      Probability of some or none of the clauses being satisfied = 1 - (1-p)^n < 1 - [1 - 1/e]^n

Probability of atleast k of n clauses being satisfied:
      Pr[#SATclauses >= k] = Pr[#SATclauses = k] + Pr[#SATclauses = k+1] + Pr[#SATclauses = k+2] + ... + Pr[#SATclauses = n]

      = (1-p)^k + (1-p)^(k+1) + ... + (1-p)^n

if q = 1-p:
      = q^k + q^(k+1) + ... + q^n
      = q^k * (1 + q + q^2 + ... + q^(n-k)) [Geometric series]
      = q^k (1 - q^(n-k+1)) / (1-q)
But from LLL, p < 1/e(d+1)
      q = 1 - p > (ed+e-1)/(ed+e)

Substituting for q:
      [(ed+e-1)/(ed+e)]^k [1 - [(ed+e-1)/(ed+e)]^(n-k+1)]
      ----
            1 - [(ed+e-1)/(ed+e)]

```
#####################################################################
#Probability of atleast k of n clauses being satisfied:          #
#     [ed+e-1]^k [(ed+e)^(n-k+1) - [ed+e-1]^(n-k+1)]       #
#     ---------------------------------------              #
#           [(ed+e)^n]                          #
#####################################################################
```

if k=n:
      (ed+e-1)^n / (ed+e)^n

If overlap of variables across the clauses is huge, ed+e-1 ~ ed+e
Maximum overlap d is (n-1) i.e a clause has common variables across all other clauses

When d is maximum:

Lt (d -> n-1, n -> infinity) ((ed+e-1)/(ed+e))^n
= [((e(n-1)+e-1)/(e(n-1) + e)]^n
= [(en-e+e-1)/(en-e+e)]^n
= [(en-1)/(en)]^n
= [(1-1/ne)]^n
= [1 - 0.3678/n]^n
For infinite n:
= e^(-0.3678)
Pr[#SATclauses = n] = 69.22%

When overlap is absent at d=0:

Pr[#SATclauses = n] = (e-1)^n/e^n = (0.6321)^n -> 0 for huge n

When d=1 (minimum overlap):

Pr[#SATclauses = n] = [2e-1]^n / [2e]^n = [1-0.5/e]^n = (0.81606)^n -> 0 for huge n

=> When the number of variables overlapping across clauses tends to a huge number almost equal to number of clauses, the system of linear equations obtained by relaxing each of the clauses are solved by least squares and probability of all clauses being satisfied tends to 69.22%. High overlap of variables across clauses is the most probable occurrence in real world SAT solvers. If the overlap as random variable is uniformly distributed:
Expected value of d = 1/(n-1) * (n-1)n/2 = n/2

For mean overlap of n/2 variables across clauses:
```
#####################################################################
#Probability of atleast k of n clauses being satisfied:          #
#     [en/2+e-1]^k [(en/2+e)^(n-k+1) - [en/2+e-1]^(n-k+1)] #
#     ---------------------------------------              #
#            [(en/2+e)^n]                                  #
#####################################################################
```

When k=n:
[en/2+e-1]^n

[en/2+e]^n

= [1 - 0.7357888/(n+1)]^n > [1 - 0.7357888/n]^n
= Lt (n->infinity) [1 - 0.735758882/n)]^n = e^-0.73578882
=> Pr[#SATclauses = n] > 47.91%

=> When the number of variables overlapping across clauses are uniformly distributed, Probability of all clauses being satisfied per random 3CNF is lowerbounded as 47.91% when number of clauses tend to infinity.
=> Caveat: This is an average case analysis of overlaps and best case analysis for number of clauses satisfied. This implies SAT is approximately solvable in polynomial time asymptotically in infinite if the overlaps are uniformly distributed. This is almost an 1/2-approximation in average overlap setting, similar to Christofides algorithm for TSP. This is not an abnormal appoximation and it does not outrageously contradict witnesses towards P != NP described earlier in this document.

Correction to Geometric distribution LLL estimate:

Previous bound assumes Pr[#SATclauses >= k] is geoemtric distribution. If k satisfying clauses are chosen from total n clauses in 3CNF, it is a tighter binomial distribution and not geometric.

=> Pr[#SATclauses >= k] = summation(nCl*(1-p)^l*(p)^(n-l)), l=k,k+1,k+2,...,n

If k=n/2, this summation is exactly same as Condorcet Jury Theorem where each clause in the 3CNF is a voter = probability of majority of the clauses are satisfied. Each satisfying clause is +1 vote and each rejecting clause is -1 vote. Condorcet Jury Theorem thus applies directly and Pr[#SATclauses >= k] tends to 1 if p > 1/2 and tends to 0 if p < 1/2. For p=0.5, Pr[#SATclauses >= k] = 0.5. From LLL, p=0.5 implies e*0.5*(d+1) > 1 (or) (d+1) > 2/e. For arbitrary values of k, computing closed form of binomial coefficient sum for Pr[#SATclauses >= k] requires hypergeometric functions.

When k=n:

Pr[#SATclauses = n] = (1-p)^n
From LLL, ep(d+1) > 1 => p > 1/e(d+1)
1-p < 1 - 1/e(d+1)
1-p < 1 - 0.36788/(d+1)
(1-p)^n < (1 - 0.36788/(d+1))^n

d=0, no overlaps:

Pr[#SATclauses=n] = (1-p)^n < (0.6321)^n which tends to 0 for huge number of clauses n.

d=n-1, maximum overlaps:

Pr[#SATclauses=n] = (1-p)^n < (1 - 0.36788/n)^n
But Limit(n->infinity) (1 - 0.36788/n)^n = e^(-0.36788) = 0.692200241
Pr[#SATclauses=n] < 69.22% for huge n.

d=n/2, expected overlaps in uniform distribution:

Pr[#SATclauses=n] = (1 - 1/(e(n/2+1)))^n = (1 - 0.735758882/(n+2))^n
0.735758882/(n+2) < 0.735758882/n
(1 - 0.735758882/(n+2))^n > (1 - 0.735758882/n)^n
As n->infinity, (1-0.73578882/(n+2)) > e^(-0.73578882) ~ 47.91%
=> Pr[#SATclauses=n] > 47.91%

=> 97-98% satisfied clauses found in few hundred iterations above for upto 20 clauses and 20 variables should tend asymptotically to 69.22% if there are high overlaps, high number of clauses and variables. For average number of overlaps, Pr[#SATclauses=n] > 47.91% and does not rule out convergence to 98%. This can be substituted for k for probability that 98% clauses are satisfied as below:
Pr[#SATclauses >= 0.98n] = summation(nCl*(1-p)^l*(p)^(n-l)), l=k,k+1,k+2,...,n
This is tail bound for binomial distribution which is defined as:
Pr[X >= k] <= e^(-nD(k/n || p))
where relative entropy D(k/n || p) = k/n log (k/np) + (1-k/n)log ((1-k/n)/(1-p))
But k/n = 0.98n/n = 0.98
=> D(k/n || p) = k/n log (k/np) + (1-k/n)log ((1-k/n)/(1-p)) = 0.98log(0.98/p) + 0.02log(0.02/(1-p))
=> Pr[X >= 0.98n] <= e^(-n*(0.98log(0.98/p) + 0.02log(0.02/(1-p))))
From LLL, p < 1/e(d+1). Substituting for p:
=> Pr[X >= 0.98n] <= e^(-n*(0.98log(0.98e(d+1)) + 0.02log(0.02(ed+e)/(ed+e-1))))

Lt(n->infinity) e^(-n*(0.98log(0.98e(d+1)) + 0.02log(0.02(ed+e)/(ed+e-1)))) = 0

Pr(#SATclauses <= n) <= e^(-(np-n)^2/2pn) by Chernoff bounds. From LLL, p < 1/e(d+1). For average number of overlaps d=n/2, Chernoff bound reduces to: Pr(#SATclauses <= n) <= e^(-1/e) ~ 69.22% which coincides with maximum overlaps bound above.

Previous imply approximate CNF SAT solver by solving system of linear equations (= clauses relaxed to reals from binary) is a 0.6922-approximation.

References:

449.1 Sum of Binomial Coefficients -
http://web.maths.unsw.edu.au/~mikeh/webpapers/paper87.pdf - "... “For years [before working with
George E. Andrews in 1973] I had been trying to point out that the rather confused world of binomial coefficient summations is best understood in the language of hypergeometric series identities. Time and again I would find first–rate mathematicians who had never heard of this insight and who would waste considerable time proving some apparently new binomial coefficient summation which almost always turned out to be a special case of one of a handful of classical hypergeometric identities. ...”
449.2 Binomial Distribution Tail Bounds -
https://en.wikipedia.org/wiki/Binomial_distribution#Tail_bounds

---
450.(THEORY) Analysis of Error Upperbound for Approximate CNFSAT Solver based on Lovasz Local Lemma clause dependency digraph - continued - 1 September 2017

(*) Least Squares lsqr() function has been replaced by recent lsmr() least square function mentioned to be faster.
(*) For 10 variables and 10 clauses for 54068 random 3CNF iterations following were the percentage of satisfied CNFs and clauses satisfied
per CNF:
Percentage of CNFs satisfied so far: 79.6722706172
Average Percentage of Clauses per CNF satisfied: 97.6276609517
(*) Again the percentage of clauses satisfied per random 3CNF converges to ~98% after ~54000 random 3CNFs.
(*) The clauses are created by randomly choosing each literal and its negation independent of any other literal or clause and thus are identically, independently distributed. CNFs can repeat since this is not a permutation.
(*) Repetitive convergence of number of clauses satisfied per CNF to 98% is mysterious in varied clause-variable combinations.
(*) Previous error bound analysis does not assume least squares and is based only on LLL. But Least Squares is about sum of squares of errors minimization i.e Error^2 = |Ax-b|^2 is minimized e.g A'Ax=A'b, partial first derivative of error set to 0. System of equations translated from each 3CNF is overdetermined or underdetermined mostly i.e number of variables and number of clauses are not equal.
(*) Motivation for least squares for boolean CNF is the relaxation achieved: An example assignment for a literal, 0.88 is inclined towards boolean 1 and 0.02 is inclined towards boolean 0.
(*) Convergence to 98% implies probability of average number of clauses satisfied per 3CNF > 0.98n must be almost 0.
(*) Following Binomial Tail Bound:
Pr[#SATclauses >= 0.98n] <= e^(-n*(0.98log(0.98e(d+1)) + 0.02log(0.02(ed+e)/(ed+e-1)))) is maximized when there are no overlaps - d=0.
Pr[#SATclauses >= 0.98n] <= e^(-n*(0.98log(0.98e) + 0.02log(0.02(e)/(e-1))))
Pr[#SATclauses >= 0.98n] <= e^(-n*(0.41701 - 0.02999)
Pr[#SATclauses >= 0.98n] <= e^(-n*0.38701) ~ 0 for large n.

451.(THEORY) PCP theorem, Hardness of Approximation, Least Squares SAT solver and Semidefinite Programming - related to 450 - 2 September 2017
and 3 September 2017

Probabilistically Checkable Proof theorem implies it is NP-hard to find satisfiable instances or 7/8+epsilon satisfiable instance to 3SAT. There are Semidefinite Programming optimization algorithms for MAX3SAT which replace the

clauses by an arithmetized polynomial for each literal. Sum of these polynomials or Sum of Squares (SOS) of these polynomials is optimized by Semidefinite decomposition. Positive literal x is replaced by (1-x)/2 and negation !x by (1+x)/2. Literal polynomials within each clause are multiplied. This is a polynomial over reals and is decomposed as XSX' > 0 i.e positive semi-definite. Values of these Sum and Sum of Squares polynomials (F) for an assignment to CNF (a) are equal to number of clauses violating an assignment. Semidefinite Program maximizes/minimizes the difference F(a)-e (thus maximizing or minimizing the number of violating clauses) where e is value of F for assignment a. In this context, the striking 98% convergence for percentage of clauses satisfiedfor multiple clause-variable combinations for least squares/LSMR solver gains importance. It implies 7/8+epsilon=0.98 (or 0.875 + 0.105 = 0.98) satisfiable clauses per CNF if proved for all permutations of variables-clauses-overlaps (and that is a big if) implying P=NP. Atleast a convergence to 0.875 if not 0.98 would suffice.

References:

451.1 PCP and hardness of approximation - algorithm satisfying 7/8+delta clauses - https://cs.stackexchange.com/questions/71040/hardness-of-approximation-of-max-3sat
451.2 SDP for MAX3SAT - http://www.isa.ewi.tudelft.nl/~heule/publications/sum_of_squares.pdf - Maximum number of clauses violating an assignment and Semidefinite Programming
451.3 Binary Solutions to System of Linear Equations - https://arxiv.org/pdf/1101.3056.pdf
451.4 Karloff-Zwick 7/8 algorithm for MAX3SAT - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.1351 - A 7/8-approximation algorithm for MAXSAT? - This is randomized polynomial time algorithm and satisfies an expected >= (7/8)n clauses.
451.5 Inapproximability results - [Johan Hastad] - https://www.nada.kth.se/~johanh/optimalinap.pdf - Assuming P != NP, MAX3SAT can not have polynomial time algorithm satisfying > (7/8)n clauses.
451.6 Gauss-Jordan Elimination For System of Linear Equations, Origin of Least Squares, Linear Programs for Inequalities - http://www-personal.umich.edu/~murty/LPINFORMStutorial2.pdf - previous CNF SAT Solver has only equalities.
451.7 Randomized Rounding for LP relaxation - https://en.wikipedia.org/wiki/Randomized_rounding - Randomized Rounding involves 3 steps - formulating a Linear Program for a 3SAT, solving it fractionally and rounding the fraction to nearest integer. Previous CNF SAT Solver maps this rounding process to system of linear equations - formulate each 3SAT as system of linear equations, one equation per clause (AX=B, B is unit vector of all 1s), get fractional values for vector X from Least Squares, Round the fraction to 0 or 1 whichever is nearest.

452.(THEORY) Approximate CNF 3SAT Solver - Least Squares Error Rounding Analysis - 9 September 2017, 11 September 2017, 14 September 2017

Notation:
A' = Transpose of A
A^-1 = Inverse of A
E(A) = Expectation of A
p(A) = probability of A

Rounding in least squares solution to system of equations per CNF 3SAT fails if and only if binary assignment obtained by rounding does not satisfy the CNF while fractional assignment satisfies the system of equations for CNF. Least Square error is minimized by setting first partial derivative of error to zero:

$$A'Ax = A'b$$
$$\Rightarrow x = (A'A)^{-1}A'b$$

Number of satisfied clauses in least squares is = tr(A(A'A)^-1*A'*b) because Ax-b is minimized.
Maximum value of tr(A(A'A)^-1*A'*b) is the maximum number of satisfied clauses
if tr(A(A'A)^-1*A'*b) > 7/8*number_of_clauses, then P=NP by PCP theorem.

Each matrix A corresponding to a CNF is a random variable - A is a random matrix (not necessarily square).
Expected value of A(A'A)^-1*A'*b where random matrix A belongs to some probability distribution = E(A(A'A)^-1*A'*b)

E(A(A'A)^-1*A'*b) = b*E(A)*E((A'A)^-1)*E(A')

E(A(A'A)^-1*A'*b) = b*E(A)*E((A'A)^-1)*E(A) because A and A' are bijections.

E(A(A'A)^-1*A'*b) = b*E(A)*E(A'A)*E(A) because A'A and (A'A)^-1 are bijections.

E(A(A'A)^-1*A'*b) = b*E(A)*E(A)*E(A')*E(A) by definition of product of expectations

E(A(A'A)^-1*A'*b) = b*(E(A))^4

E(A(A'A)^-1*A'*b) = b*(A*p(A))^4

Each random matrix for a random 3CNF has nm entries where n is number of variables and m is number of clauses. Each of nm literals is either a variable or its negation.

Probability of choosing a variable or its negation = 1/(2n) for n variables + n negations.
Probability of a random matrix A of mn entries = (1/(2n))^nm

=> E(A(A'A)^-1*A'*b) = b(sigma(A(1/2n)^nm)^4 in uniform distribution.
= b(1/(2n)^nm * sigma(A))^4 which sums up all 2^nm possible boolean matrices.
= b(1/(2n)^mn * [matrix of 2^(nm-1) for all entries])^4
= b * [matrix of 2^(mn-1)/(2n)^nm for all entries]^4

==========
Tighter estimate for probability of a random 3CNF:
==========

Per clause 3 literals have to be chosen from 2n literals = 2nC3(1/2n)^3(1/2n)^(2n-3)

For m clauses each being independent, Probability of a random 3CNF
= (2nC3(1/2n)^3(1/2n)^(2n-3))^m

==========

E(A(A'A)^-1*A'*b) = b * [matrix of 2^(mn-1)/(2n)^nm for all entries]^4 (1,m) * (m*n, n*m, m*n, n*m)
= b * [matrix of n*2^(2nm-2)/(2n)^2mn for all entries]^2 (1,m) * (m*m, m*m)
= b * [matrix of m*n^2*2^(2nm-2)/(2n)^2mn for all entries] (1,m) * (m*m)
= [matrix of m^2*n^2*2^(2nm-2)/(2n)^2mn for all entries] (1,m)

Trace of this expected matrix is the expected number of clauses that can be satisfied by least squares in uniform distribution:
= m^3*n^2*2^(2nm-2)/(2n)^2mn

if number of clauses to be satisfied <= m, m^3*n^2*2^(2nm-2) <= m*(2n)^2mn
=> m^2*n^2*2^(2nm-2) <= (2n)^2nm

which is obvious because (2n)^(2mn) grows faster.
=> m^2*n^2*2^2nm <= 4 * 2^2nm * n^2nm
=> m^2*n^2 <= 4 * n^2nm

For all m clauses to be satisfied, this must be an equality:
m^2 = 4 * n^2(nm-1)

if m=n (number of clauses = number of variables), all clauses are satisfied if:
n^2 = 4 * n^(n^2-1)
n^2/4 = n^(n^2-1)
=> log (n^2/4) = (n^2-1) log(n)
=> log (n^2/4)/log(n) = (n^2-1)

which is a contradiction because RHS grows faster.

==========
Alternatively, following simpler analysis leads to something more concrete:
For minimum error x = (A'A)^-1*A'*b
Ax=A(A'A)^-1*A'*b is maximum when A(A'A)^-1*A'*b = b
=> A(A'A)^-1*A' = I
When A is square symmetric matrix (number of variables = number of clauses) and A=A':
=> A(AA)^-1*A = I
=> Least squares perfectly solves system of equations for CNFSAT if matrix of clauses is symmetric and square i.e error is zero
Least squares thus solves a subset of NP-complete set of input 3CNFs i.e exactly solves a promise NP problem in deterministic polynomial time.

References:

452.1 Advanced Engineering Mathematics - [Erwin Kreyszig - 8th Edition] - Page 357
452.2 Linear Algebra - [Gilbert Strang] - http://math.mit.edu/~gs/linearalgebra/ila0403.pdf
452.3 Least Squares Error - [Stephen Boyd] - https://see.stanford.edu/materials/lsoeldsee263/05-ls.pdf

---
453. (THEORY) Random Matrix Analysis of Least Squares Approximate CNFSAT solver - related to 452 - 15 September 2017

From previous definition of expected value of random matrix equivalent to a random 3SAT:
E(A(A'A)^-1*A'*b) = b*(E(A))^4
Previous derivation of expected value of a random matrix is further simplified by a different definition of expectation:
E(A) = [matrix of E(x(i,j))] i.e each entry of the matrix is evaluated independently.
Let the probability of each entry x(i,j) of the random matrix = p. Each entry is a literal or its negation and can take 0 or 1 values.
E(x(i,j)) = p
=> E(A) = [matrix of p for all entries]
=> b * (E(A))^4 = b*[matrix of np^2 for all entries]^2
= b*[matrix of m*n^2*p^4 for all entries]
= [matrix of m^2*n^2*p^4 for all entries]
E(A(A'A)^-1*A'*b) = [matrix of m^2*n^2*p^4 for all entries]
Trace(E(A(A'A)^-1*A'*b)) = m^3*n^2*p^4 which has maximum value equal to number of clauses m
=> m^3*n^2*p^4 <= m
=> m^2*n^2*p^4 <= 1
=> p^4 <= 1/(mn)^2
=> p^2 <= 1/(mn)
=> p <= 1/sqrt(mn)
This retrieves the probability distribution of the random 3SAT instances.
If all clauses have to be satisfied (ExactSAT), p = 1/sqrt(mn).
In previous example iterations of the solver, number of clauses = number of variables i.e p = 1/n.
This differs from the uniform probability assumption 1/(2n) per literal in earlier derivations including the negations too.
The promise subset of NP solved exactly by least squares is defined by the set of all random 3CNFs created with each
literal occurring with probability 1/n.

References:

453.1 Random matrices -

http://www.utstat.toronto.edu/~brunner/oldclass/431s09/readings/RandomMatrices.pdf

---
454. (FEATURE and THEORY) Commits - Approximate SAT solver - 15 September 2017

(*) Some debug statements have been included to print average probability of each literal's occurrence in random 3CNF.
(*) SATsolver has been re-executed with 20 clauses and 20 variables and probability of occurrence of each literal has been logged.
(*) This was necessitated to ascertain the pseudorandomness of the clauses and CNFs created.
(*) Average probability of a literal is less than previous estimate of 1/n = 1/20 = 0.05 for 100% satisfied clauses but still converges to usual 97% for 100 iterations.
(*) logs committed to testlogs/

---
455. (THEORY) Eigenvalues of Random Matrices, Riemann Zeta Function, MAXSAT ranking of merit, Promise problems and Some additional notes on CNFSAT solver scipy/numpy least squares implementation - 18 September 2017 - related to 24,432

The set of random 3CNFs created by the solver is just a subset of all possible 3CNFs because all least squares API in SciPy require number of variables to be equal to number of clauses, and an exception is thrown at runtime if they are unequal. A better alternative for these API which works for any combination of number of clauses and variables has to be found and substituted in least square invocation (replacing lsqr()/lsmr()). This is a limitation of SciPy/NumPy and not the least squares algorithm. This makes it a Promise SAT solver and not the universal SAT solver for time being. Also for huge number of variables/clauses, both lsqr() and lsmr() functions slow down heavily and it is difficult to test on low-end personal desktops. The most plausible reason for convergence of MaxSAT to ~98% for almost all executions so far is the promise nature of the problem and only subset of 3CNFs (number of clauses=number of variables) are solved. In above random matrix analysis, if number of variables=number of clauses, probability of choosing a literal (positive or negative) is 1/n which is also uniform if two sets of positive and negated literals each of size n are separately evaluated for probability (E(x(i,j)) = 0*1/n + 1*1/n). This also raises a question: Since promise-SAT instances are NP-complete,does solving promise-NP imply solving NP. Since least squares exactly solves this promise subset (square matrix of equal number of variables and clauses) when probability of choosing a literal is 1/n, percentage of clauses satisfied is ~98% which is close to 100% implying the probability of choosing a literal is slightly less than uniform 1/n. Exact average MaxSAT percentage of least squares solver can be analyzed only if there is a suitable alternative to SciPy/NumPy. Therefore this convergence still does not contradict P!=NP and PCP.

Earlier the problem of intrinsically ranking texts etc., based on merit has been reduced to a MAXSAT problem where each text document etc., satisfies a fixed or variable 3CNFSAT formed from merit variables (e.g textgraph complexity) and merit is ranked by percentage of clauses satisfied. All the quantified outcomes of intrinsic merit algorithms described in this document can be translated to merit variables e.g variable1 = 1 if graph tensor neuron merit > threshold else 0.

Random Matrix representation of random 3SAT instances throws up an unusual window into the realm of Riemann Zeta Functions. It has been known that non-trivial zeros of Riemann Zeta Functions and Eigenvectors of Random Matrices (N*N hermitian matrices) have something in common. This indicates a possibility of a connection between Riemann Zeta Function zeros and eigenvalues of Random

Matrices for Random 3CNFs.

References:

455.1 Promise Problems - http://www.cs.tau.ac.il/~amnon/Classes/2017-BPP/Lectures/Lecture11.pdf
455.2 Survey of Promise Problems - [Oded Goldreich] - http://www.wisdom.weizmann.ac.il/~oded/PSX/prpr-r.pdf - there exist BPP-complete promise problems though in general case there are no known BPP-complete problems. This is quite applicable in equating a non-majority social choice and a Condorcet Jury Theorem majority social choice e.g. RHS majority choice with error is promise-BPP-complete and LHS non-majority interview TQBF social choice is in PSPACE.
455.3 Gaussian Unitary Ensemble - Random Matrices - http://empslocal.ex.ac.uk/people/staff/mrwatkin//zeta/random.htm

---
456. (THEORY and FEATURE) CNF3SAT Approximate Solver Update - All possible variable-clause combinations - Commits - 18 September 2017

(*) Solve_SAT2() function has been changed to take both number of variables and clauses as parameters
(*) Bug in EquationsB creation has been fixed so that length of b equals number of clauses. This was causing dimension mismatch and incompatible dimensions in LSMR and LSQR
(*) Logs for 18*16 clauses-variables random 3CNFs have been committed to testlogs/. MaxSAT percentage is ~96% after 1200 random CNF iterations . Per literal probability is too less than 1/sqrt(mn) obtained from random matrix analysis and therefore converges much below 100%.

(*) Bugs fixed in prob_dist()
(*) restored lsmr()
(*) logs for some other clause-variable combinations have been committed to testlogs/. Observed per literal probability is close to random matrix probability of 1/sqrt(mn)

---
457. (THEORY and FEATURE) Inapproximability and Random Matrix Analysis of Least Square Approximate SAT solver - Commits - 19 September 2017

Previous Random Matrix Analysis shows expected number of satisfied clauses = m^3*n^2*p^4 where there are m clauses,n variables and p is the probability of choosing a positive or negative literal. For 21 variables and 20 clauses, following are the results after few iterations:

Iteration : 29
---
solve_SAT2(): Verifying satisfying assignment computed .....
---
a: [[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0]

```
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]]
b: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
a.shape: (20, 21)
b.shape: (20,)
solve_SAT2(): lstsq(): x: (array([  5.00000000e-01,   0.00000000e+00,
0.00000000e+00,
         1.00000000e+00,   3.89920951e-13,   1.00000000e+00,
         1.00000000e+00,   5.00000000e-01,   7.50000000e-01,
         1.00000000e+00,   5.00000000e-01,   7.50000000e-01,
         1.24863661e-13,   0.00000000e+00,   0.00000000e+00,
         0.00000000e+00,   5.00000000e-01,   5.00000000e-01,
         1.00000000e+00,   5.00000000e-01,   0.00000000e+00]), 2, 12,
1.870828693386971, 5.2032984761241552e-12, 4.898979485566356, 4.001266321699533,
2.7613402542972292)
Random 3CNF: (!x11 + !x3 + !x13) * (!x2 + x12 + x11) * (x20 + x11 + !x10) * (!x1
+ !x2 + x11) * (x5 + !x10 + x19) * (!x4 + x9 + !x17) * (x4 + !x1 + !x6) * (x5 +
x7 + x13) * (x9 + x11 + !x15) * (!x20 + !x19 + x12) * (!x14 + !x9 + x4) * (!x11
+ !x6 + x19) * (x10 + !x9 + !x15) * (x7 + !x16 + x5) * (!x7 + x17 + x8) * (!x3 +
x1 + x11) * (!x18 + !x7 + !x13) * (x5 + !x21 + x6) * (!x16 + !x9 + !x6) * (x1
+ !x6 + x18)
Assignment computed from least squares: [1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
0, 0, 0, 1, 1, 1, 1, 0]
CNF Formula: [['!x11', '!x3', '!x13'], ['!x2', 'x12', 'x11'], ['x20', 'x11', '!
x10'], ['!x1', '!x2', 'x11'], ['x5', '!x10', 'x19'], ['!x4', 'x9', '!x17'],
['x4', '!x1', '!x6'], ['x5', 'x7', 'x13'], ['x9', 'x11', '!x15'], ['!x20', '!
x19', 'x12'], ['!x14', '!x9', 'x4'], ['!x11', '!x6', 'x19'], ['x10', '!x9', '!
x15'], ['x7', '!x16', 'x5'], ['!x7', 'x17', 'x8'], ['!x3', 'x1', 'x11'], ['!
x18', '!x7', '!x13'], ['x5', '!x21', 'x6'], ['!x16', '!x9', '!x6'], ['x1', '!
x6', 'x18']]
Number of clauses satisfied: 20.0
Number of clauses : 20
Assignment satisfied: 1
Percentage of clauses satisfied: 100.0
Percentage of CNFs satisfied so far: 46.6666666667
Average Percentage of Clauses per CNF satisfied: 96.0
y= 46
sumfreq= 896
y= 43
sumfreq= 896
y= 36
sumfreq= 896
y= 46
sumfreq= 896
y= 41
sumfreq= 896
y= 42
sumfreq= 896
y= 48
sumfreq= 896
y= 37
sumfreq= 896
y= 48
sumfreq= 896
y= 39
```

sumfreq= 896
y= 46
sumfreq= 896
y= 42
sumfreq= 896
y= 42
sumfreq= 896
y= 50
sumfreq= 896
y= 39
sumfreq= 896
y= 33
sumfreq= 896
y= 41
sumfreq= 896
y= 42
sumfreq= 896
y= 44
sumfreq= 896
y= 43
sumfreq= 896
y= 48
sumfreq= 896
y= 40
sumfreq= 904
y= 43
sumfreq= 904
y= 51
sumfreq= 904
y= 36
sumfreq= 904
y= 38
sumfreq= 904
y= 46
sumfreq= 904
y= 44
sumfreq= 904
y= 36
sumfreq= 904
y= 41
sumfreq= 904
y= 45
sumfreq= 904
y= 46
sumfreq= 904
y= 34
sumfreq= 904
y= 46
sumfreq= 904
y= 46
sumfreq= 904
y= 36
sumfreq= 904
y= 42
sumfreq= 904
y= 44
sumfreq= 904
y= 47
sumfreq= 904
y= 51
sumfreq= 904
y= 54
sumfreq= 904
y= 38

sumfreq= 904
Probability of Variables chosen in CNFs so far: [0.05133928571428571, 0.04799107142857143, 0.04017857142857143, 0.05133928571428571, 0.04575892857142857, 0.046875, 0.05357142857142857, 0.041294642857142856, 0.05357142857142857, 0.04352678571428571, 0.05133928571428571, 0.046875, 0.046875, 0.05580357142857143, 0.04352678571428571, 0.036830357142857144, 0.04575892857142857, 0.046875, 0.049107142857142856, 0.04799107142857143, 0.05357142857142857]
Probability of Negations chosen in CNFs so far: [0.04424778761061947, 0.04756637168141593, 0.05641592920353982, 0.03982300884955752, 0.0420353982300885, 0.05088495575221239, 0.048672566371681415, 0.03982300884955752, 0.04535398230088496, 0.049778761061946904, 0.05088495575221239, 0.03761061946902655, 0.05088495575221239, 0.05088495575221239, 0.03982300884955752, 0.046460176991150445, 0.048672566371681415, 0.051991150442477874, 0.05641592920353982, 0.059734513274336286, 0.0420353982300885]
Average probability of a variable or negation: 0.047619047619
Probability per literal from Random Matrix Analysis of Least Squared (1/sqrt(mn)): 0.0487950036474

From PCP theorem and [Hastad] inapproximability result based on it, if m^3*n^2*p^4 > 7/8*m then P=NP.
=> m^2*n^2*p^4 > 7/8
=> p^4 > 7/(8m^2n^2)
=> if p > 0.96716821/sqrt(m*n) then P=NP.

In previous iteration, y and sumfreq print the frequencies of literals chosen so far at random (which are almost evenly distributed) and observed average probability of choosing a literal is 0.047619 whereas the required Random Matrix analysis probability for exact SAT is somewhat higher at 0.048795. Substituting the observed probability 0.047619 for 21 variables and 20 clauses:
Expected number of clauses satisfied = (20)^3*(21)^2*(0.047619)^4 = 18.1405 clauses or 90.7025%
implying the MaxSAT for 21 variables and 20 clauses converges to 90.7025% asymptotically ad infinitum.

The inequality m^3*n^2*p^4 > 7/8*m relates hardness and randomness because probability distribution for p is directly related to uniformity and k-wise independence of pseudorandom (number) generator for choosing a literal at random. Presently the randomness depends on linux PRNG. If there exists a pseudorandom number generator corresponding to probability p > 0.96716821/sqrt(m*n) then P=NP.

For a binary valued random variable X, bias(X) is defined as Pr(X=0)-Pr(X=1) <= epsilon. Epsilon biased pseudorandom generators (e-PRG) are functions G:{0,1}^l->{0,1}^n expanding seed of length l to a pseudorandom string of length n and for all subsets Si (which are random variables) of {0,1}^n, bias(Si) <= epsilon. For uniform distributions, bias of all subset random variables is zero excluding the empty set. This is equivalent to: Bias(UniformDistribution)-Bias(e-PRGDistribution) <= epsilon. For random matrix approximate SAT solver probability p, previous condition of probability of choosing a positive or negative literal in a random SAT, p > 0.96716821/sqrt(m*n) for P=NP to hold, differs from the uniform probability distribution 1/2n (assuming n variables and n negations of them). This implies an epsilon biased PRG of bias p-(1-p) = 2p-1 = [1.93433642/sqrt(m*n)] - 1 for all possible values of m and n, must exist for P=NP. PRG described in reference has bias (n-1)/2^l. Equating 2p-1 = (n-1)/2^l and solving for seed length l gives an expression of required seed length l in terms of number of clauses m.

Let Xi be the random variable for choosing the literal xi. If literal xi is chosen Xi=1, else Xi=0. Therefore bias(ePRG)=Pr(Xi=1)-Pr(Xi=0). But Pr(Xi=1) = p = 0.96716821/sqrt(m*n) for choosing a literal from random matrix analysis derived previously for 7/8 approximation to exist.

An example application of Epsilon biased Pseudorandom Generator in reference 457.1 below :

=> bias(ePRG)=p-(1-p)=1.94/sqrt(m*n) - 1.
Equating this to the ePRG bias (n-1)/2^l:
=> (n-1)/2^l = [1.94/sqrt(m*n)] - 1
=> length of the seed l in ePRG = log([(n-1)*sqrt(m*n)]/[1.94-sqrt(m*n)]
Absurdity is for length of the seed to be valid logarithm, sqrt(m*n) < 1.94 => m*n < 4 which is possible only for m=1 and n=3, the trivial 1 clause 3SAT. This necessitates transforming previous into an inequality as below:

$$\frac{2^{2l} * (1.94)^2}{n*[2^l + (n-1)]^2} < m$$

For large n, inequality tends to a surd 0 < m.

An example application of Epsilon biased Pseudorandom Generator in reference 457.2 below :

=> bias(ePRG)=p-(1-p)=1.94/sqrt(m*n) - 1.
Other ePRG (Mossel-Shpilka-Trevisan) bias 1/2^(kn/c^4) mentioned in the references relates m and n for random matrix CNF SAT solver bias as:

$$\frac{(1.94)^2 * 2^{(2kn/c^4)}}{n*(1 + 2^{(kn/c^4)})^2} < m$$

For large n, inequality tends to a surd 0 < m.

Both these prescribe a relation between n and m for small n and for large number of variables both ePRGs fit to the random matrix CNF SAT solver least squares bias for P=NP to hold. This implies there are random SAT instances for small n which might not create literals with probability 0.96716821/sqrt(m*n) a hindrance to conclude that P=NP and defining this small n is non-trivial.

Generic Epsilon biased Pseudorandom Generator of bias epsilon=1/x for x > 0:

Consider a fictitious Epsilon biased Pseudorandom Generator of bias epsilon=1/x, x > 0.
Lowerbounding bias:

$$[1.94/sqrt(m*n)] - 1 <= 1/x$$

which reduces to:

$$m >= (1.94)^2*x^2/[n*(x+1)^2]$$

This implies for any epsilon biased PRG, there exists number of clauses m below which Approximate SAT solver solves less than 7/8 fraction of the clauses. Largest possible value of m occurs for n=1 and x tending to infinity => m >= 1.94*1.94 (or) m >= 3.7636 i.e all CNF random SAT instances of 4 or less number of clauses can not be solved by least squares to get atleast 7/8-approximation.

Number of possible random k-SAT instances of m clauses and n variables = (nP3 * 2)^m because each clause has either a literal or its negation, not both. Variables and their negation form a set of ordered pairs (x1,x1'),(x2,x2'),..., (xn,xn') from which 3 ordered pairs are chosen per clause in nP3 ways and variable or its negation is chosen from each ordered pair thereby creating nP3 * 2 possible clauses. An m clause random 3SAT has thus (nP3 * 2)^m possibilities.

Number of possible 4 clause 3SATs are = (nP3 * 2)^4. Therefore fraction of random SAT instances not solvable for 7/8 approximation is:

$$\frac{(nP3 * 2)^4}{(nP3 * 2)^m} = \frac{1}{(n*(n-1)*(n-2))^{(m-4)}}$$

which is negligible for large m and n but non-trivial for small m and n.

Number of variables in 4 clause 3SAT is at the maximum 12. Thus this small

subset can be solved in a constant time because m and n are fixed at 4 and 12 respectively.

[Caution: Previous derivation is still experimental with possible errors and assumes an epsilon PRG of bias atleast [1.94/sqrt(m*n)]-1 exists. If it indeed does, then it could imply P=NP and is in direct conflict with Majority version of XOR lemma - if numerator hardness does not cancel out - for hardness amplification described earlier and caveats therein which implies P != NP.]

References:

457.1 Epsilon biased Pseudo Random Generators - [Advanced Complexity Theory Course Notes - Dieter] - http://pages.cs.wisc.edu/~dieter/Courses/2008s-CS880/Scribes/lecture21.pdf
457.2 Epsilon biased Pseudo Random Generator in NC0 - every pseudorandom bit depends only on 5 bits of seed - [Elchanan Mossel, Amir Shpilka, Luca Trevisan] - https://www.stat.berkeley.edu/~mossel/publications/prginnc0.pdf - "... Then we present an ε-biased generator mapping n bits into cn bits such that ε = 1/2^Ω(n/c^4) and every bit of the output depends only on k = 5 bits of the seed. The parameter c can be chosen arbitrarily, and may depend on n. The constant in the Ω() notation does not depend on c ..."

---
458. (THEORY) Intrinsic Merit, Consensus Algorithms, Byzantine Failures and Level Playing Field - 23 September 2017

So far Intrinsic merit of a text has been analyzed mostly in the context of connectedness and meaningfulness of it. It assumes a document text-graph (subgraph of an ontology like WordNet) obtained from the Ontology graph is implicitly agreed upon metric to measure merit i.e IM(text) = subgraph of Ontology, for some intrinsic merit algorithm IM. But the problem of "agreeing" by stakeholders on some process is itself a non-trivial Consensus Problem which has been overlooked so far. For example, merit measured by interview/examination question-answering or a competition requires all parties to agree upon the terms, conditions and rules a priori. This is widely studied problem of Agreement or Consensus. Consensus is defined by:

* Agreement - all parties must agree on a correct process and its outcome
* Validity - if all correct processes receive same input value,they must all output same value
* Termination - all processes must eventually decide on an output value

There are realworld consensus implementations - Google Chubby Lock Service based on Paxos Consensus Protocol, Bitcoin's Proof-of-work hyperledgering which appends transactions of a node to common log in a distributed timestamp server etc., ensuring all participants agree. As opposed to Majority voting which requires crossing just half-way mark in number of votes(>50%), Consensus requires complete agreement (=100%). Once 100% consensus is achieved on how to measure intrinsic merit, it is accepted as a standard and levels playing field for contestants. Consensus is most required in measuring human intrinsic merit than merit of documents. In other words, majority voting permits each voter to decide on his/her own volition. Each voter can have different decision function(boolean or non-boolean). But Consensus requires all voters to reach an agreement on a standardized decision function. Recursive Gloss Overlap algorithm and its enhancements to measure intrinsic merit from graph of text can be termed as "consensual" because connectedness implies meaningfulness by WordNet distance measures (Resnick, Wu-Palmer, Jiang-Conrath etc.,) and any text can be mapped to a subgraph of an ontology like WordNet and WordNet has been accepted as a consensus peer-reviewed standard.

Consensus can be impeded by presence of malicious nodes which spoof and send spurious votes known as Byzantine generals problem. A known result implies there is a consensus resilient to byzantine failures if number of faulty nodes is not above 33%.

References:

458.1 Consensus Algorithms -
https://en.wikipedia.org/wiki/Consensus_(computer_science)

---
459. (THEORY) Random Matrix Rounding for Least Squares Approximate CNFSAT Solver, Distinguisher for Pseudorandomness, Majority Hardness Lemma - related to 318, 457 - 13 October 2017

Existence of Pseudorandom generators (PRG) implies P != NP. Proof of this is by contradiction: If there exists a distinguisher which is able to discern perfect randomness from any PRG in polynomial time then existence of PRGs which fool a distinguisher is ruled out i.e Pr[A(x)=1]-Pr[A(G(s))=1] >> epsilon where x is a perfect random string in {0,1}^n , A is a distinguisher and G is a function extending a seed s of length m to n - G:{0,1}^m to {0,1}^n , m < n .

Majority Voting Hardness Lemma is an adaptation of Yao's XOR Lemma for hardness amplification from weak voting functions composed to Majority function (related to KRW Conjecture and Boolean Function Composition). Hardness of a boolean function is the error in approximating the function f by a boolean circuit C defined by probability Pr[f(x) != C(x)]. Majority Hardness Lemma implies hardness of the majority+votingfunction composition amplifies the hardness compared to individually weak voters and inverting this composition (MajorityInverse: finding who voted in favour or against) is extremely hard to approximate by a circuit and thus in average case could be an one-way function composition. Being one-way implies hard-to-distinguish PRGs can be constructed from this composition.

Random Matrix Rounding for Least Squares Approximate CNFSAT Solver in previous sections derives an expected probability of choosing a literal in a CNF for ExactSAT(when all clauses are satisfied). This expected random matrix probability (mentioned as RMLSQR henceforth: 1/sqrt(m*n)) corresponds to some hypothetical pseudorandom generator PRG1.

Probability of choosing a CNF by RMLSQR (there are 3m literals per 3CNF) = (1/[mn])^1.5m

Alternatively,number of all possible random 3CNF SATs of m clauses and n variables = (n * n * n)^m = n^(3m)

Probability of choosing a CNF from all possible n^(3m) random 3SATs in this uniform distribution is = (1/n)^3m

Thus there are two possible probability distributions for choosing a random 3SAT: (1/[mn])^1.5m required by Random Matrix rounding and (1/n)^3m for uniform. Probability distributions and Pseudorandom generators underlying these distributions are directly related.

If m=n(number of clauses and number of variables are equal):

(1/[nn])^1.5n = (1/n)^3n and thus both RMLSQR and Uniform distributions are same implying similar pseudorandomness in RMLSQR and Uniform and distinguisher is fooled.This is a promise special case described earlier and does not suffice.

If m !=n e.g m >> n (this is most prevalent setting where number of clauses are huge and variables are relatively less):

(1/[mn])^1.5m < (1/n)^3m
(1/m)^1.5m * (1/n)^1.5m < (1/n)^1.5m * (1/n)^1.5m

(1/m)^1.5m < (1/n)^1.5m
1/m < 1/n
=> m > n

When number of clauses m differs from number of variables n, RMLSQR and Uniform distributions are dissimilar implying there are two different randomness-es: PRG1 for RMLSQR and PRG2(or perfect randomness) for Uniform. Distinguisher for these two random generators has the probability of success defined by difference of the probability distributions for RMLSQR and Uniform = (1/n)^1.5m * [(1/n)^1.5m - (1/m)^1.5m] i.e when the number of clauses is high compared to number of variables, this difference is significant and distinguisher succeeds with high probability implying PRG1 is not pseudorandom and altogether PRGs may not exist at all. This coincides with > 87.5% of clauses getting satisfied breaking 7/8 barrier in average case and could be synonymous to Karloff-Zwick SDP relaxation algorithm for > 7/8 MAXSAT. This need not contradict majority+voterfunction composition hardness because MajorityInverse is a depth-2 #P-Complete counting problem (first step counts and inverts voter inputs to majority and second step counts and inverts assignments per voter and hardness is a function of sensitivity) and could be beyond P != NP purview (i.e. There could be pseudorandom generators indistinguishable only by an algorithm harder than NP).

---
460. (THEORY and FEATURE) Random Matrix Rounding for Least Squares Approximate CNFSAT Solver - various clause-variable permutations - numbers, some anomalous - 25 October 2017

Following are some random 3SAT iteration MAXSAT percentage numbers for multiple combinations of number of variables and clauses. Observed average probabilities of linux PRNG have some anomalies when substituted in random matrix expected number of satisfied clauses (>100%). Reasons for
this could be error in estimating linux PRNG probability distribution. Deficiencies of Linux PRNGs - especially randomness extractor from SHA - are already analyzed (random.c) in https://eprint.iacr.org/2005/029.pdf - [BoazBarak-ShaiHalevi]

##
17 variables, 18 clauses - 1066 random 3SATs:
##

Iteration : 1066
---
solve_SAT2(): Verifying satisfying assignment computed .....
---
a: [[0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0]]

```
b: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
a.shape: (18, 17)
b.shape: (18,)
solve_SAT2(): lstsq(): x: (array([  0.00000000e+00,   1.00000000e+00,
8.00000000e-01,
         1.00000000e+00,   2.00000000e-01,   1.00000000e+00,
         1.00000000e+00,   6.66666667e-01,   6.00000000e-01,
         0.00000000e+00,   0.00000000e+00,  -6.66666667e-01,
         6.00000000e-01,   1.89190298e-14,   6.66666667e-01,
         4.00000000e-01,   0.00000000e+00]), 2, 11, 1.653279569018299,
9.190906242470613e-13, 4.690415759823429, 4.19738295792774, 2.625515822335343)
Random 3CNF: (x3 + !x11 + !x16) * (!x16 + x3 + x5) * (x6 + !x9 + !x4) * (x15 + !
x10 + x8) * (!x14 + x2 + !x8) * (!x11 + !x10 + !x3) * (!x9 + x2 + x14) * (!x1 +
x3 + !x7) * (x2 + x8 + x12) * (!x8 + !x1 + !x13) * (!x10 + !x8 + x9) * (!x12 +
x4 + !x8) * (!x7 + x8 + !x4) * (!x8 + x16 + x9) * (!x12 + !x7 + x15) * (!x1 + x7
+ !x14) * (x3 + x9 + !x12) * (!x11 + x13 + x16)
Assignment computed from least squares: [0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
0, 1, 1, 0]
CNF Formula: [['x3', '!x11', '!x16'], ['!x16', 'x3', 'x5'], ['x6', '!x9', '!
x4'], ['x15', '!x10', 'x8'], ['!x14', 'x2', '!x8'], ['!x11', '!x10', '!x3'], ['!
x9', 'x2', 'x14'], ['!x1', 'x3', '!x7'], ['x2', 'x8', 'x12'], ['!x8', '!x1', '!
x13'], ['!x10', '!x8', 'x9'], ['!x12', 'x4', '!x8'], ['!x7', 'x8', '!x4'], ['!
x8', 'x16', 'x9'], ['!x12', '!x7', 'x15'], ['!x1', 'x7', '!x14'], ['x3', 'x9',
'!x12'], ['!x11', 'x13', 'x16']]
Number of clauses satisfied: 18.0
Number of clauses : 18
Assignment satisfied: 1
Percentage of clauses satisfied: 100.0
Percentage of CNFs satisfied so far: 61.0121836926
Average Percentage of Clauses per CNF satisfied: 97.3341664063
y= 1675
sumfreq= 28722
y= 1683
sumfreq= 28722
y= 1716
sumfreq= 28722
y= 1750
sumfreq= 28722
y= 1699
sumfreq= 28722
y= 1690
sumfreq= 28722
y= 1697
sumfreq= 28722
y= 1662
sumfreq= 28722
y= 1732
sumfreq= 28722
y= 1740
sumfreq= 28722
y= 1687
sumfreq= 28722
y= 1669
sumfreq= 28722
y= 1677
sumfreq= 28722
y= 1662
sumfreq= 28722
y= 1664
sumfreq= 28722
y= 1660
sumfreq= 28722
y= 1659
sumfreq= 28722
```

```
y= 1716
sumfreq= 28896
y= 1782
sumfreq= 28896
y= 1660
sumfreq= 28896
y= 1714
sumfreq= 28896
y= 1704
sumfreq= 28896
y= 1716
sumfreq= 28896
y= 1721
sumfreq= 28896
y= 1688
sumfreq= 28896
y= 1679
sumfreq= 28896
y= 1653
sumfreq= 28896
y= 1662
sumfreq= 28896
y= 1686
sumfreq= 28896
y= 1702
sumfreq= 28896
y= 1673
sumfreq= 28896
y= 1735
sumfreq= 28896
y= 1678
sumfreq= 28896
y= 1727
sumfreq= 28896
Probability of Variables chosen in CNFs so far: [0.058317665900703294,
0.05859619803634844, 0.05974514309588469, 0.06092890467237658,
0.059153262307638746, 0.05883991365503795, 0.059083629273727456,
0.057865051180279924, 0.06030220736717499, 0.06058073950282014,
0.05873546410417102, 0.05810876679896943, 0.058387298934614584,
0.057865051180279924, 0.057934684214191214, 0.05779541814636864,
0.057760601629412996]
Probability of Negations chosen in CNFs so far: [0.059385382059800665,
0.061669435215946845, 0.05744739756367663, 0.05931616832779624,
0.058970099667774084, 0.059385382059800665, 0.05955841638981174,
0.05841638981173865, 0.058104928017718716, 0.057205149501661126,
0.057516611295681065, 0.05834717607973422, 0.058900885935769656,
0.057897286821705425, 0.06004291251384274, 0.0580703211517165,
0.059766057585825025]
Average probability of a variable or negation: 0.0588235294118
Probability per literal from Random Matrix Analysis of Least Squared
(1/sqrt(mn)): 0.0571661950475

##############################################
18 variables, 19 clauses - 137 random 3SATs:
##############################################
------------------------------------------------------------
Iteration : 137
------------------------------------------------------------
solve_SAT2(): Verifying satisfying assignment computed .....
------------------------------------------------------------
a: [[1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0]
 [0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]]
b: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
a.shape: (19, 18)
b.shape: (19,)
solve_SAT2(): lstsq(): x: (array([  6.66666667e-01,   1.00000000e+00,  -
1.11111111e-01,
         3.33333333e-01,   0.00000000e+00,   1.00000000e+00,
         4.44444445e-01,   1.66666667e+00,   8.93039852e-11,
         0.00000000e+00,   0.00000000e+00,   6.66666667e-01,
        -6.66666666e-01,   0.00000000e+00,  -5.55555556e-01,
         6.66666667e-01,  -8.43769499e-15,   1.00000000e+00]), 2, 13,
2.0816659994661344, 5.027196399901854e-09, 5.2915026221291805,
4.8051011045747947, 2.8609762647129626)
Random 3CNF: (x4 + x1 + !x7) * (!x3 + x6 + x9) * (x3 + x12 + x7) * (!x13 + !x4 +
!x10) * (x12 + x16 + !x6) * (x12 + !x17 + !x14) * (x16 + !x2 + !x17) * (x17 +
x18 + !x7) * (x16 + !x2 + x4) * (x6 + !x17 + !x18) * (!x14 + !x5 + !x17) * (x2 +
!x14 + x17) * (x6 + x16 + x13) * (!x12 + !x10 + !x7) * (x13 + !x12 + x8) * (x15
+ x8 + x3) * (!x3 + !x9 + !x7) * (x6 + !x16 + !x13) * (x18 + !x6 + !x2)
Assignment computed from least squares: [1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
0, 0, 1, 0, 1]
CNF Formula: [['x4', 'x1', '!x7'], ['!x3', 'x6', 'x9'], ['x3', 'x12', 'x7'], ['!
x13', '!x4', '!x10'], ['x12', 'x16', '!x6'], ['x12', '!x17', '!x14'], ['x16', '!
x2', '!x17'], ['x17', 'x18', '!x7'], ['x16', '!x2', 'x4'], ['x6', '!x17', '!
x18'], ['!x14', '!x5', '!x17'], ['x2', '!x14', 'x17'], ['x6', 'x16', 'x13'], ['!
x12', '!x10', '!x7'], ['x13', '!x12', 'x8'], ['x15', 'x8', 'x3'], ['!x3', '!x9',
'!x7'], ['x6', '!x16', '!x13'], ['x18', '!x6', '!x2']]
Number of clauses satisfied: 19.0
Number of clauses : 19
Assignment satisfied: 1
Percentage of clauses satisfied: 100.0
Percentage of CNFs satisfied so far: 52.8985507246
Average Percentage of Clauses per CNF satisfied: 96.7581998474
y= 220
sumfreq= 3952
y= 243
sumfreq= 3952
y= 252
sumfreq= 3952
y= 230
sumfreq= 3952
y= 195
sumfreq= 3952
y= 203
sumfreq= 3952
y= 231
sumfreq= 3952
y= 225
sumfreq= 3952
y= 224
sumfreq= 3952
```

```
y= 228
sumfreq= 3952
y= 221
sumfreq= 3952
y= 224
sumfreq= 3952
y= 205
sumfreq= 3952
y= 216
sumfreq= 3952
y= 199
sumfreq= 3952
y= 206
sumfreq= 3952
y= 203
sumfreq= 3952
y= 227
sumfreq= 3952
y= 207
sumfreq= 3914
y= 218
sumfreq= 3914
y= 214
sumfreq= 3914
y= 220
sumfreq= 3914
y= 223
sumfreq= 3914
y= 204
sumfreq= 3914
y= 220
sumfreq= 3914
y= 233
sumfreq= 3914
y= 203
sumfreq= 3914
y= 210
sumfreq= 3914
y= 234
sumfreq= 3914
y= 208
sumfreq= 3914
y= 239
sumfreq= 3914
y= 180
sumfreq= 3914
y= 205
sumfreq= 3914
y= 232
sumfreq= 3914
y= 259
sumfreq= 3914
y= 205
sumfreq= 3914
Probability of Variables chosen in CNFs so far: [0.05566801619433198,
0.06148785425101214, 0.06376518218623482, 0.05819838056680162,
0.049342105263157895, 0.0513663967611336, 0.058451417004048586,
0.056933198380566805, 0.05668016194331984, 0.057692307692307696,
0.05592105263157895, 0.05668016194331984, 0.05187246963562753,
0.05465587044534413, 0.05035425101214575, 0.05212550607287449,
0.0513663967611336, 0.05743927125506073]
Probability of Negations chosen in CNFs so far: [0.05288707204905468,
0.05569749616760347, 0.054675523760858456, 0.05620848237097598,
0.05697496167603475, 0.052120592743995914, 0.05620848237097598,
```

```
0.05952989269289729, 0.051865099642309655, 0.05365355135411344,
0.05978538579458355, 0.05314256515074093, 0.061062851303014816,
0.045988758303525806, 0.052376085845682166, 0.05927439959121104,
0.06617271333673991, 0.052376085845682166]
Average probability of a variable or negation: 0.0555555555556
Probability per literal from Random Matrix Analysis of Least Squared
(1/sqrt(mn)): 0.0540738070436

############################################################
19 variables, 18 clauses - 34 random 3SATs
############################################################
------------------------------------------------------------
Iteration : 34
------------------------------------------------------------
solve_SAT2(): Verifying satisfying assignment computed .....
------------------------------------------------------------
a: [[0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0]
 [0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
b: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
a.shape: (18, 19)
b.shape: (18,)
solve_SAT2(): lstsq(): x: (array([  1.00000000e+00,   1.00000000e+00,
1.00000000e+00,
         1.00000000e+00,   1.00000000e+00,   1.00000000e+00,
        -3.65332764e-15,   0.00000000e+00,   0.00000000e+00,
         9.05525654e-15,   1.00000000e+00,   1.00000000e+00,
         0.00000000e+00,   9.05525654e-15,   0.00000000e+00,
        -1.00000000e+00,   1.00000000e+00,   3.34888367e-14,
        -3.65332764e-15]), 2, 12, 1.414213562373095, 2.093877713811129e-13,
4.795831523312719, 3.8125525032467102, 3.1622776601683884)
Random 3CNF: (x4 + x14 + x10) * (!x2 + !x11 + x5) * (x16 + x2 + x17) * (!x8 +
x11 + x7) * (!x5 + x11 + !x14) * (x4 + x18 + !x7) * (x17 + !x6 + !x5) * (!x10 +
x18 + x12) * (!x4 + x17 + !x11) * (!x8 + x1 + !x13) * (!x7 + x6 + x18) * (x11 +
x19 + !x16) * (!x6 + x2 + !x3) * (!x1 + !x2 + !x8) * (!x13 + !x10 + !x7) * (!x6
+ !x10 + x1) * (!x7 + x6 + !x10) * (x3 + !x18 + !x1)
Assignment computed from least squares: [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0,
0, 0, 0, 1, 0, 0]
CNF Formula: [['x4', 'x14', 'x10'], ['!x2', '!x11', 'x5'], ['x16', 'x2', 'x17'],
['!x8', 'x11', 'x7'], ['!x5', 'x11', '!x14'], ['x4', 'x18', '!x7'], ['x17', '!
x6', '!x5'], ['!x10', 'x18', 'x12'], ['!x4', 'x17', '!x11'], ['!x8', 'x1', '!
x13'], ['!x7', 'x6', 'x18'], ['x11', 'x19', '!x16'], ['!x6', 'x2', '!x3'], ['!
x1', '!x2', '!x8'], ['!x13', '!x10', '!x7'], ['!x6', '!x10', 'x1'], ['!x7',
'x6', '!x10'], ['x3', '!x18', '!x1']]
Number of clauses satisfied: 18.0
Number of clauses : 18
Assignment satisfied: 1
Percentage of clauses satisfied: 100.0
Percentage of CNFs satisfied so far: 80.0
```

```
Average Percentage of Clauses per CNF satisfied: 98.7301587302
y= 52
sumfreq= 942
y= 39
sumfreq= 942
y= 46
sumfreq= 942
y= 50
sumfreq= 942
y= 53
sumfreq= 942
y= 43
sumfreq= 942
y= 47
sumfreq= 942
y= 51
sumfreq= 942
y= 57
sumfreq= 942
y= 48
sumfreq= 942
y= 54
sumfreq= 942
y= 49
sumfreq= 942
y= 46
sumfreq= 942
y= 61
sumfreq= 942
y= 54
sumfreq= 942
y= 44
sumfreq= 942
y= 48
sumfreq= 942
y= 49
sumfreq= 942
y= 51
sumfreq= 942
y= 44
sumfreq= 948
y= 42
sumfreq= 948
y= 43
sumfreq= 948
y= 43
sumfreq= 948
y= 61
sumfreq= 948
y= 60
sumfreq= 948
y= 49
sumfreq= 948
y= 60
sumfreq= 948
y= 43
sumfreq= 948
y= 57
sumfreq= 948
y= 47
sumfreq= 948
y= 60
sumfreq= 948
y= 49
```

```
sumfreq= 948
y= 41
sumfreq= 948
y= 60
sumfreq= 948
y= 46
sumfreq= 948
y= 53
sumfreq= 948
y= 41
sumfreq= 948
y= 49
sumfreq= 948
Probability of Variables chosen in CNFs so far: [0.055201698513800426,
0.041401273885350316, 0.04883227176220807, 0.05307855626326964,
0.05626326963906582, 0.045647558386411886, 0.049893842887473464,
0.054140127388535034, 0.06050955414012739, 0.050955414012738856,
0.05732484076433121, 0.05201698513800425, 0.04883227176220807,
0.06475583864118896, 0.05732484076433121, 0.04670912951167728,
0.050955414012738856, 0.05201698513800425, 0.054140127388535034]
Probability of Negations chosen in CNFs so far: [0.046413502109704644,
0.04430379746835443, 0.04535864978902954, 0.04535864978902954,
0.06434599156118144, 0.06329113924050633, 0.05168776371308017,
0.06329113924050633, 0.04535864978902954, 0.060126582278481014,
0.049578059071729956, 0.06329113924050633, 0.05168776371308017,
0.043248945147679324, 0.06329113924050633, 0.04852320675105485,
0.05590717299578059, 0.043248945147679324, 0.05168776371308017]
Observed Average probability of a variable or negation: 0.0526315789474
Probability per literal from Random Matrix Analysis of Least Squared
(1/sqrt(mn)): 0.0540738070436
Percentage of Clauses satisfied - Observed Average Probability substituted in
Random Matrix Analysis of Least Squared (m^2*n^2*p^4): 89.7506925208

############################################################
16 variables, 15 clauses - 60 random 3SATs
############################################################
------------------------------------------------------------
Iteration : 60
------------------------------------------------------------
solve_SAT2(): Verifying satisfying assignment computed .....
------------------------------------------------------------
a: [[0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]]
b: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
a.shape: (15, 16)
b.shape: (15,)
solve_SAT2(): lstsq(): x: (array([  0.00000000e+00,   4.00000000e-01,
1.00000000e+00,
         1.00000000e+00,  -2.00000000e-01,   1.00000000e+00,
         0.00000000e+00,   6.00000000e-01,   6.00000000e-01,
         8.00000000e-01,  -2.28234722e-16,   0.00000000e+00,
```

```
         5.00000000e-01,   5.00000000e-01,   1.00000000e+00,
         0.00000000e+00]), 2, 9, 1.5491933384829675, 1.3065051242742054e-12,
4.242640687119285, 2.7186789946524619, 2.4617067250183342)
Random 3CNF: (!x8 + x10 + !x11) * (x11 + x3 + !x8) * (!x10 + !x16 + !x1) * (!x8
+ x6 + !x4) * (x9 + !x1 + !x10) * (!x5 + !x1 + x6) * (!x6 + x2 + x8) * (x10 + x2
+ x5) * (!x1 + !x8 + !x7) * (x3 + !x1 + !x12) * (x14 + !x11 + x13) * (!x1 + x10
+ !x2) * (!x13 + x15 + x11) * (x9 + !x1 + x10) * (!x1 + x4 + !x8)
Assignment computed from least squares: [0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1,
1, 1, 0]
CNF Formula: [['!x8', 'x10', '!x11'], ['x11', 'x3', '!x8'], ['!x10', '!x16', '!
x1'], ['!x8', 'x6', '!x4'], ['x9', '!x1', '!x10'], ['!x5', '!x1', 'x6'], ['!x6',
'x2', 'x8'], ['x10', 'x2', 'x5'], ['!x1', '!x8', '!x7'], ['x3', '!x1', '!x12'],
['x14', '!x11', 'x13'], ['!x1', 'x10', '!x2'], ['!x13', 'x15', 'x11'], ['x9', '!
x1', 'x10'], ['!x1', 'x4', '!x8']]
Number of clauses satisfied: 15.0
Number of clauses : 15
Assignment satisfied: 1
Percentage of clauses satisfied: 100.0
Percentage of CNFs satisfied so far: 65.5737704918
Average Percentage of Clauses per CNF satisfied: 97.1584699454
y= 75
sumfreq= 1376
y= 95
sumfreq= 1376
y= 77
sumfreq= 1376
y= 79
sumfreq= 1376
y= 96
sumfreq= 1376
y= 89
sumfreq= 1376
y= 79
sumfreq= 1376
y= 94
sumfreq= 1376
y= 86
sumfreq= 1376
y= 93
sumfreq= 1376
y= 101
sumfreq= 1376
y= 91
sumfreq= 1376
y= 92
sumfreq= 1376
y= 80
sumfreq= 1376
y= 77
sumfreq= 1376
y= 72
sumfreq= 1376
y= 93
sumfreq= 1369
y= 82
sumfreq= 1369
y= 82
sumfreq= 1369
y= 85
sumfreq= 1369
y= 84
sumfreq= 1369
y= 87
sumfreq= 1369
```

```
y= 84
sumfreq= 1369
y= 71
sumfreq= 1369
y= 80
sumfreq= 1369
y= 74
sumfreq= 1369
y= 90
sumfreq= 1369
y= 91
sumfreq= 1369
y= 100
sumfreq= 1369
y= 100
sumfreq= 1369
y= 88
sumfreq= 1369
y= 78
sumfreq= 1369
Probability of Variables chosen in CNFs so far: [0.05450581395348837,
0.0690406976744186, 0.0559593023255814, 0.057412790697674417,
0.06976744186046512, 0.06468023255813954, 0.057412790697674417,
0.06831395348837209, 0.0625, 0.06758720930232558, 0.07340116279069768,
0.06613372093023256, 0.06686046511627906, 0.05813953488372093,
0.0559593023255814, 0.05232558139534884]
Probability of Negations chosen in CNFs so far: [0.0679327976625274,
0.05989773557341125, 0.05989773557341125, 0.0620891161431702,
0.061358655953250546, 0.0635500365230095, 0.061358655953250546,
0.05186267348429511, 0.05843681519357195, 0.05405405405405406,
0.06574141709276844, 0.0664718772826881, 0.07304601899196493,
0.07304601899196493, 0.06428049671292914, 0.05697589481373265]
Observed Average probability of a variable or negation: 0.0625
Probability per literal from Random Matrix Analysis of Least Squared
(1/sqrt(mn)): 0.0645497224368
Percentage of Clauses satisfied - Observed Average Probability substituted in
Random Matrix Analysis of Least Squared (m^2*n^2*p^4): 87.890625

#########################################################
21 variables, 18 clauses - 6 random 3SATs
#########################################################
-------------------------------------------------------------
Iteration : 6
-------------------------------------------------------------
solve_SAT2(): Verifying satisfying assignment computed .....
-------------------------------------------------------------
a: [[0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

```
b: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
a.shape: (18, 21)
b.shape: (18,)
solve_SAT2(): lstsq(): x: (array([  5.00000000e-01,   1.00000000e+00,
0.00000000e+00,
         5.00000000e-01,   1.00000000e+00,   0.00000000e+00,
        -2.63027447e-16,   1.00000000e+00,  -3.67351548e-14,
         0.00000000e+00,   8.39171896e-12,   9.28077060e-17,
         0.00000000e+00,  -3.67351548e-14,   0.00000000e+00,
         2.65906351e-12,   1.00000000e+00,  -2.52681990e-14,
         5.00000000e-01,   1.00000000e+00,   5.00000000e-01]), 2, 12,
1.7320508075688767, 6.3185165416969518e-11, 4.898979485566356,
3.9602900491395312, 2.4494897427873532)
Random 3CNF: (x11 + x8 + x16) * (!x12 + x11 + x5) * (!x4 + x17 + x18) * (x5 + !
x18 + !x8) * (!x8 + x21 + x1) * (!x9 + !x20 + !x17) * (x4 + !x20 + x19) * (!x6 +
x17 + !x12) * (x2 + x7 + !x9) * (x17 + x14 + x9) * (x2 + !x13 + !x6) * (!x13 + !
x19 + x20) * (x20 + !x3 + x12) * (x8 + !x20 + !x7) * (!x8 + !x2 + !x7) * (!x2
+ !x9 + !x10) * (x5 + !x1 + x11) * (!x13 + x2 + !x11)
Assignment computed from least squares: [1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 1, 1, 1]
CNF Formula: [['x11', 'x8', 'x16'], ['!x12', 'x11', 'x5'], ['!x4', 'x17',
'x18'], ['x5', '!x18', '!x8'], ['!x8', 'x21', 'x1'], ['!x9', '!x20', '!x17'],
['x4', '!x20', 'x19'], ['!x6', 'x17', '!x12'], ['x2', 'x7', '!x9'], ['x17',
'x14', 'x9'], ['x2', '!x13', '!x6'], ['!x13', '!x19', 'x20'], ['x20', '!x3',
'x12'], ['x8', '!x20', '!x7'], ['!x8', '!x2', '!x7'], ['!x2', '!x9', '!x10'],
['x5', '!x1', 'x11'], ['!x13', 'x2', '!x11']]
Number of clauses satisfied: 18.0
Number of clauses : 18
Assignment satisfied: 1
Percentage of clauses satisfied: 100.0
Percentage of CNFs satisfied so far: 71.4285714286
Average Percentage of Clauses per CNF satisfied: 97.619047619
y= 10
sumfreq= 184
y= 9
sumfreq= 184
y= 5
sumfreq= 184
y= 7
sumfreq= 184
y= 10
sumfreq= 184
y= 2
sumfreq= 184
y= 11
sumfreq= 184
y= 5
sumfreq= 184
y= 8
sumfreq= 184
y= 13
sumfreq= 184
y= 14
sumfreq= 184
y= 9
sumfreq= 184
y= 8
sumfreq= 184
y= 10
sumfreq= 184
y= 4
sumfreq= 184
y= 9
sumfreq= 184
```

```
y= 11
sumfreq= 184
y= 9
sumfreq= 184
y= 10
sumfreq= 184
y= 9
sumfreq= 184
y= 11
sumfreq= 184
y= 8
sumfreq= 194
y= 14
sumfreq= 194
y= 9
sumfreq= 194
y= 11
sumfreq= 194
y= 7
sumfreq= 194
y= 10
sumfreq= 194
y= 11
sumfreq= 194
y= 11
sumfreq= 194
y= 11
sumfreq= 194
y= 11
sumfreq= 194
y= 11
sumfreq= 194
y= 6
sumfreq= 194
y= 11
sumfreq= 194
y= 4
sumfreq= 194
y= 6
sumfreq= 194
y= 11
sumfreq= 194
y= 3
sumfreq= 194
y= 8
sumfreq= 194
y= 14
sumfreq= 194
y= 4
sumfreq= 194
y= 13
sumfreq= 194
y= 11
sumfreq= 194
Probability of Variables chosen in CNFs so far: [0.05434782608695652,
0.04891304347826087, 0.02717391304347826, 0.03804347826086957,
0.05434782608695652, 0.010869565217391304, 0.059782608695652176,
0.02717391304347826, 0.043478260869565216, 0.07065217391304347,
0.07608695652173914, 0.04891304347826087, 0.043478260869565216,
0.05434782608695652, 0.021739130434782608, 0.04891304347826087,
0.059782608695652176, 0.04891304347826087, 0.05434782608695652,
0.04891304347826087, 0.059782608695652176]
Probability of Negations chosen in CNFs so far: [0.041237113402061855,
0.07216494845360824, 0.04639175257731959, 0.05670103092783505,
0.03608247422680412, 0.05154639175257732, 0.05670103092783505,
0.05670103092783505, 0.05670103092783505, 0.05670103092783505,
```

0.030927835051546393, 0.05670103092783505, 0.020618556701030927,
0.030927835051546393, 0.05670103092783505, 0.015463917525773196,
0.041237113402061855, 0.07216494845360824, 0.020618556701030927,
0.06701030927835051, 0.05670103092783505]
Observed Average probability of a variable or negation: 0.047619047619
Probability per literal from Random Matrix Analysis of Least Squared
(1/sqrt(mn)): 0.0514344499874
Percentage of Clauses satisfied - Observed Average Probability substituted in
Random Matrix Analysis of Least Squared (m^2*n^2*p^4): 73.4693877551

---

461. (THEORY) Space-Filling, Random Close Packing, Bin Packing and Voter Decision Functions - 27 October 2017 - related to 135

Parallel PRG and Cellular Automaton Algorithms for randomly filling a space with objects and their relevance to Linear Programming were described earlier. Space Filling is a problem studied in the field of Structural Topology. For example, density of spheres randomly packed into a container is approximately ~63.6% (reference below) which is exactly the problem solved by Parallel PRG and Cellular Automaton Randomized Space filling algorithms. This space filling problem can be formulated as a Voter Constraint Satisfaction Problem. Variant of Space filling is the NP-hard Packing problem where set of items of variable sizes have to be packed into set of bins of same volume by minimizing number of bins whereas Space filling has just one container and items are of similar sizes and random close packing converges to a fixed density percentage. If set of items in a random close packing within a container are equivalent to satisfied clauses (having same number of variables) of a Voter Decision Function, this topology result implies not more than ~V*63.6% of clauses (where V is the volume of container) can be satisfied per candidate(MAXSAT). Each n-sphere can be mapped to a satisfied clause of n variables (each variable corresponds to a bounded length equal to diameter of n-sphere on a dimension of the n-space).

Reference:

461.1 Random Close Packings of Spheres in a Container - Space-filling and Structural Topology - http://www.math.cornell.edu/~connelly/PackingsIII.IV.pdf and http://www-iri.upc.es/people/ros/StructuralTopology/

---

462. (THEORY) Reduction from Random Close Packing to CNFSAT - related to 461 - 30 October 2017

Each random close pack after a random shuffle shifts the centre of an n-sphere. Set of all possible centroids of an n-sphere in each random close pack are connected in a clause by disjunctions. There are as many clauses as number of n-spheres connected by conjunctions. For example,if R1,R2,R3,... are random close packs and n-spheres s1,s2,s3,... are constituents of these packs R1,R2,R3,... , Centroid of n-sphere si = Cik is different for each Rk i.e n-sphere si can exist on any of these possible centroids which implies disjunction Ci1 V Ci2 V ... for locations of n-sphere si in Random Close Packs R1,R2,R3,.... This creates one clause per n-sphere. For all n-spheres packed in container at random, following conjunction of clauses for all n-spheres completes the reduction:
      (C11 V C12 V C13 ...) /\ (C21 V C22 V C32 ...) /\ (C31 V C32 V C33 ...) /\
...
Each Cik is assumed to be boolean variable which is 1 if n-sphere si is located in centroid Cik in random close pack Rk, else 0.

Randomized Algorithm (Parallel PRG or Cellular Automaton) for this space filling random close pack finds a satisfying assignment to previous kSAT. This is not exactSAT because some spheres (clauses) might lie outside the container.

Topological maximum limit of 63% for this random close
pack is the limit on density = Total Volume of Spheres (or) Total number of satisfied clauses / Volume of Container. As density increases , number of satisfied clauses increases. In other words, Total number of satisfied clauses = Volume of Container * PackingDensity = V*0.636. What Volume of Container translates to in the context of CNFs is open to interpretation - it might depend on the maximum-minimum range of centroid dimensions.

Filling the space within the container by n-spheres in parallel monte carlo random choice of centroids, simulates many natural parallel processes e.g shaking a container filled with equal sized balls. Parallel PRG and Cellular Automaton algorithms are for these special settings of random close packing (wherein size of the ball is infinitesimally finite) which are BPP/BPNC/RNC randomized algorithms to find a random close pack assignment to the previous kSAT. Each literal in this random close pack SAT is not just a centroid but also covers circular space around it of finite constant radius. Considering the usual example - shaking a container having closely packed balls: Every successive shake of the container creates an assignment to the kSAT. Formulating this kSAT requires prior knowledge of all possible centroid tuples which could be infinite. If shaking is of constant or polynomial time, churning out successive assignments to previous kSAT is surprisingly not hard. Also previous kSAT does not have overlapping literals across clauses.

---
463. (FEATURE-DONE) Ephemeris Search Script Update - Celestial Pattern Mining - 31 October 2017

Updated Ephemeris Search for Sequence Mined Celestial Configurations - toString() function has been changed to concatenate "Unoccupied" for vacant zodiac signs while creating encoded celestial chart.

---
464. (FEATURE-DONE) Ephemeris Search - Sequence Mining of Tropical Monsoon for mid-November 2017 - 1 November 2017

(#) Apriori GSP SequenceMining on autogen_classifier_dataset historic Storms data has been executed
(#) asfer.enchoros.seqmining has been updated from autogen_classifier_dataset
(#) MinedClassAssociationRules.txt has been rewritten containing almost 2500 celestial planetary patterns
(#) Ephemeris has been searched for almost 250 top astronomical patterns of these 2500 configurations for mid-November 2017
(#) There has been a significant match of most the patterns during this period. Range of search has been narrowed to 2 days because exhaustive search for all patterns is intensive.
(#) This matches to NOAA CPC forecast in http://www.cpc.ncep.noaa.gov/products/JAWF_Monitoring/India/GFS_forecasts.shtml

---
465. (THEORY) Discrete Hyperbolic Computational Geometric Factorization, Chvatal Art Gallery Theorem and Parallel Tiling - 6 November 2017, 8 November 2017 - related to 34

Chvatal Art Gallery Theorem states for floodlighting an art gallery shaped as n-polygon, floor(n/3) guards are sufficient. There is a graph theoretic proof of this by [Fisk]. Discrete Hyperbolic Factorization requires pixelation of continuous hyperbolic curve as set of contiguous tiles (array of pixels). This

computational geometric standard pixelation creates a polygon surrounding hyperbola of 4*(number_of_factors + 1) sides. This is because there are (number_of_factors + 1) pixelated rectangles in the polygon and each rectangle of the polygon has 4 sides. From Chvtal art gallery theorem 4/3 * (number_of_factors + 1) guards are sufficient to cover this polygon. This optimum number of guards is approximately the minimum number of parallel processors (PRAMs) required for tiling (pixelation) pre-processing step. Once these tiles are created in parallel, parallel sorting requires O(logN) time and post-processing binary search is also O(logN) - If there is a parallel binary search algorithm this could be sub-logarithmic. Assumption made previously is each guard has only vertical and horizontal 90 degree visibility.

As opposed to polygon pixelation, if the hyperbolic curve is discretized into set of line segments by plain rounding off creating a line segment for each interval [N/x, N/(x+1)], lower envelope of this set of segments is the list of endpoints of segments and there are no overlaps - segments meet only at endpoints. Number of line segments is twice the number of factors of N or O(loglogN). Computing lower envelope in parallel is equivalent to tiling in parallel. Lower envelope can be computed in parallel by CREW PRAM in O(logN) time and O(NlogN) operations.

References:

465.1 Chvatal Art Gallery Theorem and Fisk's Proof - https://www.cut-the-knot.org/Curriculum/Combinatorics/Chvatal.shtml
465.2 Number of prime factors of an integer - [Srinivasa Ramanujan] - Quarterly Journal of Mathematics, XLVIII, 1917, 76 – 92 - Ramanujan Papers and Notebooks - http://ramanujan.sirinudi.org/Volumes/published/ram35.pdf - number of prime factors are of O(loglogN). This implies number of rectangles (and hence number of PRAMs) in the pixelated hyperbola polygon could be sub-logarithmic in N.
465.3 Introduction to Parallel Algorithms - [Joseph JaJa] - Chapter 4 - Searching,Sorting,Merging - Corollary 4.5 - Cole's Pipelined Mergesort and Chapter 6 - Planar Geometry - Lower Envelopes and Visibility polygon - Theorem 6.7 - https://people.ksp.sk/~ppershing/data/skola/JaJa.pdf

---
466. (FEATURE-DONE) Support Vector Machines implementation - based on CVXPY - 9 November 2017

(*) New Support Vector Machines python implementation is committed to NeuronRain AsFer repository
(*) This minimizes an objective function 1/2*||w|| subject to constraint || WX+b|| >= 1 which
labels a point +1 or -1 on either side of a decision separating hyperplane WX+b for weight vector W
and bias b (Reference: Machine Learning - Ethem Alpaydin - Chapter 13 - Kernel Machines)
(*) Optimization is solved by CVXPY DCCP Convex-Concave program
(*) logs for this have been added to testlogs/
(*) Present NeuronRain AsFer SVMRetriever.cpp depends on third-party SVMlight opensource software. With
this new implementation, references to SVMlight are to be phased out.

---
467. (THEORY) Random Matrix Rounding for CNFSAT Solver, Blum-Micali PRG, Distinguisher for Pseudorandomness - 10 November 2017 -
related to 459

Blum-Micali PRG depends on intractability of Discrete Logarithm f(x)=g^x mod p for primes p,g and group element x. Blum-Micali PRG creates

sequences of (f(x), f^2(x), f^3(x), ...) by composition of f and computes stream of bits by hard-core boolean predicate function b(.) - (b(f(x)), b(f^2(x)), ...) which is sequence of unpredictable pseudorandom bits. In Random Matrix Rounding for CNFSAT solver by least squares, probability of choosing a literal xn = probability of the bit stream (b(f(x)),b(f^2(x)), ...) corresponding to binary encoding of n.

Probability of choosing a CNF by RMLSQR probability (p=1/sqrt(mn) and there are 3m literals per 3CNF) = (1/[mn])^1.5m
Probability of choosing a CNF from all possible n^(3m) random 3SATs in uniform distribution is = (1/n)^3m

Inverse of RMLSQR probability = (mn)^1.5m is the expected number of pseudorandom binary strings churned out by PRG before a required 3SAT is found.
Inverse of uniform probability = n^3m is the expected number of pseudorandom binary strings churned out by PRG before required 3SAT is found.

Distinguisher for these 2 distributions iterates through both sequences of bitstreams and prints "RMLSQR" if match occurs in (mn)^1.5m expected number of iterations else prints "Uniform". This requires exponential time in number of clauses.

From PCP [Hastad] inapproximability result and previous Random Matrix analysis for approximate CNFSATSolver, if m^3*n^2*p^4 > 7/8*m then P=NP i.e. if p > 0.96716821/sqrt(m*n) then P=NP. For the previous distinguisher, this requires (mn)^1.5m / (0.96716821)^3m exponential iterations. From condition for P=NP, this number of iterations has to be polynomial for efficient distinguishing of a PRG from perfect random sequence, which is a contradiction.

References:

467.1 Existence of Pseudorandom Generators - [Goldreich-Hugo-Luby] - http://www.wisdom.weizmann.ac.il/~oded/X/gkl.pdf

---
468. (FEATURE-DONE) Support Vector Machines - update - 10 November 2017

(*) Numpy indexing has been changed
(*) Both random point and parametric point distances have been tested
(*) DCCP log and Support Vector logs for 2 points and a random point have been committed to testlogs/
(*) Distances are printed in the matrix result - two diametrically opposite points have equal distances
(*) Distance matrix is returned from distance_from_separating_hyperplane() function
(*) The distance minimized is the L1 norm (sum of tuple elements) and not L2 norm
 (sum of squares of tuple elements)

---
469. (FEATURE-DONE) Computational Geometric Hyperbolic Factorization - Pixelated Segments Spark Bitonic Sort - 13 November 2017

(*) Numbers 147,219,251,253 are factorized.
(*) C++ tiling pre-processing routines have been changed for this and Pixelated Tiles storage text files for bitonic sort have been updated
(*) Factorization is benchmarked on single node cluster on dual core (which is parallel RAM).
(*) This is just a representative number on single dual-core CPU and not a cloud parallelism benchmark.

(*) Each DAGScheduler Spark work item is independent code executable in parallel and benchmark has to be this parallel time per work unit which is captured in per task duration logs below by Spark Driver:

```
17/11/13 21:18:22 INFO TaskSetManager: Finished task 1.0 in stage 1791.0 (TID 3583) in 46 ms on localhost (executor driver) (1/2)
17/11/13 21:18:22 INFO TaskSetManager: Finished task 0.0 in stage 1791.0 (TID 3582) in 47 ms on localhost (executor driver) (2/2)
17/11/13 21:18:22 INFO TaskSetManager: Starting task 0.0 in stage 1792.0 (TID 3584, localhost, executor driver, partition 0, PROCESS_LOCAL, 6281 bytes)
17/11/13 21:18:22 INFO TaskSetManager: Starting task 1.0 in stage 1792.0 (TID 3585, localhost, executor driver, partition 1, PROCESS_LOCAL, 6309 bytes)
17/11/13 21:18:22 INFO TaskSetManager: Finished task 1.0 in stage 1792.0 (TID 3585) in 46 ms on localhost (executor driver) (1/2)
17/11/13 21:18:22 INFO TaskSetManager: Finished task 0.0 in stage 1792.0 (TID 3584) in 50 ms on localhost (executor driver) (2/2)
17/11/13 21:26:39 INFO TaskSetManager: Finished task 0.0 in stage 1791.0 (TID 3582) in 50 ms on localhost (executor driver) (1/2)
17/11/13 21:26:39 INFO TaskSetManager: Finished task 1.0 in stage 1791.0 (TID 3583) in 49 ms on localhost (executor driver) (2/2)
17/11/13 21:26:39 INFO TaskSetManager: Starting task 0.0 in stage 1792.0 (TID 3584, localhost, executor driver, partition 0, PROCESS_LOCAL, 6281 bytes)
17/11/13 21:26:39 INFO TaskSetManager: Starting task 1.0 in stage 1792.0 (TID 3585, localhost, executor driver, partition 1, PROCESS_LOCAL, 6309 bytes)
17/11/13 21:26:39 INFO TaskSetManager: Finished task 0.0 in stage 1792.0 (TID 3584) in 48 ms on localhost (executor driver) (1/2)
17/11/13 21:26:39 INFO TaskSetManager: Finished task 1.0 in stage 1792.0 (TID 3585) in 49 ms on localhost (executor driver) (2/2)
17/11/13 21:33:59 INFO TaskSetManager: Finished task 0.0 in stage 1791.0 (TID 3582) in 48 ms on localhost (executor driver) (1/2)
17/11/13 21:33:59 INFO TaskSetManager: Finished task 1.0 in stage 1791.0 (TID 3583) in 50 ms on localhost (executor driver) (2/2)
17/11/13 21:33:59 INFO TaskSetManager: Starting task 0.0 in stage 1792.0 (TID 3584, localhost, executor driver, partition 0, PROCESS_LOCAL, 6281 bytes)
17/11/13 21:33:59 INFO TaskSetManager: Starting task 1.0 in stage 1792.0 (TID 3585, localhost, executor driver, partition 1, PROCESS_LOCAL, 6309 bytes)
17/11/13 21:33:59 INFO TaskSetManager: Finished task 0.0 in stage 1792.0 (TID 3584) in 55 ms on localhost (executor driver) (1/2)
17/11/13 21:33:59 INFO TaskSetManager: Finished task 1.0 in stage 1792.0 (TID 3585) in 57 ms on localhost (executor driver) (2/2)
17/11/13 21:40:20 INFO TaskSetManager: Finished task 1.0 in stage 1791.0 (TID 3583) in 49 ms on localhost (executor driver) (1/2)
17/11/13 21:40:20 INFO TaskSetManager: Finished task 0.0 in stage 1791.0 (TID 3582) in 52 ms on localhost (executor driver) (2/2)
17/11/13 21:40:20 INFO TaskSetManager: Starting task 0.0 in stage 1792.0 (TID 3584, localhost, executor driver, partition 0, PROCESS_LOCAL, 6281 bytes)
17/11/13 21:40:20 INFO TaskSetManager: Starting task 1.0 in stage 1792.0 (TID 3585, localhost, executor driver, partition 1, PROCESS_LOCAL, 6309 bytes)
17/11/13 21:40:20 INFO TaskSetManager: Finished task 0.0 in stage 1792.0 (TID 3584) in 50 ms on localhost (executor driver) (1/2)
17/11/13 21:40:20 INFO TaskSetManager: Finished task 1.0 in stage 1792.0 (TID 3585) in 50 ms on localhost (executor driver) (2/2)
```

(*) Bitonic Sort is O(logN*logN) and number of processors required is approximately O(N^2.5) but abides by NC definition. Cole Pipelined Merge Sort which is O(logN) is both in NC and Work-Time Optimal.

---

470. (FEATURE-DONE) Support Vector Machines Update - Learn and Classify functions - 13 November 2017

(*) Support Vector Machines python implementation has been updated to include two functions :

- for learning set of support vectors from a training dataset tuples and

store the vectors in a dictionary map of distance-to-vectors.
      - to classify a tuple by finding its distance with reference to the support vector regions - distance-to-vectors map is sorted and
      - if there are more than 1 tuples per minimum distance, those have to be reckoned as support vectors

---

471. (THEORY) Jones-Sato-Wada-Wiens Theorem, Complement Functions, Prime-Composite Complementation, Prime Number Theorem, Ulam Spiral, Ramsey 2-coloring of integers, Hilbert Tenth Problem, Unique Factorization , Matiyasevich-Robinson-Davis-Putnam Theorem, Riemann Hypothesis - related to 24,370,390,394 - 13 November 2017, 14 November 2017, 29 November 2017,1 December 2017,4 December 2017,23 December 2017
(Draft updates to https://arxiv.org/abs/1106.4102)

Jones-Sato-Wada-Wiens Theorem proves existence of a polynomial in 25 degree-26 variables which has values equal to set of all primes. This relates to Prime-Composite Function Complementation - Jones-Sato-Wada-Wiens polynomial is a complement function of set of composites which are formalized by unique factorization theorem. This is special case of Matiyasevich-Davis-Robinson-Putnam Theorem which proves any recursively enumerable set accepted by a Turing machine is equivalent to a polynomial accepting the elements of the set. Another closely related problem is: Does there exist a prime between two integers p=xy and q=x(y+1). Prime number theorem states and proves number of primes less than N=O(N/logN).

Number of primes < p=xy:
      = c1*xy/logxy
Number of primes < q=(x+1)y:
      = c1*(x+1)y/log(x+1)y

Number of primes between p=xy and q=(x+1)y :
      = c1*(x+1)y/log(x+1)y - c1*xy/logxy

Assuming number of primes between p and q = c1*(x+1)y/log(x+1)y - c1*xy/logxy > 0:
      (x+1)y/log(x+1)y > xy/logxy
After reducing:
      log(xy) > x(log(x+1) - logx)
      log(xy) > xlog(x+1/x)
For large x:
      log(x+1/x) ~ log 1 = 0
=>   log(xy) > 0
thus proving the assumption. This estimate of number of primes between two composites thus directly has bearing on discrepancy of this 2-colored (prime-composite) integer sequence where discrepancy is difference between number of elements of 2 colors in monochromatic arithmetic progressions. Ulam Spiral, which is sequence of integers written in concentric spiralling rectangle, has diagonals aligned along points of prime numbers described by polynomials. Rectangle of Ulam Spiral can be written as polynomial x(x+1) or x(x-1). Previous derivation on existence of primes between x(x+1) and x*x implies there is a prime diagonal. Ulam rectangle sequence for x(x+1) or x(x-1) is 1,2,4,6,9,12,16,20,25,30,36,42,...

Matiyasevich-Davis-Robinson-Putnam theorem implies every recursively enumerable set has a diophantine equation. Complement Function in several variables is nothing but a diophantine equation for the complement set by rewriting a diophantine as function in several variables. Thus concepts of Diophantine sets/ equations and Complement functions are synonymous. Decidability of complement functions (https://arxiv.org/abs/1106.4102) is equivalent to decidability of diophantine equations. MRDP theorem requires every recursively enumerable set to have a diophantine equation and therefore to have a function for it. [There

exists a set which is not recursively enumerable e.g set of subsets of infinite set]. Undecidability of Function Complementation follows from MRDP theorem because there exists a recursively enumerable set which is not computably recursive and hence has no algorithm (algorithms are computable recursive languages which do not loop and have yes/no halt on all inputs) i.e Undecidability of Hilbert's Tenth Problem applies directly to Undecidability of Function Complementation. In other words set of diophantines has cardinality greater than set of all computable recursive languages and therefore there exists a diophantine function for a complement set which cannot be computed by a Turing machine. This is one more proof of undecidability of complementation and simpler than Post Correspondence Problem based proof described earlier.

Definition:

For a set S and subsets A,B of S, if {A,B} is a disjoint set cover of S and if A has a diophantine equation diophantine(A) (expressible as values of polynomial), then diophantine equation for B = diophantine(B), is the complement function of diophantine(A).

Theorem: Existence of Complement Diophantine Equation or Complement Function is Undecidable when neither of the complementary sets are recursive.

Proof:

MRDP theorem for Hilbert's tenth problem implies every recursively enumerable set is expressible as values of a diophantine polynomial.

Possibility 1 - Generic - Both complementary sets A and B are recursively enumerable:

If both complementary sets A and B are recursively enumerable, they always have a diophantine polynomial - diophantine(A) and diophantine(B).
There is a known result which states: if set A and B=S-A are both recursively enumerable, then A is recursive. But there exists a recursively enumerable set B which is not recursive, for some complement A => There is no algorithm for construction of diophantine(B).

Possibility 2 - Special - One of the complementary sets A or B is recursively enumerable and the other is recursive:

Both A and B have a diophantine polynomial. If A is recursively enumerable but not recursive, there is a complement B (A=S-B) such that there is no algorithm for construction of diophantine(A).

Possibility 3 - Special - Both complementary sets A and B are recursive:

Both A and B are recursive and recursively enumerable => Both A and B have diophantine polynomials and both diophantine(A) and diophantine(B) can be constructed. There are many examples for this complementation: [Squares,PellEquation],[Composites/UniqueFactorizationDomain,Primes] etc.,

Possibility 4 - Special - Both complementary sets are non recursively enumerable:

Obviously both sets A and B are beyond Chomsky hierarchy and there is no algorithm for construction of diophantine(A) and diophantine(B)

Possibility 5 - Special - One of the complementary sets is recursively

enumerable and the other is non recursively enumerable:

One of the sets A is recursively enumerable and thus has a diophantine polynomial. But there exists a complement B=S-A such that A is not recursive => There is no algorithm for construction of diophantine(A)

Proof in one line: Any set is a complementary set of some other set and thus any complementary set which is recursively enumerable but not recursive has a diophantine equation, and thus undecidable by an algorithm.

Construction of a complement function for complementary set:

Construction of a complement function is to find a mapping function f defined as:

```
f(0) = a1
f(1) = a2
f(2) = a3
...
f(n) = an
```

for a1,a2,a3,...,an in Diophantine complementary set, which is equivalent to definition of recursively enumerable total function. This enumeration can be written as a diophantine equation:

```
f(x) - a = 0
```

Function f can internally be any mathematical function and can have additional parameters besides x. Finding the enumerated mapping previously is equivalent to solving an arbitrary diophantine f(x) - a = 0 i.e finding solutions to x and a which is Hilbert's Tenth Problem - MRDP theorem proves finding integer solutions to arbitrary Diophantine is undecidable. ArXiv version at https://arxiv.org/abs/1106.4102 mentions some algorithms for constructing a complement e.g Fourier series, Lambda calculus, Polynomial interpolation. Polynomial Reconstruction Problem also provides a polynomial approximation of the set and has origins in Error Correction and List Decoding. These algorithms apply only to recursive sets - complement function mappings are constructible only for recursive sets. This is intuitively obvious because a Turing Machine computing the previous mapping for a recursively enumerable set might loop for ever.

An important example for applicability of function complementation is ABC Conjecture which is defined in references below. Let S be the universal set of all possible integer triples (a,b,c). Set of coprime triples which have quality q(a,b,c) = log(c)/log(radical(abc)) > 1 + epsilon for all epsilon > 0 is the subset A of S. ABC conjecture states that there are only finitely many such triples for every epsilon > 0. Then the set B=S-A is the set of all triples which violate this criterion - sets A and B are complementary. Proving ABC Conjecture is equivalent to proving the violation of this criterion in the complement set B - there are infinite non-coprime triples for each epsilon > 0 which have q(a,b,c) > 1.This converts ABC conjecture from primal finiteness to its dual infiniteness.

References:

471.1 Jones-Sato-Wada-Wiens Theorem and Prime valued Polynomial - https://www.maa.org/sites/default/files/pdf/upload_library/22/Ford/JonesSatoWadaWiens.pdf - Jones-Sato-Wada-Wiens polynomial generates negative numbers too and its positive values are set of primes.
471.2 Matijasevic Polynomial - http://primes.utm.edu/glossary/xpage/MatijasevicPoly.html - negative values can be removed by setting squared terms of polynomial to zero and thus only set of all primes is generated which is the complement function of factorization.
471.3 Hilbert's Tenth Problem and Matiyasevich-Robinson-Davis-Putnam (MRDP) Theorem - https://en.wikipedia.org/wiki/Hilbert%27s_tenth_problem - Hilbert's Tenth Problem poses if there exists an algorithm which generates integer values for unknowns in a multivariate diophantine equation. Hilbert's Tenth Problem was

proved undecidable. Converse of this is finding a diophantine equation for a set - MRDP Theorem: Every recursively enumerable set is diophantine.
471.4 Primes are nonnegative values of a polynomial in 10 variables - [Yu.V.Matijasevic - https://logic.pdmi.ras.ru/~yumat/publications/publications.php] - https://link.springer.com/article/10.1007/BF01404106 - [In Russian, Translated to English by Louise Guy and James P. Jones, The University of Calgary, Calgary, Canada]
471.5 Pell's Equation - https://en.wikipedia.org/wiki/Pell%27s_equation - Diophantine equation for set of nonsquare integers - Discovered first in treatise Brahmasphutasiddhanta of Brahmagupta. This complements set of squared integers.
471.6 Non-Recursively Enumerable Languages - https://www.seas.upenn.edu/~cit596/notes/dave/relang8.html - powerset of infinite set is NonRecursivelyEnumerable.
471.7 Goedel's First Incompleteness Theorem Follows From MRDP theorem - https://en.wikipedia.org/wiki/G%C3%B6del%27s_incompleteness_theorems and http://www.scholarpedia.org/article/Matiyasevich_theorem - Diophantines and Complement Functions - "... The Diophantine equation of the general form P(a,x1, …,xm)=0 in the definition of a Diophantine representation can be replaced by a Diophantine equation of a special kind, namely, with the parameter isolated in the right-hand side, thus giving a representation of form a ∈ R ⇔ ∃x1…xn{Q(x1, …,xn)=a} In other word, every Diophantine (and hence every listable) set of non-negative integers is the set of all values assumed by some polynomial with integer coefficients with many variables. ..." - Any complement set which is enumerable/listable is diophantine and there exists a diophantine set which is not recursive but enumerable.
471.8 What is Mathematics: Goedel Theorem and Around - https://dspace.lu.lv/dspace/bitstream/handle/7/5306/Podnieks_What_is_Mathematics_Goedel.pdf?sequence=1 - Ramsey Theorem (finite and infinite) - Decidability of Function Complementation is the question: "For any two sets A and B which are disjoint set covers of a universal set S (A U B = S), and if set A is diophantine (is expressible as values of a polynomial), is B also diophantine and if yes is there a generic algorithm to construct a diophantine polynomial for B?". [Equivalently A and B are Ramsey 2-coloring of S and diophantine polynomials for A and B(if it exists) are 2-coloring schemes]. Answer to this question is two-fold:

(*) Is set B recursively enumerable? (There are sets which are not recursively enumerable e.g set of subsets of infinite set)

(*) If set B is recursively enumerable, B has an equivalent diophantine polynomial. But there exists a set B which is enumerable but not computably recursive and there is no generic algorithm for constructing complement diphantine polynomial. Undecidability of Complementation implies finding 2-coloring scheme is also undecidable.

(*) In both possibilities, there is a set which does not have a diophantine polynomial - there is a non-recursively enumerable set which is outside Chomsky-Schutzenberger Type-0,1,2,3 hierarchy and there is a recursively enumerable set which is not recursive computable by a Turing machine thus ruling out a generic procedure for finding 2-coloring schemes.
471.9 Goedel Incompleteness and MRDP theorem - https://plato.stanford.edu/entries/goedel-incompleteness/#HilTenProMRDThe - "... Beginning in the early 1950s, Julia Robinson and Martin Davis worked on this problem, later joined by Hilary Putnam. As a result of their collaboration, the first important result in this direction was achieved. Call an equation “an exponential Diophantine equation” if it involves also exponentiation, as well as addition and multiplication (that is, one can have both constants and variables as exponents); naturally, the focus is still in the integer solutions. Davis, Putnam, and Robinson (1961), showed that the problem of solvability of exponential Diophantine equations is undecidable. In 1970, Yuri Matiyasevich added the final missing piece, and demonstrated that the problem of the solvability of Diophantine equations is undecidable. Hence the overall result is often called MRDP Theorem (for an exposition, see, e.g., Davis 1973; Matiyasevich 1993). The essential technical achievement was that all semi-decidable (recursively enumerable) sets can be given a Diophantine representation, i.e., they can be represented by a simple formula of the form

∃x1…∃xn(s = t), where (s = t) is a Diophantine equation. More exactly, for any given recursively enumerable set S, there is a Diophantine equation (s(y, x1, … , xn) = t(y, x1, … , xn)) such that n ∈ S if and only if ∃x1…∃xn(s(n, x1, … , xn) = t(n, x1, … , xn)). As there are semi-decidable (recursively enumerable) sets which are not decidable (recursive), the general conclusion follows immediately:

MRDP Theorem

There is no general method for deciding whether or not a given Diophantine equation has a solution. ..."

471.10 Special Case of Complement Functions and MRDP Theorem - http://www.logicmatters.net/resources/pdfs/MRDP.pdf - Section 3 (Diophantine equation for set of Primes) and Section 4 - Theorem 4.3 - "If a set K and its complement N-K are both recursively enumerable then K is recursive". But MRDP theorem implies all recursively enumerable sets are diophantine and because there exists a recursively enumerable set which is not recursive, there is no algorithm to find a diophantine polynomial whose values is a set and thus the general case procedure for complementation is undecidable. Rephrasing theorem 4.3, If a set K is diophantine(K is expressible as values of a polynomial) and the complement set N-K is also diophantine(N-K is expressible as values of a polynomial), then K is recursive(there exists an Yes/No halt Turing machine accepting K). It has to be observed here that diophantine equations for K and N-K are complement functions. This special case is for the converse direction: assuming 2 complementary sets are diophantine, then one of the two is computable by an algorithm. Quoted excerpts from Definition 4.1 - "...Definition 4.1. A set of numbers K is recursively enumerable iff it is (i) the range of a total recursive function – or equivalently, it is (ii) the domain of a partial recursive function. What does this come to? Definition (i) says that K is recursively enumerable if there is an algorithmically computable (total) function f such that as you go through f(0), f(1), f(2), . . . you spit out values k0, k1, k2 . . . in K, with any member of K eventually appearing (perhaps with repetitions). The equivalent definition (ii) says that there is an algorithm such that the set of input numbers for which the algorithm halts is exactly the set of numbers in K ...". Partial recursive function is a primitive recursive function not necessarily defined for all inputs and computable by a Turing machine. Total recursive function is a partial recursive function which is defined for all inputs.

471.11 MRDP Theorem, Jone-Sato-Wada-Wiens Polynomial and Undecidability in Number Theory - [Bjorn Poonen] - http://www.cis.upenn.edu/~cis262/notes/rademacher.pdf - Diophantine statement of Riemann Hypothesis - " MRDP theorem gives a polynomial equation that has integer solutions if and only if Riemann Hypothesis is false " i.e find a counterexample to Riemann Zeta Function non-trivial zeroes - one which does not have Re(s) = 0.5. Turing machine for this might loop forever and thus recursively enumerable. Hence this Turing machine has a diophantine polynomial representation.

471.12 Formulas for Primes - https://oeis.org/wiki/Formulas_for_primes#Solutions_to_Diophantine_equations

471.13 Simplest Diophantine Representation - [Panu Raatikainen] - https://pdfs.semanticscholar.org/cd96/ead1a00b73ecc9cfabf4b9a617907ce9bdd6.pdf - Theorem 4 - This translates to the problem of determining simplest complement diophantine function of complexity measure K such that with respect to a complexity measure (e.g Kolmogorov Complexity and Chaitin incompleteness Theorem) every other diophantine for a complementary set has complexity greater than K. Finding Simplest Complement Diophantine is also undecidable.

471.14 Waring Problem and Diophantos/Bachet/Lagrange Four Square Theorem for real quarternions - Topics in Algebra - Lemma 7.4.3 and Theorem 7.4.1 - Page 373-377 - [Israel N. Herstein] - Ever positive integer is sum of four squares i.e There is a diophantine polynomial f(a,b,c,d) = a^2 + b^2 + c^2 + d^2 = n in N.

471.15 Complement Functions, Diophantine Analysis and ABC Conjecture - https://en.wikipedia.org/wiki/Abc_conjecture - ABC Conjecture in number theory is the following:

Let a + b = c be a mutually coprime integer triple f(a,b,c). Quality q(a,b,c) is defined as = log(c)/log(radical(abc)) where radical(abc) is the product of distinct prime factors of product abc. In most cases q(a,b,c) < 1. ABC

conjecture postulates existence of finitely many triples (a,b,c) for which q(a,b,c) > 1 + epsilon. f(a,b,c) : a + b = c is the diophantine equation for the set of coprime triples.
471.16 Erdos-Straus Diophantine Conjecture - https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93Straus_conjecture - For any integer n >= 2, there exist integer triples a,b,c which are solutions to diophantine equation : 4/n = 1/a + 1/b + 1/c
471.17 ABC Conjecture Proof-designate - Interuniversal Teichmuller Theory - [Shinichi Mochizuki] - http://www.kurims.kyoto-u.ac.jp/~motizuki/top-english.html
471.18 Exponential Diophantines, Fibonacci and Lucas Sequences, Maximum limit on solvability of Diophantine - Reduction of Unknowns in Diophantine Representations - [SunZhiWei] - http://maths.nju.edu.cn/~zwsun/12d.pdf - "... If we take into account the number of unknowns, then it is natural to ask that for what n there does not exist an algorithm to test (polynomial) Diophantine equations with n unknowns for solvability in integers..." - n=27 or n=11
471.19 Diophantine Equation Representation of Fibonacci Series - [James P.Jones] - https://www.fq.math.ca/Scanned/13-1/jones.pdf
471.20 Vandermonde Matrix and Polynomial Interpolation - https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/05Interpolation/vandermonde/
471.21 Number of zeroes of Riemann Zeta Function - [Norman Levinson - Hugh L.Montgomery] - https://projecteuclid.org/download/pdf_1/euclid.acta/1485889821
471.22 Diophantine Representation of Riemann Zeta Function - Davis-Matiyasevich-Robinson Theorem and Matiyasevich(1993) Theorem - "...Theorem 2 (Davis, Matiyasevich, and Robinson). The Riemann hypothesis is equivalent to the assertion (for n = 1, 2, 3, . . .) that (Sum(k≤δ(n) [1/k − n^2/2])^2 < 36n^3 ..." and "...Matiyasevich essentially takes advantage of the fact that the Riemann hypothesis implies that for n ≥ 600, |ψ(n) − n| < n^1/2*(logn)^2. Since ψ(n) = log(lcm(1, 2, . . . , n)),..." - http://web.stanford.edu/~anayebi/projects/RH_Diophantine.pdf (Philosophical Implication: Diophantines are contained in Adleman-Manders Complexity Class D a subset of NP. Proof of P=NP implies Riemann Hypothesis is easily solvable by an algorithm e.g ZetaGrid or humans)

---
472. (FEATURE-DONE) Support Vector Machines Update and Discrete Hyperbolic Factorization Spark Benchmarks
- 14 November 2017

(*) Support Vector Machines implementation has been updated to persist the learnt support vectors to
 a disk file SupportVectorMachines.txt
(*) This text file is read in classify()
(*) Support Vectors Dictionary is JSON dumped and eval()-ed and not JSON loaded because of serialization
  glitch in defaultdict(list)

(*) Computational Geometric Hyperbolic Factorization Spark implementation has been benchmarked for
 2 more integers of 9 and 10 bits. Some 8 bit numbers were benchmarked earlier.
(*) Spark logs for these benchmarks have been committed and time duration is calculated as 3 way split
 of real/user/system by time shell utility
(*) These benchmark numbers are on dual core single node Spark cluster
(*) C++/python files for tilings have been updated and pixelated tiles storage text files have been rewritten
---
Note on Factorization Spark benchmarks

Following are approximate correlations of the observed numbers to the theoretical polylogarithmic time bound - exponents of logN (=number of bits)

increase probably because number of parallel RAMs(cores) do not increase commensurate with the number of bits and is static dual core:

```
10 bit - real     17m11.931s = 1031.931s (~k*10^x3) ~ (logN)^3.013 (x3=3.013)
9 bit  - real     7m46.247s  = 466.247s  (~k*9^x2) ~ (logN)^2.796 (x2=2.796)
8 bit  - real     3m40.091s  = 220.091s  (~k*8^x1) ~ (logN)^2.589 (x1=2.589)
8 bit  - real     3m39.539s  = 219.539s  (~k*8^x1) ~ (logN)^2.589 (x1=2.589)
8 bit  - real     3m38.795s  = 218.795s  (~k*8^x1) ~ (logN)^2.589 (x1=2.589)
8 bit  - real     3m38.622s  = 218.622s  (~k*8^x1) ~ (logN)^2.589 (x1=2.589)
8 bit  - real     3m38.920s  = 218.920s  (~k*8^x1) ~ (logN)^2.589 (x1=2.589)
```

---

473. (THEORY) Computational Geometric Hyperbolic Factorization, Discrete Geometry, Rastering in Graphics/Computational Digital Geometry, Bresenham's Line Algorithm adapted for Hyperbolic tiling, Point Location, Ray shooting - 15,16 November 2017 - related to 34, 465

Finding factors of integer N by creating pixelated polygon for hyperbolic curve xy=N has great visual intuition. Similar algorithms already exist in discrete geometry and computer graphics disciplines. Bresenham's Line drawing algorithm is a classic used still in vector graphics which approximates a continuous line on a pixelated digital space. This approximation of continuous curves on digital space is called Rastering. Tile segments of hyperbola in preprocessing step of factorization are found by solving for deltay in:

```
xy = N
(x+1)(y-deltay) = N
xy - x*deltay + y - deltay = N
y = deltay * (x+1)
deltay = y/(x+1) for deltax=1
```

which is similar to Bresenham Line algorithm for finding next point to plot on raster. Tile segment (N/x, N/(x+1)) along y-axis is equal to interval (y, y-(y/x+1)) on y-axis.

Tiling preprocessing phase of factorization embeds a continuous hyperbola in a grid of horizontal and vertical straight lines. Each horizontal line corresponds to an integer in y-axis and vertical line to an integer in x-axis. Hyperbolic arc traverses the squares(pixels) of the grid. Thus the polygon approximating the continuous hyperbola is the union of all squares(pixels) through which hyperbolic arc passes. Union of squares create rectangular faces of the art gallery polygon vertices of which are the locations of the guards. Vertices of these rectangles through which hyperbolic arc passes through are the factors.

Art Gallery Pixelated polygon approximating a hyperbola is a Planar Simple Line Graph (PSLG) where each side of the polygon is an edge in PSLG. This PSLG has (number_of_factors + 1) rectangular faces which is O(loglogN). Vertices where two adjacent rectangular faces meet are the factor points of N. Planar point location has to find these factor points. Geometric Ray Shooting Query for this is : "Find points of vertices where two rectangular faces of polygon meet". This ray shooting can be picturised as a line from origin intersecting the rastered hyperbolic polygon and multiple rays from a common origin are shot in parallel of various (0-90 degrees) angles. O(loglogN) of these rays pass through the factor vertices. Number of rays required is proportional to length of the hyperbolic curve = O(N). Thus parallel ray shooting is a geometric sieve for factoring and is an alternative to sorting and binary searching the hyperbolic tile segments.

References:

473.1 Bresenham Algorithm for Line Rastering - https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

473.2 Rasterizing curves - http://members.chello.at/easyfilter/bresenham.pdf
473.3 Efficient Algorithms for Ray Shooting Queries - [Pankaj K.Agarwal] - https://epubs.siam.org/doi/pdf/10.1137/0222051
473.4 Parallel Planar Point Location - [Richard Cole] - https://ia601408.us.archive.org/33/items/onoptimalparalle00cole/onoptimalparalle00cole.pdf
473.5 Parallel Geometric Search - [Albert Chan,Frank Dehne, Andrew Rau-Chaplin] - https://web.cs.dal.ca/~arc/publications/1-16/paper.pdf
473.6 Parallel Computational Geometry - https://www.cs.princeton.edu/~chazelle/pubs/ParallelCompGeom.pdf - [A.Aggarwal,B.Chazelle,L.Guibas,C.O.Dunlaing,C.Yap] - Section 2 - Definition of NC based on PRAM - Section 5 - Algorithms for line segment intersection, art gallery guards, polygon triangulation, partitioning (kd-trees, quadtrees) are in NC. Factorization by approximating a hyperbola into a pixelated polygon can be rephrased as line segment intersection problem, where sides of the polygons are line segments and their intersecting points contain factor vertices. There are O(loglogN) intersection points in hyperbolic polygon where sides meet. Thus factorization can be solved in NC assuming pixelated hyperbolic polygon already exists.

---
474. (FEATURE-DONE) Computational Geometric Factorization - Tiling Update and benchmark numbers for factoring few integers - 16 November 2017

(*) Existing hyperbolic tiling code depends on C++ code in cpp-src/miscellaneous
(*) To remove this dependency, python script for tiling the hyperbolic arc with simple pixelation similar to bresenham algorithm has been committed to repository - this script writes two files suffixed .coordinates and .mergedtiles from rounding off the interval [N/x,N/(x+1)] in a function hyperbolic_tiling()
(*) This function is invoked in DiscreteHyperbolicFactorizationUpperbound_Bitonic_Spark.py before bitonic sort
(*) DiscreteHyperbolicFactorizationUpperbound_Bitonic_Spark.py is parametrized and accepts number to factorize as commandline argument:
```
       #$SPARK_2.1.0_HOME/spark-submit DiscreteHyperbolicFactorizationUpperbound_Bitonic_Spark.py <number_to_factorize>
```
(*) Few more time shell builtin benchmark numbers for factorizing 3-bit, 4-bit, 5-bit, 6-bit , 7-bit integers have been committed in testlogs/

---
475. (FEATURE-DONE) Computational Geometric Factorization Update - Spark Accumulators, JSON for Tiling etc., - 19 November 2017

(*) Hyperbolic tiling code in python has been changed to reflect C++ tiling in cpp-src/miscellaneous
(*) globalcoordinates global variable has been made Spark Accumulator Mutable global state - globalmergedtiles is already a Spark Accumulator global mutable state
(*) bitoniclock acquire/release statements have been added for global variables, but commented for benchmarking
(*) New boolean flags for bitonic comparator python style variable swap, for enabling multiple threads for assign have been added
(*) merge_sorted_halves() has been invoked as a separate function
(*) C++ tiling has been chosen because accuracy of long double in creating tile pixels is better than similar tiling code in python
(*) Benchmark numbers for factoring 511 has been added to testlogs/ and are similar to previous numbers
(*) .mergedtiles and .coordinates files are loaded/dumped as JSON in python hyperbolic tiling

---

---
476. (FEATURE-DONE and THEORY) Computational Geometric Factorization Update - Benchmarks and Tiling - 20 November 2017 - related to 34

(*) Some further changes to Python hyperbolic tiling have been made - tile endpoints have been cast from floating point to integer
for create_tile()
(*) C++ tiling in cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound_Bitonic.cpp has been parametrized and takes
as commandline argument the integer to factorize.
(*) Shell script cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound_Bitonic.sh has been changed to pass in the integer to
factorize to the C++ tiling binary in cpp-src/ and spark-submit for PySpark executable in python-src/ from shell args $*
(*) This shell script unifies C++ tiling and PySpark factorization
(*) Known issues: Python tiling still differs from C++ tiling significantly. Changing the hardcoded tile arrays storage in C++ tiling
to malloc()-ed heap storage causes faulty tiling. There are only optimization issues which do not affect the factorization. Factorization
works well and more benchmarks were done e.g for integers 723 and 501. Numbers for these and past numbers are charted below
(*) Following profiling numbers along with previous benchmarks for 8,9,10 bits (14 November 2017) capture the trend in the exponent of logN
(*) Gradual increase in exponent of number of bits because of static number of parallel RAMs (cores) is not quite steep from
3-bit to 10-bit integers as one might expect. Integers of same bit numbers need almost equal duration to factorize which implies the exponent
could be a function of logN and not N. Htop shows equal loading of both cores of CPU and consumption is almost 100%+100%.
(*) Cloud computing is not exactly a parallel RAM but multiple cores are PRAMs having concurrent access to a shared memory. Nodes in
cloud have local memory processing too. Spark's Global State Variables (Accumulators) which are reflected across all cloud nodes are
software simulations of Parallel RAMs - same global state is concurrently accessed by CPUs of spark nodes. Present PySpark implementation
 does the sorting on accumulators. Binary search is not necessary because Maximum elements at the end of the k-merge sorting of globalmergetiles
automatically have shuffled coordinates as factors in globalcoordinates accumulator. This shaves of additional O(logN). This optimization
is an update to drafts in 34.1 and 34.2:

- http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.tex/download
- http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.pdf/download

(*) Following are only representative figures and ideal benchmark requires a Spark cloud preferably on machines of high number multicore CPUs
 and comparison with existing General Number Field Sieve implementations (which is quasipolynomial-exponential).
---
723 - 10-bit - real   16m30.346s = 990.346s ~ O(logN^2.9956)
511 - 9-bit - real   7m46.518s = 466.518s ~ O(logN^2.8382)
511 - 9-bit - real   7m35.642s = 455.642s ~ O(logN^2.7854)
501 - 9-bit - real   7m36.879s = 456.642s ~ O(logN^2.7864) [shell script - includes time duration for C++ tiling]
100 - 7-bit - real   2m33.911s = 153.911s ~ O(logN^2.5851)
123 - 7-bit - real   2m52.704s = 172.704s ~ O(logN^2.6482)

63 - 6-bit - real   1m17.460s = 77.460s ~ O(logN^2.4243)
33 - 6-bit - real   0m58.950s = 58.950s ~ O(logN^2.2757)
14 - 4-bit - real   0m19.927s = 19.927s ~ O(logN^2.1609)
12 - 4-bit - real   0m19.651s = 19.651s ~ O(logN^2.1609)
6 - 3-bit - real   0m15.118s = 15.118s ~ O(logN^2.4649)

---

477. (THEORY) Computational Geometric Factorization - Tiling Optimizations - K-Merge Sort is dispensable and Local Tile Search is sufficient - 21 November 2017

Finding factor vertices in pixelated hyperbolic polygon is equivalent to geometric search query: "Is there a right turn followed by a downturn in the polygon?". These turning points coinciding with hyperbolic arc are factor points. Number of rectangles in pixelated hyperbolic polygon = O(loglogN). Each tile along y-axis in the polygon (array of pixels) is of length deltay = N/[x(x+1)]. Maximum number of tiles in the pixelation can be derived as:

xy=N
(x+deltax)(y-deltay)=N
=> xy + y*deltax - x*deltay - deltax*deltay = N
But deltax = 1,
=> y - (x+1) deltay = 0
deltay = y/(x+1) and y = N/x
=> N/x(x+1) = deltay > 1 [each tile should be of length atleast 1]
N > x*x + x
=> x = (sqrt(1 + 4N) - 1) / 2
For large N, sqrt(1 + 4N) ~ sqrt(4N) = 2*sqrt(N)
=> x ~ sqrt(N) for large N

Maximum number of tiles in the pixelation is O(sqrt(N)). If number of processors is O(sqrt(N)), each tile can be assigned to a processor and tiling can be done in O(1) parallel time. Number of processors required can be reduced by having O(sqrt(N)/(logN)^c) processors. O(sqrt(N)) tiles can be created in O((logN)^c) total parallel time i.e in each iteration O(sqrt(N)/(logN)^c) tiles can be created in parallel by as many processors and there are O((logN)^c) iterations. Creating a tile in a processor is defined as assigning the tile interval ordered pair of coordinates (tilestart, tileend) to it where tilestart=(x, y1) and tileend=(x, y2). This tile segment interval is implicitly sorted ascending/descending in one of the axes and therefore can be locally binary searched. This does away with Global K-Merge Sort of the locally sorted tile segments. Because each tile is of length N/x(x+1), |y2-y1| = N/[x(x+1)]. If N exists in a tile segment then there exists a factor point (x,y) in the tile segment such that y1 < y < y2 and xy=N. Binary search per implicitly sorted tile segment is of O(log(N/[x(x+1)]) <= O(logN).

Thus factors can be found just by:

477.1 Tiling in parallel requiring O((logN)^c) tile and O(sqrt(N)/(logN)^c) processors where number of tiles = O(sqrt(N))

477.2 Local Binary Search per processor on each sorted tile of O(logN) time and no k-merge sort is necessary.

Previous optimization reduces number of PRAMs by orders of (logN)^c but increases exponent of parallel time O((logN)^c). Bitonic K-merge sort is global and least parallel time though number of processors required is huge (yet in NC and work-time optimal). Bitonic K-merge sort doesnot require binary search because factors are in the forefront of the mergesorted and shuffled tile coordinates always.

References:

477.1 Line segment turn detection -

http://fileadmin.cs.lth.se/cs/Personal/Rolf_Karlsson/lect9.pdf - For two line segments, find if there is a left or right turn - line segments are vectors and sign of their cross product determines left or right turn. Assuming a pixelated polygon and
its sides as input array of line segments, cross product of adjacent pairs of line segment vectors can be computed in parallel and direction of turn can be found.
477.2 Sweepline algorithms - http://www.ics.uci.edu/~goodrich/pubs/ggb-sweep-j.pdf
477.3 Rectangle Stabbing - School on Geometric Computing, IIT Delhi (2010) - http://www.cse.iitd.ernet.in/~ssen/geomschool/nandy/TR-RS.pdf - Stabbing number for a set of axis-parallel rectangles is the minimum number of vertical and horizontal lines required such that each line passes through one of the rectangles covering all rectangles. This problem is NP-hard by reduction from set cover. Hyperbolic pixelation is a special case inverse problem of stabbing where a polygon of adjoining axis-parallel rectangles are created such that hyperbolic arc passes through each of them.

---
478. (FEATURE-DONE and THEORY) Computational Geometric Factorization and Parallel Tile Search Updates
- related to 34 and 477 - 22 November 2017

(*) cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound_Bitonic.cpp has been revamped and unnecessary code has been removed. Function names have been changed. NUMBER_OF_TILES has been declared as a macro and has to be manually changed and compiled to nearest power of 2 greater than number to factorize. Compilation and PySpark factorization is taken care of by shell script
cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound_Bitonic.sh
(*) New Spark python script
python-src/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been added to repository. This is the parallel tile search optimization already mentioned in NeuronRain AsFer Design Document. Spark parallelize() distributes the tiles-coordinates array of ordered pairs as RDDs across cloud nodes and node prints the factor point in y-axis if x-axis matches the number to factorize. Spark parallelize() simulates the binary search on a local tile segment per PRAM processor. Presently binary search has not been implemented in tilesearch() function of foreach() Spark Action. This is because map/foreach functions act on a single element argument of a parallelized RDD.
(*) How Spark parallelizes a sequential datastructure on cloud nodes is equivalent to binary search on an array of length 1 which is a no-op. Because of this parallize() of Spark has been assumed to be equivalent to a binary search.
(*) Benchmark numbers for two integers 1011 and 1013 with and without sorting have been committed to testlogs/ and Tiling time durations have also been captured on how well mere tile search optimizes as opposed to sorting.
(*) As evidenced from
python-src/testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.log.22November2017, tiling preprocessing is quite insignificant (<<1 sec) and tile search for 1011 and 1013 requires approximately 20 seconds while bitonic sorting duration for same numbers is ~16minutes (960 seconds) a speedup of almost 50x.
(*) Benchmarks for Local Tile Search obviate the requirement for k-merge sorting of pixelated hyperbolic tiles, if parallelism is huge.
(*) Following is the trade-off:
	- Bitonic K-merge sorting of tiles : binary search is not necessary but a tremendous performance drag and overkill, suitable for low number of PRAMs
	- Local tile search : binary search is necessary only per processor but requires relatively high number of PRAMs

---

---
479. (FEATURE-DONE) Computational Geometric Factorization Tiling Optimization - Binary Search for Tile Segments in Spark - 23 November 2017

(*) Binary Search has been implemented in Spark Tile Search Optimization by parallizing the set of intervals of pixelated hyperbolic arc.
(*) By this each tile interval can be binary searched in parallel to find the factor point with no
necessity for global k-merge sorting.
(*) Separate pixelation and tile interval file creation C++ source file has been added in cpp-src/miscellaneous
(*) Arrayless tile creation has been chosen and only intervals are written to a file suffixed as .tileintervals
(*) Shell script which compiles and executes the tile interval creation file and spark interval binary search script has been added.
(*) Invoking the shell script as:

cpp-src/miscellaneous/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.sh <number_to_factorize)
is sufficient to print the factors
(*) Following are benchmark numbers for factorizing few reasonably high bit integers. Removing K-Merge sort has significant effect on the
throughput even for a single node spark cluster on dual core (local[2]):
Factorization of 12093 (14-bit): real   0m33.108s (single core - local[1])
Factorization of 12093 (14-bit): real   0m29.589s (dual core - local[2])

Factorization of 65532 (16-bit): real   0m44.899s (single core - local[1])
Factorization of 65532 (16-bit): real   0m39.621s (dual core - local[2])

Factorization of 102349 (17-bit): real   1m5.490s (single core - local[1])
Factorization of 102349 (17-bit): real   0m58.776s (dual core - local[2])

Factorization of 934323 (20-bit): real   8m20.017s (single core - local[1])
Factorization of 934323 (20-bit): real   7m37.958s (dual core - local[2])

Factorization of 1343231 (21-bit): real 16m34.759s (single core - local[1])
Factorization of 1343231 (21-bit): real 10m49.218s (dual core - local[2])

(*) Above tile binary search numbers beat bitonic k-mergesort of tiles by many orders of magnitude
(*) logs containing factors of above integers have been committed to testlogs/
(*) This optimization thus effectively supersedes mergesort of tiles and is thus a better PySpark implementation of Discrete Hyperbolic Factorization.

(*) Updated AsFer Design Document - benchmark numbers for both single and dual cores for tile search
in computational geometric factorization
(*) Updated
python-src/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py for dual cores
(local[2])

---
480. (FEATURE-DONE) Computation Geometric Factorization - Binary Search for Tiles - Benchmarks
- 24 November 2017

Benchmark numbers for factoring 24 bit integer by Tile BinarySearch Optimization script have been committed to testlogs/ (single node cluster, dual core):

```
Factorization of 9333123 (24-bit):
--------------------------------
real    112m24.101s (Spark duration 12:21 to 14:02 = 101minutes = 6060seconds)
user    0m0.000s
sys     0m0.004s

Again binary search of tiles in parallel is way better than k-mergesort of
tiles. Largest known numbers
widely used are 128-bit. Trend for different bits indicates, Spark cloud of few
multicore nodes is
sufficient to factorize 128-bit integers in few minutes. Previous benchmark uses
spark-default.conf files
from python-src/InterviewAlgorithm/ (2GB of heapspace)

--------------------------------------------------------------------------------
------------------------------------------------------------
481. (THEORY) Computational Geometric Factorization - Parallel Local Binary
Search for Tiled Hyperbolic Arc - 27 November 2017
- Snir's Theorem for Parallel Search - related to 34,477
--------------------------------------------------------------------------------
------------------------------------------------------------
Previous optimized factorization with no sorting and only local binary search
per tile interval can achieve further speed-up (from logarithmic to
sublogarithmic) if search of each locally sorted tile can be done in PRAMs.
Snir's Theorem implies searching a table of sorted elements can be done in
sublogarithmic O(logN/logp) time by p CREW PRAM processors.

Tiling preprocessing of hyperbolic arc creates O(N) tiles. [This is proportional
to length of hyperbolic arc obtainable from elementary
calculus = DefiniteIntegral(sqrt(1 + [N^2/x^4]))]

Number of iterations = O((logN)^k)
Number of PRAMs = (N/(logN)^k)

Factorization in parallel has following algorithm:
while(iterations <= O((logN)^k))
{
      *) Assign N/(logN)^k tiles to N/(logN)^k PRAMs in parallel (O(1) parallel
time because each interval tile in a global shared memory array can be accessed
by PRAM id as index)
      *) Binary Search tile in each PRAM for factors (O(logN) parallel time
which can be reduced to O(logN/logp) by parallel binary search from Snir's
Theorem)
}

Previous loop totally is of [O(1) + O(logN)]*O((logN)^k) = O((logN)^(k+1))
parallel time.For minimum value of k=1, Factorization by parallel
local binary search of tiled hyperbolic arc, can be done in O((logN)^2) parallel
time and O(N/logN) PRAM processors without k-mergesort. Assigning N/(logN)^k
tiles to each of N/((logN)^k) PRAMs is of constant time assuming tile intervals
are in a global shared memory state i.e PRAM is simulated by a cloud global
distributed state - e.g. Spark Accumulators. Snir's Theorem implies O((logN)^2)
could be optimized to
O(logN*logN/logp) in CREW PRAM.

References:
----------
481.1 Efficient Parallel Algorithms and subclasses of NC - [KruskalRudolphSnir]
- https://ac.els-cdn.com/030439759090192K/1-s2.0-030439759090192K-main.pdf?
_tid=4ba471c8-d35a-11e7-9c39-
00000aab0f26&acdnat=1511777222_ae2aa80751a47624aeb524f723e969b0
481.2 On Parallel Search - Snir's Theorem -
https://pdfs.semanticscholar.org/3a58/58e8517f28fa586364daffb34160c437bf78.pdf
481.3 Point in Polygon (PIP) problem -
```

https://en.wikipedia.org/wiki/Point_in_polygon - queries if a point is inside, outside or on the polygon. Factor points are always within the pixelated hyperbolic polygon.
481.4 Simulation of BSP and CRCW PRAM in MapReduce - Sorting, Searching, and Simulation in the MapReduce Framework - [Michael T. Goodrich, Nodari Sitchinava, Qin Zhang] - https://arxiv.org/abs/1101.1902 - Theorem 3.2 - CRCW PRAM can be simulated on MapReduce clouds by logarithmic increase in parallel time.
481.5 Models of Parallel Computation - PRAM shared memory model can be simulated on BSP distributed memory model - http://mpla.math.uoa.gr/media/theses/msc/Lentaris_G.pdf - Bulk Synchronous Parallel model consists of sequence of supersteps. Each superstep performs a local computation in a processor, communicates to other parallel processors and synchronizes multiple local computations by barriers. BSP which is a distributed memory model than shared thus has close resemblance to cloud parallelism than PRAM.
481.6 PRAM memory access is unit time - [Guy Blelloch] - http://www.cs.cmu.edu/afs/cs/academic/class/15499-s09/www/scribe/lec2/lec2.pdf - "...Once again, recall that all instructions - including reads and writes - takes unit time. The memory is shared amongst all processors, making this a similar abstraction to a multi-core machine - all processors read from and write to the same memory, and communicate between each other using this...". First step in the loop of previous factorization thus implies a hypothetical machine of N/(logN)^k cores and each core reads tile assigned to it in O(1) time.

---
482. (FEATURE-DONE) Support Vector Machines - Mercer Theorem - Kernel Implementation - 29 November 2017

(*) This commit implements the kernel trick for lifting points in lower dimension to higher dimension by a Feature map so that
 decision hyperplane in higher dimension separates the points accurately.
(*) Mercer Theorem creates a kernel function unifies Feature map lifting and Dot product in higher dimension
(*) Feature map phi maps a point in a dimension d to a point in dimension d+k: phi(x) = X. Inner Product (Dot) of two vectors in dimension
d+k = phi(x)*phi(y). Mercer Theorem unifies the Feature map and Dot product into a Kernel function defined as series:
   K(x,y) = Sigma(eigenvalue(i)*eigenfunction(x)*eigenfunction(y))
(*) This implementation randomly instantiates a N*N square matrix, finds its Eigenvalues and Eigenvectors, and computes the series for
K(x,y) as per previous identity (neglecting imaginary parts of Eigenvalues and Eigenvectors)
(*) logs for this have been added to testlogs/

---
483. (FEATURE-DONE) Support Vector Machines - Mercer Kernel Update - 30 November 2017

(*) Mercer Kernel Function has been changed to return a tuple of feature mapped points and the dot product
(*) Feature mapped points in higher dimension are : [.....,
square_root(eigenvalue[i])*eigenfunction(x[i]),....]

---
484. (FEATURE-DONE) Compressed Sensing - Image Sketch implementation - 1 December 2017

(*) Sketch B of an image bitmap X is computed by multiplying with a random

matrix A: B=AX
(*) import ImageToBitMatrix from image_pattern_mining/ for mapping an image to a bitmap matrix
(*) Sketch matrix B contains compressed information of the larger image. Original image can be sensed from this sketch.

---

485. (FEATURE-DONE) Compressed Sensing Update - Decompression and Error estimation of recovered image bitmap from sketch - 4 December 2017

(*) Sketch Ax=B of an image bitmap has been persisted to a file CompressedSensing.sketch
(*) A is a random matrix of dimensions (m,n) m<<n and m is scaled by a sketch ratio variable
(*) Original image x is recovered from sketch Ax=B by inverting A and multiplying with sketch:

Ainverse.Ax = x_recovered

(*) For non-square only approximate pseudo inverse is computable.
(*) For inverting non-square matrix A, Moore-Penrose Pseudoinverse function pinv() from NumPy is invoked.
(*) Error of the recovered image computed by trace (sum of all entries) of the recovered image.
(*) Logs for multiple sketch ratios (row values:100,200,300,400,50) have been committed to testlogs/ which show an increase in error
as size of the sketch decreases.

---

486. (THEORY) Computational Geometric Factorization, Tile Search Optimization and Parallel Interval/Segment Search Trees - 5 December 2017
and 22 December 2017 - related to 34, 481

Tile Search optimization for finding factors of an integer described earlier, binary-search set of intervals in parallel for factor point.
This is the classic segment/interval search tree problem in Computational Geometry. Segments/Intervals are 1-dimensional polygons and represented in a binary search tree. Query point and the containing interval are searched on this binary search tree. Sequential construction of interval/segment search tree needs O(nlogn) time. Searching each tile segment in parallel and searching an interval/segment tree in parallel for factor points are equivalent. There are parallel segment tree construction algorithms on Bulk Synchronous Parallel(BSP) model and as Distributed Segment Trees. Distributed Segment Trees are based on Distributed Hash Tables and insertion/retrieval of a key is based on keyspace partitioning and routing in the network i.e Each node in the network is assigned a subset of segment id(s) in a binary partitioned space. Insertion/Retrieval primitives hop through the network and find the matching node for the segment. Most cloud key-store implementations are distributed hash tables. Example: For inserting n keys in O(logN) time, number of hops has to be O(logN) per key and each key is inserted in parallel in O(1) time. Minimizing number of hops implies maximizing mean degree of a node in the network and is a trade-off.

Unique Prime Factorization of an integer N = p1^k1 * p2^k2 ... * pn*kn
This can be rewritten as:

2^logN = 2^(k1*log(p1) + k2*log(p2) + ... + kn*log(pn))
=> logN = (k1*log(p1) + k2*log(p2) + ... + kn*log(pn))

SmallOmega(N) = number of distinct prime factors of N = O(loglogN) from Hardy-Ramanujan Theorem
BigOmega(N) = number of prime factors including multiplicity = sum of prime powers = k1 + k2 + k3 + ... + kn
But k1 + k2 + k3 + ... + kn < (k1*log(p1) + k2*log(p2) + ... + kn*log(pn)) = logN

=>	BigOmega(N) < logN
	BigOmega(N) = O(logN) [This is very high upperbound and not tight estimate]

Segment binary search tree representation of pixelated hyperbolic tile segments is described in references below.

Factorization algorithm for N based on Segment Tree Search is:
	(#) Construct Segment Tree of Pixelated Hyperbolic Segments in Parallel (O(logN) or between O((logN)^2) and O((logN)^3) depending on sorting)
	(#) Query the Segment Tree for factor points in O(k + logN) where k is the number of segment intervals to be reported having factor point N=pq. It is sufficient to report atleast 1 segment having factor point and thus k=1. Number of segments containing both prime and non-prime factor points is proportional to size of set of all subsets of distinct prime factors and multiplicity (SmallOmega and BigOmega). Geometric intuition for this is: By moving a sweepline across the x-y plane, factor points and segments containing them (which are nothing but all possible size 2 partitions of set of distinct prime factors which correspond to p and q in N=pq, where as Bell Number of this set is number of all possible partitions) are reached in sequence left-to-right sorted ascending. Size of set of all subsets of distinct prime factors = 2^SmallOmega(N) = 2^loglogN = logN. If multiplicity is taken into consideration, from the very high bound above number of sets of subsets can be as high as O(2^logN).

Thus there are 3 Computational Geometric Factorization algorithms described in this draft and all of them are PRAM algorithms and in NC requiring between O((logN)^2) and O((logN)^3) parallel time:
	(#) K-MergeSort and BinarySearch of Hyperbolic tile segments (mentioned in 34 - requires merge sort of tile intervals/segments)
	(#) Local Binary Search in parallel of hyperbolic tiles without K-MergeSort (mentioned in 481 - recent optimization, better than k-mergesort)
	(#) Segment Tree Representation and Binary Search of hyperbolic tile Segments (uses a classic datastructure, preprocessing is non-trivial and segment tree has to be constructed in parallel - requires merge sort of tile intervals/segments)

Note: First and Third factorization algorithms are equivalent because both involve sorting of pixelated hyperbolic tile segments. Second algorithm obviates sorting by assumption that each PRAM can locally do a binary search without involving peer PRAM processors. Theoretically all three have similar polylogarithmic (O(logN*logN) at best) runtime upperbounds. Second algorithm is better than First and Third probably because the constant involved in Big-O notation in second algorithm is very small compared to First and Third.

It is apt to mention the resemblance of K-MergeSort of Hyperbolic Tile segments and Burrows-Wheeler Transform (BWT) of a string. Burrows-Wheeler Transform sorts all rotation permutations of a string, extracts the last index fr
om each sorted permutation, concatenates them into a transformed string which has co-located substrings. This co-location of similar substrings is invertible and useful for compression and indexes. K-MergeSort Computational Geometric Factorization merges and sorts concatenated tile segment strings, resulting in a sorted string which has
colocated similar factor points.

An Unsorted Search algorithm has been implemented in NeuronRain (Section 500) to locate a query point on an unsorted array of numbers, by representing the numbers as arrays of hashtables for each digit.This algorithm can find factors in the concatenation of merged pixelated hyperbolic tile segments with no requirement for sorting. But this involves hashtable preprocessing.

Length of each tile on x-axis can be derived from equating for N:
	xy = (x+delta)(y-1)
	delta = x/(y-1) = N/[y(y-1)]

Sum of lengths of all pixelated hyperbolic tiles along x-axis is the series:
N/(1*2) + N/(2*3) + N/(3*4) + ...
But:
N/(1*2) + N/(2*3) + N/(3*4) + ... < N/1 + N/2^2 + N/3^2 + ... < N +
DefiniteIntegral_2_to_N(dx/x^2) = 1.5N-1
=> Sum of lengths of all pixelated hyperbolic tiles is upperbounded by 1.5N-1 or O(N)

Some more optimizations:

[#] Sorting is required only if end points of two adjoined segments are in conflict (only some, not all, elements in next tile are greater than last element of the preceding tile).

[Subsegment1] Elements of succeeding tile segment become bigger than last element in preceding tile segment if:
xy < (x + delta)(y-1)
xy < xy - x + delta(y-1)
=> delta > x/(y-1)

[Subsegment2] Similarly first element in succeeding tile is bigger than last element in Subsegment1 if:
(x+delta)y < (x + N/(y-1)(y-2))(y-1)

Splitting each tile segment of length l into two subsegments of length delta and l-delta and binary searching two sets of concatenations of these split tile segments can find factors in O(logN) sequentially with no necessity for PRAMs if concatenation can be done in sequential in O(logN) time (tree of list concatenations can be done in parallel on PRAMs in O(logN) time, but doing sequential concatenation of lists in sublinear time is an open problem). In other words hyperbolic arc bow is broken into two concatenated tile segement sets and these two tile concatenations are searched.

[#] If the quadrant containing hyperbolic arc is partitioned by parallel ray shooting queries from origin, separated by angles proportional to location of factor points, there is no necessity for sorting. Number of primes factors are O(loglogN). Thus kloglogN ray shooting queries from origin pierce the hyperbolic arc bow in parallel. If angular spacing between ray shooting queries are approximately equal, following trigonometric expression gives the approximate prime factor for each ray angle:
y(m) = SquareRoot(N/[tan(m*pi/(2*k*loglogN))]) - 1 for m=1,2,3,...,kloglogN
These are only approximate angles. Factors should lie in close proximity of the intersection points of these rays with in hyperbolic bow.

References:

486.1 Parallel Segment Trees - [AV Gerbessiotis] - https://web.njit.edu/~alexg/pubs/papers/segment.ps.gz
486.2 Distributed Segment Trees - [Guobin Shen, Changxi Zheng, Wei Pu, and Shipeng Li - Microsoft Research] - http://www.cs.columbia.edu/~cxz/publications/TR-2007-30.pdf
486.3 Computational Geometry - Algorithms and Applications - [Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars] - Chapter 10 - More Geometric Datastructures - Interval Trees and Segment Trees - http://people.inf.elte.hu/fekete/algoritmusok_msc/terinfo_geom/konyvek/Computational%20Geometry%20-%20Algorithms%20and%20Applications,%203rd%20Ed.pdf
486.4 Parallel Construction of Binary Search Trees - [MJAtallah,SRKosaraju,LLLarmore,GLMiller,SHTeng] - https://www.cs.cmu.edu/~glmiller/Publications/Papers/ConstructingTreesInParallel.pdf - general binary search trees can be constructed in parallel.
486.5 Segment Trees - definition and diagrams - [Computational Geometry Course Notes - Antoine Vigneron - King Abdullah University of Science and Technology] -

https://algo.kaust.edu.sa/documents/cs372l07.pdf - Segments are sorted by endpoints. Leaves of the binary search segment tree are atomic elementary intervals created by start-end coordinates of segments. Internal nodes of the segment tree contain list of segments. Internal node n of segment tree has a segment [s1,s2] if and only if Interval(n) is a subset of [s1,s2] and Interval(parent(n)) is not a subset of [s1,s2] and n is closest to root. Stabbing Query "Which are the segments containing a point q?" is answered by traversing the segment search tree from root recursively and choosing one of the two child subtree intervals containing q at each level. In Computational Geometric Factorization, segment binary search tree stores the tile segments of pixelated hyperbola and stabbing query is "Which of the tile segments have the factorization point N=pq?" which is O(k+logN) and k is the O(loglogN) because number of tile segments containing N is exactly equal to number of factors of N which is loglogN from Hardy-Ramanujan Theorem. Thus finding all factor points is O(loglogN + logN). It has to be noted here this further optimizes the factorization algorithm by local binary search of tiles in parallel described earlier in 481. Loop in the algorithm is replaced by a segment tree of tile segments. But construction preprocessing time of segment search tree is O(NlogN) which requires a parallel tile segment tree construction in polylogarithmic time mentioned in references below - Thus segment tree construction + stabbing query for factor points is: O(logN^2 + loglogN + logN) = O((logN)^2) which is same as the time required in Factorization loop.
486.6 Hardy-Ramanujan Theorem - https://en.wikipedia.org/wiki/Hardy%E2%80%93Ramanujan_theorem
486.7 Parallel Construction of Segment Trees - https://www.cs.dartmouth.edu/~trdata/reports/TR92-184.pdf - [Peter Su, Scot Drysdale] - Section 4.2 - Analysis of Algorithm S - "...Thus we expect runtime of algorithm S would be between O(logN^2) and O(logN^3) depending on how good our sorting algorithm is..."
486.8 Parallel Computational Geometry - Parallel Construction of Segment Trees is O(logN) time - Section 5 - Pages 308-309 - [A.Aggarwal,B.Chazelle,L.Guibas,C.O.Dunlaing,C.Yap] - https://www.cs.princeton.edu/~chazelle/pubs/ParallelCompGeom.pdf
486.9 Small omega - Number distinct prime factors of an integer - https://oeis.org/A001221 - is O(loglogN) from Hardy-Ramanujan and Erdos-Kac Theorems
486.10 Big omega - Number prime factors of an integer with multiplicity - https://oeis.org/A001222 - is sum of prime powers in unique factorization of n. For quadratfrei(squarefree) integers Big Omega = Small Omega. An optimization in the segment tree search or local binary search in parallel for factor points is it is not necessary to find all factorization points in stabbing query. Segment tree search or local binary search can stop once first factor is found. Other factors are obtained by repetitive application of this algorithm after dividing the integer by factor found in previous step.
486.11 Parallel Construction of Segment Trees applied in another setting - [Helmut Alt, Ludmila Scharf] - computing depth of arrangement of axis parallel rectangles - http://cs.au.dk/fileadmin/madalgo/PDF/Parallel_Algorithms_for_Shape_Matching.pdf and https://pdfs.semanticscholar.org/99e5/2d0c99263dd97d5db289b72f82b233528765.pdf - O((logN)^2) time parallel construction of a balanced search tree
486.12 Function plot of Number of PRAM processors - N/(logN)^k - For k=1, N/logN has been plotted in https://github.com/shrinivaasanka/asfer-github-code/blob/master/python-src/testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.NbylogN_Desmos_Function_Plot.pdf - N/(logN)^k can be rewritten as 2^(logN-kloglogN). If N^m = 2^(logN-kloglogN), m = log(logN - kloglogN)/logN << 1 for large kloglogN.
486.13 Interval Hash Trees - [T. F. Syeda-Mahmood, P. Raghavan, N. Megiddo] - http://www.almaden.ibm.com/cs/people/stf/papers/caivd99.pdf - Variant of Interval Trees which includes Merkle Hash Trees for Region hashing of affine rectangles within images. Merkle trees are trees of hashes - leaves are hashes of regions of file blocks, internal nodes are hashes of concatenations of hashes of its children nodes. If hyperbola is represented as an image, computational

geometric factorization reduces to querying the factor point or rectangles having factors in the image.
486.14 Efficient Parallel Algorithms for Geometric Clustering and Partitioning Problems (1994) - [Amitava Datta] - http://www.informatik.uni-freiburg.de/tr/1994/Report64/report64.ps.gz - Section 2.1 - Parallel Construction of Range Tree by creating Segment Tree in parallel in O(logN) time and O(N) processors.
486.15 Parallel computational geometry of rectangles - [Chandran-Kim-Mount] - https://link.springer.com/article/10.1007/BF01758750 - this algorithm is applied in 486.14 for parallel construction of segment trees.
486.16 SCALABLE PARALLEL COMPUTATIONAL GEOMETRY FOR COARSE GRAINED MULTICOMPUTERS - [FRANK DEHNE, ANDREAS FABRI, ANDREW RAU-CHAPLIN] - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.46.1429&rep=rep1&type=pdf - describes a Scalable Parallel Segment Tree Construction and Search Algorithm.
486.17 Simpler example of Segment trees - tree of nested intervals - http://www.eng.biu.ac.il/~wimers/files/courses/VLSI_Backend_CAD/Lecture_Notes/SegmentTree.ppt - each internal node has an interval of endpoints which is union of intervals of subtree rooted in it. E.g Internal node having children [10,11] and [11,13] has interval [10,13] - Set of pixelated tile segments of hyperbolic arc bow creates a lower envelope - list of tile segment endpoints - which has no overlaps. Segment tree of this envelope is parallelly created by one of the parallel segment tree constructions algorithms referred previously.
486.18 Parallel construction of Bounding Volume Hierarchy Trees (BVH) - BVH trees are computational geometric binary search datastructures similar to BSP and kd-trees for hierarchical space partitions. Each internal node of the search tree has union of volumes bounded by its children. In this respect BVH trees are 3-dimensional counterparts of segment trees. BVH trees can be constructed in parallel in GPU - https://dcgi.felk.cvut.cz/projects/ploc/ploc-tvcg.pdf
486.19 Parallel Algorithms for Geometric Problems - [Anita Chow] - Ph.D Thesis - http://www.dtic.mil/dtic/tr/fulltext/u2/a124353.pdf - Planar Point Location Binary Search Tree for Plain Simple Line Graph (similar to segment tree) - "...In this section we describe two algorithms: (i) the construction of a search structure for the set of edges on the SMl4 with N processors and (ii) the concurrent location of M points with M processors. The construction and the location run in time 0((logN)^2*loglogN) and O((logN)^2) respectively..." - Section 4.1 - Page 60
486.20 Parallel Augmented Maps (PAM) - https://arxiv.org/pdf/1612.05665.pdf - [Yihan Sun, Daniel Ferizovic, Guy Blelloch] - Recent Algorithm for Parallel Construction of Interval Trees by Parallel Augmented Maps - Segment Trees are specialized versions of Interval Trees. Interval Trees can also answer stabbing query on interval, not just a point - Section 5.1 and Section 6.2 - "...In parallel, on 10^8 intervals, our code can build an interval tree in about 0.23 second, achieving a 63-fold speedup. We also give the speedup of our PAM interval tree in Figure 6(d). Both construction and queries scale up to 144 threads (72 cores with hyperthreads)...."
486.21 Burrows-Wheeler Transform - Example transformed string "^BANANA|" - https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform

---
487. (THEORY) Thermodynamics Gas Diffusion Entropy Model of Information Diffusion in Social Networks and Text Graphs - 11 December 2017 - related to 383

Second Law of Thermodynamics implies entropy of a closed system always increases. Gas diffusion in a closed chamber distributes molecules at random with in the space of the chamber. Gas chamber is a 3-dimensional grid of volume N. If in initial state, all gas molecules were confined to a small fractional cubic volume, probability of finding m gas molecules in kN (k < 1) grid points is kNCm / NCm. As the fractional volume gets close to N (k is almost 1), gas molecules engulf the chamber. Increase in fractional volume facilitates increase in degrees of freedom. Entropy of this closed system is defined as

clogN where c is Boltzmann constant. Diffusion of information in Social network graphs can be likened to gas diffusion in closed chamber and adjacent vertices of a social network node are the neigbouring grid points. In other words, social network graph is plotted as non-planar simple line graph in a 3-dimensional space and information diffuses along edges of this graph. Entropy of this system can then be defined as O(logV) where V is number of vertices. Probability of finding a diffused information in any fraction kV of the network follows previous Gas diffusion model = (kV)Ci/VCi where i is the total number of information units diffused among social network vertices. For text defintion graphs, this is nothing but Korner Entropy.

References:

487.1 Solomon Asch Conformity Experiments and Herd behaviour - Information Diffusion in Social Media - Section 7.1 - http://dmml.asu.edu/smm/SMM.pdf - this is a psychology experiment where set of subjects are swayed by peer opinions and rational independent decision making ceases i.e majority voting can go wrong.
487.2 Emperor's New Mind - [Roger Penrose] - Inexorable increase in entropy - Page 405

---
488. (FEATURE-DONE) NeuronRain AsFer-KingCobra MAC Electronic Money - Proof-of-Work and Universally Unique ID Hash implementation -
14 December 2017

(#) Fictitious Message-As-Currency (MAC) in AsFer-KingCobra has been named "Neuro".
(#) This commit implements a non-trivial proof-of-work computation and finds a universally unique hash id for each Neuro currency
 which has 2 leading "ff"s - Proof of Work is analogous to reCAPTCHA in websites for filtering out robots
(#) Unique id is created from Boost UUID random generator and checked in a loop for 2 leading "ff" in stringified hex representation
(#) Protocol Buffer version has been upgraded to 3.5/15 and currency.proto has been recompiled with protoc
(#) asfercloudmoveclient.cpp has been updated to make a choice between std::move() and std::forward() move semantics:
     - std::move() just does =operator overload and moves Neuro currency over network
     - std::forward() + std::move() first deferences rvalue of && for currency uuid and then does =operator overload to move Neuro currency
(#) Limitation: Presently there is no documented way to create an rvalue for non-primitive datatypes. For example std::string can have rvalue
as literal string "xxxxxx".

---
489. (FEATURE-DONE) AsFer-KingCobra Neuro Electronic Currency - Rvalue NonPrimitive Perfect Forward - 18 December 2017

(#) New constructor for cloudmove which takes const char* uuid has been defined. This enables
assigning a string literal rvalue to cloudmove<currency::Currency> objects.
(#) New move operator= which takes const char* uuid has been defined. This internally instantiates
a currency::Currency object
(#) New move operator= which takes cloudmove<T>& lvalue has been defined.
(#) New value and a clause for move semantics, "nonprimitiveforward" has been defined. This clause
does std::forward() of an rvalue cloudmove<currency::Currency> and then invokes std::move() of the currency over network.

(#) This differs from move semantics "std::forward" which is specific to std::string uuid only, and thus solves the generic currency::Currency object rvalue forward and move.
(#) client and server logs for this network move have been committed to testlogs.

---
490. (THEORY) Generalization of Complementation Undecidability to Rationals(Q) and Reals(R) by H10(Hilbert Tenth Problem) and
Connection between ZF with Axiom of Choice (ZFC) and Complementation - related to 471 - 28 December 2017, 29 March 2018

Thus far Function Complementation described in drafts of this document are oriented towards set of natural numbers only - e.g Coloring of Integer Sequences, Diophantines Sets of Integers and associated Diophantine polynomials having Integer Solutions. As mentioned earlier, finding the map f from a diophantine set a={a1,a2,...,an} is equivalent to solving the diophantine f(x,a)=0 for unknown x and parameter a and f itself being an arbitrary unknown diophantine. Integer solutions to x in f(x,a)=0 creates the enumeration f(0)=a1,f(1)=a2,... which is the constructed complement map. But from MRDP theorem solving integer diophantine f(x,a)=0 is undecidable. MRDP theorem does not apply for set of reals which is uncountable/non-recursively-enumerable - Reals have cardinality 2^|N| where N is set of natural numbers (there are atleast two levels of infinities). Because any set of cardinality greater than set of natural integers are uncountable, reals cannot be enumerated as f(0),f(1),... [By Cantor diagonalization there exists a real between any two reals]. For reals, Sturm's and Tarski methods solve arbitrary diophantines for real solutions to unknowns and thus H10 is decidable for reals. But there are undecidable real diophantine problems which involve sine() function(details in references). Construction of complementation is not just about solving unknowns in a diophantine but to construct the diophantine itself which is harder problem than proving undecidability from H10. Constructing the complement diophantine by previous mapping procedure mandates x to be always a natural integer for enumerability/listability and existence of a corresponding diophantine for this enumerable set - f(0),f(1),f(2),... - once the mapping process terminates, interpolation can find the diophantine polynomial from the ordered pairs (x1,f(x1)),(x2,f(x2)),...,(xn,f(xn)). Undecidability of complement construction for recursively enumerable sets arises when mapping procedure itself is not recursive. Removing restriction that x has to be integer and admitting real solutions for a and x, makes f(x,a)=0 decidable.

Axiom of Choice: For set of sets S={s1,s2,s3,...} there exists a choice function C(si)=xi which chooses an element xi from every set si in S.

Following is a special case example of Axiom of Choice in Boolean Social Choice functions:
Depth-2 boolean function/circuit B leaves of which are sets s1,s2,s3,.... and nodes at level one are C(s1)=x1,C(s2)=x2,... for C(si) = 0 or 1 and si={si1,si2}, si1,si2 in {0,1}. Root is an AND/OR gate. For example C(si) is a boolean AND/OR function. Boolean complementation of C(si) flips the chosen element per each set si. This yields a complement choice function per set. If this example of Boolean Axiom of Choice complementation is generalized to any set and is True, S={C(s1),C(s2),C(s3),...,C(sn)} is a recursively enumerable set listed by function C and has a diophantine representation by MRDP theorem and its complement set created from complement choice function C' of C denoted by S'={C'(s1),C'(s2),C'(s3),...}. If this complement set S' is also recursively enumerable, then both sets S and S' are recursive. This implies choice functions C and C' are constructible and have a diophantine representation.

Disjoint Set Cover is also known as Exact Cover defined as subcollection S' of collection of subsets S of a universal set X, and each element in
X is contained in exactly one subset in S'. S' is a disjoint collection and the

exact set cover of X. Exact Cover problem is NP-complete and is solved by DLX Dancing Links algorithm. Complementary or Ramsey k-colored sets are created by Exact Set Cover.

Hilbert's Tenth Problem for Rationals (Q) is open. Some approaches to solving Diophantines over Q - defining integers via rationals i.e integers are definable in first order theory of rationals - have been mentioned in references.

References:

490.1 Diophantines for Reals - http://wwwmayr.in.tum.de/konferenzen/Jass07/courses/1/Sadovnikov/Sadovnikov_Paper.pdf - Section 1.2 - "...It would be natural to ask if the complements of the sets listed above are also Diophantine. We can easily build a Diophantine representation for the complement of the first set while the answer for two other complements is not so evident: ... " and Section 3
490.2 Uncountable sets and non-recursively enumerable languages - http://www.cs.colostate.edu/~massey/Teaching/cs301/RestrictedAccess/Slides/301lecture23.pdf
490.3 Deciding the Undecidable - https://books.google.co.in/books?id=1rjnCwAAQBAJ&pg=PA10&lpg=PA10&dq=real+solutions+to+diophantine+tarski&source=bl&ots=W2QvvG1KJX&sig=PqhrS-5ThqyokMAvXDIYNivBPAc&hl=en&sa=X&ved=0ahUKEwj05ofgmqzYAhUBro8KHUaYA-sQ6AEINDAC#v=onepage&q=real%20solutions%20to%20diophantine%20tarski&f=false - Tarski's Decision Procedure for Algebra over Reals
490.4 Introduction to Automata Theory, Languages and Computation - [John E.Hopcroft,Rajeev Motwani,Jeffrey D.Ullman] - Theorem 9.4 - Page 376 - If both language L and its complement L' are recursively enumerable, then L is recursive and L' is recursive as well. L'=A*-L for set of alphabets A.
490.5 Real Roots of polynomials - https://en.wikipedia.org/wiki/Root-finding_algorithm
490.6 Constructing Diophantine Representation of a Listable Set, Register Machines preferred over Turing Machines - https://books.google.co.in/books?id=GlgLDgAAQBAJ&pg=PA43&lpg=PA43&dq=constructing+diophantine+representations&source=bl&ots=UT5_nW-OUn&sig=vpIkAMBvoNBD1PvqdZ-qmP7QYME&hl=en&sa=X&ved=0ahUKEwjkxKWnsa_YAhVFr48KHcpJCvUQ6AEIPjAC#v=onepage&q=constructing%20diophantine%20representations&f=false - Chapter 2 - Martin Davis and H10 - [Y.Matiyasevich] - Constructing Diophantine representation for a listable complementary set is exactly construction of a complement function.
490.7 Diophantine Representation, Non-Deterministic Diophantine Machine (NDDM), Complexity class NP, Single Fold Diophantine - [Yuri Matiyasevich, St.Petersburg, Department of Steklov Institute of Mathematics of Russian Academy of Sciences] - https://www.newton.ac.uk/files/seminar/20120111170017302-152989.pdf - Adleman-Manders class D of Diophantines which have total binary length of unknowns <= [binary length of parameter]^k - Open Problem: Is D = NP?
490.8 DPR theorem and finite-fold diophantine representations - [Yuri Matiyasevich] - ftp://ftp.pdmi.ras.ru/pub/publicat/znsl/v377/p078.pdf - Every effectively enumerable set has single-fold exponential diophantine representation (a diophantine polynomial which has unknowns in exponents and value for unknowns per parameter tuple is unique) of the form: <a1,a2,...> in S <=> There exist y,x1,x2,...,a1,a2,... P(y,x1,x2,...,a1,a2,...) = 4^y + y for a diophantine polynomial P in integer coefficients.
490.9 Factoring Semi-primes (numbers of the form N=pq for prime p,q) by Diophantine Equations - http://www2.mae.ufl.edu/~uhk/FACTORING-VIA-DIOPHANTINE.pdf
490.10 Dancing Links X Algorithm For Exact Cover - Pentominoes Example - 4-way doubly linked list - [Donald E. Knuth] - https://arxiv.org/pdf/cs/0011047.pdf
490.11 Dancing Links - [Hitotumatu, Hirosi; Noshita, Kohei] (1979). "A Technique for Implementing Backtrack Algorithms and its Application". Information Processing Letters. 8 (4): 174–175. doi:10.1016/0020-0190(79)90016-4.
490.12 Hilbert Tenth Problem for Rationals - https://rjlipton.wordpress.com/2010/08/07/hilberts-tenth-over-the-rationals/ - "...One of the most celebrated such results is that deciding whether or not a polynomial {P(x_1,x_2,\dots,x_m)} has an integer solution is undecidable. This

is the famous negative solution to Hilbert’s Tenth. An obvious question, which is still open, is what happens if we ask for solutions where the {x_1,x_2,\ dots,x_m} are allowed to be rationals? This is open—I discussed it recently as a potential million dollar problem....An approach to proving that H10 extended to the rationals is also undecidable is to show that the integers are definable by a polynomial formula. Suppose \displaystyle \exists y_1 \dots \exists y_n \ R(x,y_1,\dots,y_n) = 0 is true for a rational {x} if and only if {x} is actually an integer. Note, the quantifiers range over rationals. Then, we can transform a question about integers into a question about rationals—this would show that H10 for rationals is also undecidable. ..."

---
491. (FEATURE-DONE) All pairs of encoded strings - Sum of Distances - algorithm mentioned in Grafit course notes implemented - 3 January 2018

(#) For pairwise pattern mining, sum of distances between binary encoded strings algorithm mentioned in Grafit course notes has been
implemented which is better than bruteforce.
(#) asfer.conf has been updated
(#) asferencodestr.cpp and asferencodestr.h have been updated for new class member functions (factorial and combinations)
(#) logs have been committed to testlogs/
(#) This is only in GitHub (NeuronRain Enterprise) repo which has binary encoded strings. NeuronRain Research repo in SourceForge
has astronomically encoded unicode strings which requires a variant of this implementation (must be sum of 10 combination terms for each character - each term is per symbol).

---
492. (THEORY and FEATURE-DONE) Complement Function Map Construction - Diophantine Representation - Lagrange's Four Square Theorem SymPy solver (Draft updates to: https://arxiv.org/abs/1106.4102) - related to 472 - 4 January 2018

(#) This commit implements diophantine representation of an enumerable recursive set by Sum of Four squares solver
in SymPy.
(#) New function that enumerates each element x of the complement set and applies it as a parameter to Sum of Squares Diophantine solver
to obtain the quadruple (a,b,c,d) such that a^2 + b^2 + c^2 + d^2 = x
(#) This is mostly partial recursive function and not necessarily a total recursive function(defined for all possible quadruples).
(#) logs for this have been committed to testlogs/
(#) SymPy has other diophantine solvers too which require more than one parameters.
(#) SymPy does not have exponential diophantine support yet which if available could construct an exponential diophantine for almost every
recursively enumerable set ('almost' because of MRDP theorem - there are diophantine equations for recursively enumerable non-recursive sets and undecidable)

Any partial function f:X - Y which maps a subset X' of X to Y, can be converted to a total function by mapping all elements in the unmapped complement domain set X - X' to an element y in a new extended codomain Y' = Y U {y} or an existing element in Y. This partial-turned-total function map can be interpolated to obtain a polynomial.

References:

492.1 Reduction of arbitrary diophantine equation to one in 13 unknowns -

[Matijasevic-Robinson] - http://matwbn.icm.edu.pl/ksiazki/aa/aa27/aa27125.pdf - This implies every effectively enumerable complement set can be represented by a diophantine in 13 unknowns.
492.2 Distinction between Enumerable and Effectively Enumerable - http://faculty.washington.edu/keyt/Effenumerability.pdf - "1) A set is enumerable if, and only if, it is the range of a total or partial function on the natural numbers. 2) A set is effectively enumerable if, and only if, it is the range of a total or partial effectively computable function on the natural numbers. 3) A function f is effectively computable if, and only if, there is a list of instructions giving a step-by-step procedure that will determine in a finite number of steps the value f(n) for any argument n for which f returns a value...." - Effectively enumerable sets are recursive sets created by a partial or total recursive function i.e an algorithm can always find a diophantine representation of the set. From previous result every recursive set can be represented by a diophantine in 13 unknowns. If each unknown has maximum limit l, then there are l^13 possible 13-tuples t(i) of unknowns which is the maximum cardinality of the effectively enumerable set - mapping is enumerated as f(t(0)),f(t(1)),...,f(t(l^13-1)). This kind of complementation has two phases:

(*) first 13-tuples are found by a diophantine solver and
(*) these tuples are listed as t(0),t(1),t(2),....

492.3 Function Spaces - Extending partial function to a total function - https://en.wikipedia.org/wiki/Partial_function#Total_function

---

493. (FEATURE-DONE) Computation Geometric Factorization - Binary Search for Tiles - Optimization - Benchmarks - Revised - 4 January 2018
- related to 480

(#) Factorization benchmarks for Tile binary search optimization have been redone on dual core (local[2]) after removing some print statements in python spark code which were CPU intensive.
(#) Following are the durations for factoring same 24 bit integer factorized previously after removing print statements - an improvement of more than 3X:

```
---------------------------------------------------------------
Factorization of 9333123 (24-bit) - without print statements
---------------------------------------------------------------
real  35m56.196s (Spark Duration = 1839.146seconds)
user  3m2.836s
sys   4m56.668s
---------------------------------------------------------------
Factorization of 9333123 (24-bit) - with print statements earlier
---------------------------------------------------------------
real    112m24.101s (Spark duration 12:21 to 14:02 = 101minutes = 6060seconds)
user    0m0.000s
sys     0m0.004s
================================================
Factors of 9333123 are (excerpt from logs):
================================================
...
Factor is =  219
Factor is =  1387
Factor is =  2243
Factor is =  4161
Factor is =  6729
Factor is =  6729
Factor is =  42617
Factor is =  127851
Factor is =  163739
Factor is =  491217
Factor is =  3111041
Factor is =  9333123
================================================
```

(#) Compressed Spark log for this factorization has been committed to python-src/testlogs/
(#) General Number Field Sieve algorithm is sequential, exponential in number of bits and requires O(2^(c(logN)^0.33*(loglogN)^0.67)) where c=1.901883 (for atleast one factor). For 24-bit (=logN) this is approximately k*128 time units for some constant k. But Computational Geometric Factorization Sieve finds all factors in O(logN*logN) time on multicore(PRAMs).

---

494. (THEORY and FEATURE-DONE) Complement Diophantine Map - Converts a Partial Function to Total Function - 6 January 2018

(#) Sum of Four Squares diophantine solutions are quadruples (a,b,c,d) which form a subset of all possible tuples and thus
complement function unknows-to-parameter map is partial.
(#) This is remedied by extending the parameter codomain by adding -1 as additional possible parameter value and mapping all unmapped tuples in domain to -1. Resultant map is Total defined for all possible unknown tuples.
(#) logs for this have been committed to testlogs/

---

495. (THEORY) Factoring as a service - General Number Field Sieve on Amazon EC2 cloud - RSA 512 bit - Relevance to Computational
Geometric NC-PRAM-Multicore Factorization Theoretical O((logN)^2) bound - related to 481 - 9 January 2018

Number field sieve has been implemented as Amazon EC2 cloud service. Benchmarks for RSA 512 bit achieve factorization in less than
4 hours applying CADO-NFS and MSieve NFS implementations optimized for Elastic Cloud. Instance type used is c4.8xlarge (each instance has two Intel Xeon E5-2666 v3 processor chips, with 36 vCPUs in a NUMA configuration with 60 GB of RAM) and maximum of 200 instances were in the cloud connected by Elastic Network Adapter(ENA). RSA 512 GNFS benchmark has some parallels in Computational Geometric Factorization - Most time intensive preprocessing step is to create tile segments in parallel by pixelating the hyperbolic arc. Presently this is implemented as SparkContext.parallelize(tilesegementslist) which creates tile segment RDDs in parallel on cloud reading tile segment array on flatfile. Ideally, each PRAM processor (assuming there are N/(logN)^k PRAMs) must read in a tile by computation and not from storage. For example in the algorithm mentioned in 481, p-th PRAM has to compute the tile segment interval (tilestart,tileend) locally in i-th iteration as tilestart=(x,N/x) and tileend=(x,N/x-N/x(x+1)) for x=(p+i*N/(logN)^k). Factoring 512 bit by Hyperbolic pixelation and sort-search of tiles theoretically requires constant*(512)^k parallel time units and 2^512/(512)^k PRAM processors. It has to be mentioned that real advantage of this Computational Geometric Factorization can be felt only for large integers on huge multicores and is by-and-large a theoretical fancy abiding by definition of NC ("Factorization is in NC") in the absence of a practical large scale cloud benchmark. Number of PRAMs required in definition of NC-PRAM equivalence is high even for work-optimal NC. Necessity of sorting the tile segments arises because successive segments though locally sorted ascending cannot be merged by splicing end-to-end directly because some elements in end of a segment could be larger than some elements in the beginning of next segment. Local tile search redresses this deficit.

References:

495.1 Factoring as a service - slides - http://crypto.2013.rump.cr.yp.to/981774ce07e51813fd4466612a78601b.pdf
495.2 Factoring as a service - https://github.com/eniac/faas - Parallelizing Number Field Seive - Polynomial selection, Sieving, Linear Algebra, Square root

- paper - [Luke Valenta, Shaanan Cohney, Alex Liao, Joshua Fried, Satya Bodduluri, Nadia Heninger - University of Pennsylvania] - https://eprint.iacr.org/2015/1000.pdf - "...The current public factorization record, a 768-bit RSA modulus, was reported in 2009 by Kleinjung, Aoki, Franke, Lenstra, Thomé, Bos, Gaudry, Kruppa, Montgomery, Osvik, te Riele, Timofeev, and Zimmermann, and took about 2.5 calendar years and a large academic effort [23],...We experimented with Apache Spark [37] to manage data flow, but Spark was not flexible enough for our needs, and our initial tests suggested that a Spark-based job distribution system was more than twice as slow as the system we were aiming to replace. Ultimately we chose Slurm (Simple Linux Utility for Resource Management) [36] for job distribution..."

---
496. (THEORY and FEATURE-DONE) Computational Geometric Factorization Update - Local Tile Computation and necessity for storage removed -
10 January 2018 - 30-bit integer dual core single node Spark Cluster benchmark - related to 495

As mentioned in previous section, reading tiles from storage in SparkContext.parallelize() is a serious bottleneck. In this commit, tile interval storage is obviated and each foreach() invokes a function which computes non-persisted tile interval locally and does binary search to print factors. SparkContext.parallelize() takes xrange() as argument. Python xrange() is an optimized implementation of list and returns XRange object.XRange object does not store all the elements in memory and is more like an iterator. A thirty bit integer has been factored and numbers are below:

```
---------------------------
Factorization of 921234437:
---------------------------
18/01/10 17:56:56 INFO Utils: /home/shrinivaasanka/Krishna_iResearch_OpenSource/GitHub/asfer-github-code/python-src/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been previously copied to /tmp/spark-9fba8482-8ca1-4022-9bcc-1c285a8e1f58/userFiles-3e4bf574-9b1a-44c5-8b96-3873e96dc6c3/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py
Factor is =  1
Factor is =  17
Factor is =  19
Factor is =  59
Factor is =  323
Factor is =  1003
Factor is =  1121
Factor is =  19057
Factor is =  48341
Factor is =  821797
Factor is =  918479
Factor is =  2852119
Factor is =  15614143
Factor is =  48486023
Factor is =  54190261
18/01/10 18:08:51 INFO PythonRunner: Times: total = 714516, boot = 649, init = 23, finish = 713844
```

Spark Duration: 18:08:51 - 17:56:56 = 535seconds

Spark-Python has RPC and serialization latencies and following numbers could be better if implemented in a different frameworks like Slurm and on a Gigabit ENA cloud. First non-trivial factor above was printed within few seconds and this benchmark is for sieving all factors. Time utility is misleading because it includes all Spark RPC overhead.

---
497. (THEORY and FEATURE-DONE) Computational Geometric Factorization - Non-Persistent Tile Segments - 30 bit Single Node Dual Core Spark Benchmarks - 11 January 2018 - related to 495

(#) Benchmarks for factoring another 30-bit integer 999994437 on single node dual core Spark cluster have been committed to python-src/testlogs/
(#) Similar to previous 30 bit integer, time utility prints 40 minutes 56 seconds while the actual Spark duration is between first and last factors:
18/01/11 13:34:34 INFO Utils: /home/shrinivaasanka/Krishna_iResearch_OpenSource/GitHub/asfer-github-code/python-src/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been previously copied to /tmp/spark-5630f619-b8a7-46b8-b15b-7296508792c5/userFiles-44c23be8-a7b3-48d3-bfd7-98acf1971e10/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py
==========
Factor is = 1
==========
==========
Factor is = 3
==========
==========
Factor is = 9
==========
==========
Factor is = 13
==========
==========
Factor is = 23
==========
==========
Factor is = 27
==========
==========
Factor is = 39
==========
==========
Factor is = 69
==========
==========
Factor is = 97
==========
==========
Factor is = 117
==========
==========
Factor is = 207
==========
==========
Factor is = 291
==========
==========
Factor is = 299
==========
==========
Factor is = 351
==========
==========
Factor is = 621
==========
==========
Factor is = 873
==========

==========
Factor is = 897
==========
==========
Factor is = 1261
==========
==========
Factor is = 1277
==========
==========
Factor is = 2231
==========
==========
Factor is = 2619
==========
==========
Factor is = 2691
==========
==========
Factor is = 3783
==========
==========
Factor is = 3831
==========
==========
Factor is = 6693
==========
==========
Factor is = 8073
==========
==========
Factor is = 11349
==========
==========
Factor is = 11493
==========
==========
Factor is = 16601
==========
==========
Factor is = 20079
==========
==========
Factor is = 29003
==========
==========
Factor is = 29371
==========
==========
Factor is = 34047
==========
==========
Factor is = 34479
==========
==========
Factor is = 49803
==========
==========
Factor is = 60237
==========
==========
Factor is = 87009
==========
==========

Factor is = 88113
==========
==========
Factor is = 123869
==========
==========
Factor is = 149409
==========
==========
Factor is = 261027
==========
==========
Factor is = 264339
==========
==========
Factor is = 371607
==========
==========
Factor is = 381823
==========
==========
Factor is = 448227
==========
==========
Factor is = 783081
==========
==========
Factor is = 793017
==========
==========
Factor is = 1114821
==========
==========
Factor is = 1145469
==========
==========
Factor is = 1610297
==========
==========
Factor is = 2848987
==========
==========
Factor is = 3344463
==========
==========
Factor is = 3436407
==========
==========
Factor is = 4830891
==========
==========
Factor is = 8546961
==========
==========
Factor is = 10309221
==========
==========
Factor is = 14492673
==========
==========
Factor is = 25640883
==========
==========
Factor is = 37036831

```
================================================
================================================
Factor is = 43478019
================================================
================================================
Factor is = 76922649
================================================
================================================
Factor is = 111110493
================================================
18/01/11 13:48:32 INFO PythonRunner: Times: total = 837634, boot = 641, init =
37, finish = 836956
18/01/11 13:48:32 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 1529
bytes result sent to driver
18/01/11 13:48:32 INFO TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1,
localhost, executor driver, partition 1, PROCESS_LOCAL, 6208 bytes)
18/01/11 13:48:32 INFO Executor: Running task 1.0 in stage 0.0 (TID 1)
================================================
Factor is = 333331479
================================================
18/01/11 13:48:32 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in
838426 ms on localhost (executor driver) (1/3)

which is almost 14 minutes ( 13:48:32 - 13:34:34 ). First non-trivial factor was
printed within 10 seconds. Time utility prints 40 minutes including overhead for
finishing Spark RPC tasks

--------------------------------------------------------------------------------
--------------------------------------------------------------
498. (THEORY and FEATURE-DONE) Computational Geometric Factorization - Non-
Persistent Tile Segments - 31 bit Single Node Dual Core Spark Benchmarks -
Primality Testing - Multiplicative Partition - Factorisatio Numerorum - 12
January 2018 - related to 495
--------------------------------------------------------------------------------
--------------------------------------------------------------
31-bit integer 2147483647 has been factorized by searching non-persisted
pixelated hyperbolic tile segments. Only trivial factors are found and is a
mersenne prime. This integer is the maximum permissible limit for xrange() and
printed by python sys module (sys.maxsize). Similar to integer partition
function which is number of ways of splitting an integer as sum of smaller
integers, multiplicative partition function f(n) = number of ways of unordered
factorizations of n. This function has an upperbound of : a{n} <= n(exp{{log n *
logloglog n}/{loglog n}})^{{-1+o(1)}}.

------------------------------------------------------------------------------
Factorization/Primality of 2147483647
------------------------------------------------------------------------------
18/01/12 14:33:12 INFO Utils: /home/shrinivaasanka/Krishna_iResearch_OpenSource/
GitHub/asfer-github-code/python-src/
DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been
previously copied to /tmp/spark-169ef9d3-56df-46eb-9b09-d63561491f87/userFiles-
9f55cc79-e7cc-46e6-bad7-540f950587b5/
DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py
================================================
Factor is =  1
================================================
18/01/12 15:02:57 INFO PythonRunner: Times: total = 1784724, boot = 643, init =
23, finish = 1784058
------------------------------------------------------------------------------
Spark Duration 15:02:57 - 14:33:12 = 29 minutes 45 seconds

Spark Logs have been committed to python-src/testlogs/

References:
```

498.1 Multiplicative Partition Function - ON THE OPPENHEIM'S "FACTORISATIO NUMERORUM" FUNCTION - [FLORIAN LUCA, ANIRBAN MUKHOPADHYAY AND KOTYADA SRINIVAS] - https://arxiv.org/pdf/0807.0986.pdf - "... The function f(n) is related to various partition functions. For example, f(2^n) = p(n), where p(n) is the number of partitions of n. Furthermore, f(p1p2 · · · pk) = Bk, where Bk is the kth Bell number which counts the number of partitions of a set with k elements in nonempty disjoint subsets. In general, f(p1^α1*p2^α2 · · · pk^αk) is the number of partitions of a multiset consisting of αi copies of {i} for each i = 1, . . . , k. Throughout the paper, we put log x for the natural logarithm of x. We use p and q for prime numbers and O and o for the Landau symbols. ..."
498.2 Multiplicative Partition Function - https://oeis.org/A001055
498.3 Bound for Multiplicative Partition Function - [Canfield-Erdos-Pomerance] - https://www.math.dartmouth.edu/~carlp/PDF/paper39.pdf
498.4 Eigth Mersenne Prime - 2^31 - 1 = 2147483647 - https://en.wikipedia.org/wiki/2,147,483,647 - this was largest known prime till 1867.
498.5 M31 - http://primes.utm.edu/curios/page.php/2147483647.html - The first prime that cannot be tested on 32-bit primality-check software.
498.6 Cryptography in NC0 - [Benny Applebaum, Yuval Ishai, Eyal Kushilevitz] - http://www.cs.technion.ac.il/~abenny/pubs/nc0.pdf, Hardness of factoring - [Emanuele Viola] - https://emanueleviola.wordpress.com/2018/02/16/i-believe-pnp/ - One way functions are in NC0 and there are PRGs for which output bit depends only on O(1) bits of seed implying factoring is hard for constant depth. Computational Geometric Factorization doable in O(logN^2) PRAM time in NC2, implies factoring can be made easy by parallelism and polylog depth circuits.
498.7 Binary Quadratic Diophantine Equations (BQDE) and BQDE for Factorization - [JC Lagarias] - https://arxiv.org/pdf/math/0611209 - Succinct Certificates for the Solvability of BQDE.
498.8 Polynomial Time Quantum Algorithm for Solving Pell's Equation - [Hallgren] - http://www.cse.psu.edu/~sjh26/pell.pdf - Integer Solutions to Pell's Equation reduce to Factoring - e.g Solving for x,y for known N in x^2 - Ny^2 = 1 factorizes N by rewriting as: (x^2 - 1) / y^2 = N => ((x+1)/y)((x-1)/y) = N. In this respect, Hallgren's algorithm is equivalent to Shor Quantum Factorization. Computational Geometric Factorization in NC-PRAM or Sequential Optimization in P could therefore imply Pell's equation is solvable by PRAM model and in NC or in P as follows:

```
(x+1)/y = a
(x-1)/y = b
2/y = a-b for factors a,b of N.
y = 2/a-b
=> (x+1) = 2a/(a-b)
=> x = 2a/(a-b) - 1
```

Historicity of this problem (Brahmagupta's Chakravala) and a polylog approximation based on Newton-Raphson square root recurrence is described in GRAFIT course material https://kuja27.blogspot.in/2018/04/grafit-course-material-newton-raphson.html

---
499. (FEATURE-DONE) Secure Neuro Currency Cloud Perfect Forward Move - OpenSSL client and server - 19 January 2018

Neuro Currency Cloud Perfect Forward Move socket code has been openSSL enabled:
- Makefile updated include -DOPENSSL #ifdef option for compiling SSL client-server headers
- license headers updated
- new header asfercloudmove_openssl.h has been added to repository for invoking openSSL client and server functions in opensslclient.h
and opensslserver.h
- An example fictitious X.509 cert.pem and key.pem have been created by openssl utility for certificate verification
- new headers opensslclient.h and opensslserver.h have been added to repository

which define openSSL client and server functions (these
 are reference examples in www.openssl.org Wiki changed for cloud move)
- logs for SSL cloud_move client and server have been committed to
cloud_move/testlogs/

---
500. (FEATURE-DONE) Searching Unsorted List of Numbers - Algorithm in GRAFIT
Open Learning Implemented - 21 January 2018

An algorithm to search list of numbers better than bruteforce search mentioned
in GRAFIT Course Notes:
https://github.com/shrinivaasanka/Grafit/blob/master/course_material/
ComputerScienceMiscellaneous/ComputerScienceMiscellaneous_CourseNotes.txt
has been implemented in NeuronRain AsFer.

This lifts one dimensional numbers to multidimensional tuples and creates
hashtables for each dimension of tuple. Lookup for a query number is
lookup of each digit in query in these hash tables in succession. If all hash
lookups succeed queried number exists in the unsorted list

This implementation pads each number with "#" so that it is right justified and
maximum number of digits is stipulated.

Searching Unsorted lists is the most fundamental open research problem and this
implementation uses Locality sensitive hashing as the hash table
. Python dictionaries are hash maps and not hash tables which support
collisions. Each digit in query is lookedup for nearest neighbours and if
 there is a match digitmatch is set to True. Logical AND of all digit matches is
returned as True/False.

---
501. (THEORY) Answer-Questioning (Reversal of Interview) and Recursive Gloss
Overlap Dense Subgraph Classification - 21 January 2018
- related to 412, 420

Recursive Gloss Overlap Definition Graph of a text previously has been
demonstrated to classify documents in unsupervised manner based on
dense subgraphs: centrality, k-cores, pageranks etc., Dense subgraphs of text
graph extract the essence of an article and highlight the keywords
which may or may not be present in the document. Keywords not present in the
document can be classes of a document because of the recursive
deep learning of definitions. From the keywords, a question or set of
permutations of questions can be constructed which the text answers.
Relevance of Question depends on the relatedness e.g Tensor Neuron of all
possible pairs of keyword class vertices.

Example:

If a text graph of an academic research text has been classified in following
word vertices of high core numbers
            [Prime, Factorization, Theorem, Composites, Diophantine,
Polynomials,...]
in decreasing value of centrality/k-core/pagerank and pairwise Tensor Neuron
potentials are ranked as below:
            Prime-Factorization = 0.34234
            Theorem-Composites=0.33333
            Diophantine-polynomials=0.221212
            ... and so on
then question(s) can be constructed based on previous ranking of Tensor
relatedness as:

Does this article discuss relation between [Primes] and [Factorization]?
....

Reference:

501.1 CALO, SIRI, PAL - https://pal.sri.com/architecture/ - Cognitive Assistants, Question Answering

---
502. (FEATURE-DONE) Scheduler Analytics - Interprocess Distance Computation by DictDiffer - 24 January 2018
---
1. Representing OS Process Information as Feature Vectors has been mentioned as Software Scheduler Analytics usecase in http://neuronrain-documentation.readthedocs.io/en/latest/
2. This commit implements a new python function which reads process info as a dictionary from psutils. Psutils has an option to read individual process metrics or to read the dictionary in its entirety.
3. Psutils per-process dictionary has all the basic details pertaining to instantaneous resource consumptions of a process
4. These per-process features are collated in an array and are json.dump()-ed as strings in asfer.enterprise.encstr.scheduleranalytics file which can be used as input by clustering/classification NeuronRain-AsFer C++ implementations.
5. logs for this have been committed to software_analytics/testlogs/DeepLearning_SchedulerAnalytics.log.24January2018
6. Distance between two process feature dictionaries/vectors are printed by DictDiffer which has been imported and length of the diff is
 the distance function between any two processes.

---
503. (FEATURE-DONE) Scheduler Analytics - Process kNN classification and K-Means clustering, Process Dict Hashing - 25 January 2018
(Only in NeuronRain Enterprise - GitHub)
---
1. python-src/software_analytics/DeepLearning_SchedulerAnalytics.py has been updated to define a getHash() function similar to Streaming_<algorithm>.py implementations, which creates an MD5 hash of process psutils dictionary string.

2. This hashing of dictionary is because of huge size of process dictionary storing of which requires few KBs for each process. Computing distance (Levenshtein) of these strings is again memory intensive.
3. DictDiffer distance implemented in python-src/software_analytics/DeepLearning_SchedulerAnalytics.py is not directly invocable in C++ and diff between process dictionary strings is also huge in KBs.
4. Succinct representation of process dictionary string - Fingerprinting - is therefore a necessity to optimize space. MD5 binary hash digest of the process dictionary string is written to python-src/software_analytics/asfer.enterprise.encstr.scheduleranalytics prefixed by process name and process id , instead of complete chunk.
5. cpp-src/asferkNNclustering.cpp and cpp-src/asferkmeansclustering.cpp are updated for EOF check and number of clusters has been increased to 10 in kNN classifier.
6. asfer binary has been rebuilt and asfer.conf has been updated to do clustering and classification.
7. process statistics dataset - cpp-src/asfer.enterprise.encstr, cpp-src/asfer.enterprise.encstr.clustered and cpp-src/asfer.enterprise.encstr.kNN have been updated.

8. asfer kNN classification and KMeans clustering has been executed on this encoded process statistics dataset
9. logs have been committed to cpp-src/testlogs/asfer.SchedulerAnalytics.KMeansAndkNNClustering.log.25January2018 (Scheduler Analytics) and cpp-src/testlogs/asfer.SchedulerAnalytics.KMeansAndkNNClustering.log.25January2018 (kNN classification and KMeans clustering) - for 194 processes
10. logs for clustering and classification imply similar processes flock together - e.g excerpt from KMeans cluster logs show lot of kworkers in same cluster:

```
===========
Cluster 6
===========
encoded string
[5574:bash:0b1101110111101000111110100011000011111011100001101001000100110011010
0110110000011100101001011000001100001100010110011001001100000010000111101000101010
0000000000]
encoded string
[139:kworker/u4:4:0b1100111010110010011101010001000010011101100110010011110011001
0011011000110011111111010110110010101111000000000101101111110110000101010011101010
0100010010010010001]
encoded string [145:usb-
storage:0b1100001010101110000111110011101100010111011101010000000001100110001100
1001101101010011010100101000000110111110001001011101011100010010101111011010001111
1000001]
encoded string
[35:fsnotify_mark:0b1011011110000001111010011000110101011010001100000010010001010
0100110101111101001111000011100101000111000111001000000101000011000101110101100100
1010100001101101110]
encoded string [2769:cups-
browsed:0b1110011010000000111100101100010010111011110111111011110011111010110100100
0000110011101010011011101111100101010001001111101001100010110100010011101110011000
000000100]
encoded string
[5673:bash:0b1000000101010011010111000111111011100010100001101000010111101011000
0011010110100100001100010111000011010001011101000100110100000011011011101010111101
101011100011]
encoded string
[49:acpi_thermal_pm:0b1110010100000011111010000001000010111100000100000110010011110
1010111001010000001011101111100100010100101111101000011100000110101011001100011000
0110100011100001010100]
encoded string
[6220:kworker/u4:1:0b1011111100100011001100000010111100111111110001000101011110100
1011100110101011010101011110100001110010101100011011111111001010001100011111011110
1010110011100110010 0]
encoded string
[5977:kworker/0:0:0b1101000010010001011101100100001011011011011101100111100101110
0000111111011101100101001011010101100000000111001101011011011000001101110000111100
1011010100001100100 10]
encoded string
[5905:kworker/u4:0:0b1011011110010010111000111001101101001000001010100100100101000
1110000011110011100110111010010110100000101110101000001001000110101010011000100000
1111111001001111000 10]
encoded string
[2433:kworker/u5:0:0b1100100101100010100101000010001010111010010111110110011110100
1110000111011011010111001001010100110101111101100010111000000100100011010010001000
1110111100001100001 01]
encoded string
[2439:kworker/u5:2:0b1110100001111010101011110110101110101101001100000011100011110
0001101000011001111101100101110100010011010110001010110001010001001000101111011011
1110010110100000 1111]
encoded string
[5:kworker/0:0H:0b1000110000100110011111100011111100100001000110011011001000111000
0101110010011010110111010110110010100111001110111010111111111111101110111000010010
```

000100010010011000]
encoded string
[9:migration/0:0b1100110100001111011000100001110011000010111110010101001011110010101101000101101100111001100110111010110111101111100100110111101001000010100100000011001001000]
encoded string
[6300:kworker/0:1:0b1100011001101110000000000100000100011100110110001011101110011010000010111110010110100000000011010101010001011001011010000111110101100110001101000111010 0]
encoded string [5065:upstart-dbus-bridge:0b1010000111111000000000100111010110100110001100101110101010111001100011000111111110101110100001010011110011100001101001111101101100010101100100100101100011101000]
==========

Reference:

503.1 Randomized Algorithms - [Rajeev Motwani, Prabhakar Raghavan] - Pages 168 and 214 - 7.4 Verifying Equality of Strings and 8.4 Hash Tables for string fingerprinting

---
504. (FEATURE-DONE) Streaming Algorithms - Encoded Strings Data Source - 26 January 2018
(For MD5 hash and other Encoded Strings)
---
1. python-src/Streaming_AbstractGenerator.py iterator has been updated for new data storage "AsFer_Encoded_Strings" and data source "NeuronRain"
for streaming encoded strings created by NeuronRain AsFer
2. Streaming algorithm implementations have been updated to stream data from this new datasource and logs have been committed to testlogs/

---
505. (FEATURE-DONE) Scheduler Analytics - Sequence Mining - 29 January 2018
(Only in NeuronRain Enterprise - GitHub)
---
1. Sequence Mining has been executed on Scheduler Analytics process statistics (few processes and max length set to 5).
2. Candidate initial item set has been changed to alphanumeric
3. Declared boolean flag for choosing between MD5 hash encoding of process dicts or plain string representation of process dict
4. updated software_analytics/asfer.enterprise.encstr.scheduleranalytics
5. Sequence Mined Scheduler Analytics has been committed to testlogs/SequenceMining.log.SchedulerAnalytics.29January2018

---
506. (THEORY) Computational Geometric Factorization - Feasibility of Sequential Optimization - related to 486 - 6 February 2018, 1 September 2018

Doing away with Parallel RAMs in Computational Geometric Factorization by breaking the hyperbolic arc bow into two sets of tilesegments
and searching their concatenations which are implicitly sorted ascending, has problems because concatenation of multiple tile segments may not be sublinear.
Concatenating tile segments can be likened to concatenation of multiple strings.
Concatenation of two strings can be done
in O(logN) time by using Rope Strings which are tree representations of strings (logarithmic in length of strings and not in number of
strings). But concatenation of multiple strings/tiles logarithmic time in number

of strings is still open.

Following sequential optimization tries to find feasibility of doing Sequential Factorization without PRAMs and concatenation of tile segments:

number_to_factorize=N
factor_candidate=N/2

Loop for subsegment1 tree:
---
while(factor is not found)
{
	1. Find the tile segment/interval containing factor candidate.
	[2. Split the candidate tile segment to two subsegments - subsegment1 contains points which are less than previous segment's end point,subsegment2 contains points which are greater than previous segment's end point.]
	3. Binary Search each node in subsegment1 tree for factor points (p,q) (N=pq) - There are 3 possibilities:
	3.1 All points in this subsegment1 tree node are less than N => Binary Search has to be done on right subsegment1 subtree recursively, factor_candidate is updated
	3.2 All points in this subsegment1 tree node are greater than N => Search has to be done on left subsegment1 subtree recursively, factor_candidate is updated
	3.3 N is present in this subsegment1 tree node and factor point N=pq is found
}

Loop for subsegment2 tree:
---
while(factor is not found)
{
	4. Find the tile segment/interval containing factor candidate.
	[5. Split the candidate tile segment to two subsegments - subsegment1 contains points which are less than previous segment's end point,subsegment2 contains points which are greater than previous segment's end point.]
	6. Binary Search each node in subsegment2 tree for factor points (p,q) (N=pq) - There are 3 possibilities:
	6.1 All points in this subsegment2 tree node are less than N => Binary Search has to be done on right subsegment2 subtree recursively
	6.2 All points in this subsegment2 tree node are greater than N => Search has to be done on left subsegment2 subtree recursively
	6.3 N is present in this subsegment2 tree node and factor point N=pq is found
}

7.Above algorithm is Depth-2 Two Level Binary Search:
7.1) First binary search finds the subsegment node in subsegment trees in O(logN)
7.2) Second binary search searches within subsegment node in O(logN)
and thus requires O(logN*logN) sequential time.

8. Sets of subsegment1(s) and subsegment2(s) constitute 2 binary search trees of segments, but these two search trees are not physically created. Binary search of tile containing factor candidate dynamically creates treelike traversal.

9. Finding tile segment containing a factor candidate point is non-trivial planar point location problem in general. But Following algorithm finds the interval containing the point in O(1) time specific to hyperbolic pixelated tiling:
Sum of lengths of k consecutive hyperbolic pixelated tiles = N/(1*2) + N/(2*3) + N/(3*4) + ... + N/(k*(k+1))

= N(1/1 - 1/2 + 1/2 - 1/3 + 1/3 - 1/4 + ... + 1/k - 1/(k+1))
= N(1 - 1/(k+1))
= Nk/(k+1)

Tile containing point x:

Nk/(k+1) < x < N(k+1)/(k+2)
k < x/(N-x)

integer round-off of k is the index of tile interval containing x.

10. Previous sequential optimization of O((logN)^2) though removes PRAMs, proves only that Factorization is in P. Computational geometric factorization by PRAMs is still a better result because it implies Factorization is in NC from PRAM-NC equivalence despite requirement of high number of parallel processors.

11. For each segment k in loops previously, 2 subsegments of it have following interval endpoints - (xleft,yleft,xright,yright):

subsegment1: (N/(k+2),(k+1),N/(k+2)+delta,(k+1))
subsegment2: (N/(k+2)+delta,(k+1),N/(k+1),(k+1))

and delta=N/((k+1)(k+2))

12. CAVEAT: Two factor points (x1,y1) and (x2,y2) can equal by x1*y1=x2*y2 but these two points are located in different subsegments of different tiles because points within subsegment1 or subsegment2 of a tile segment have same y-coordinates. Previous inequality applies only to a binary search segment tree comprised of subsegment2(s) of all segments by implicit geometric spatial sortedness i.e end of subsegment2 in previous segment is less than start of subsegment2 in next segment. Set of subsegment1(s) need not be implicitly sorted and thus can not be searched by a segment tree constructed sequentially. This renders previous optimization feasible only to a subset of integers. Algorithms for Parallel RAM construction of segment tree described previously are therefore necessary to find factor points in polylogarithmic time and thus in NC and all these algorithms require parallel k-mergesort. This sequential algorithm has been mentioned only as an optimization.

---

13. Tile Summation and Wavelet trees:

Sum of lengths of first n tile segments of pixelated hyperbolic curve derived previously:

N/[1*2] + N/[2*3] + ... + N/[n*(n+1)] = Nn/(n+1)

which can be equated to sum of distances between first m prime factors by Hardy-Ramanujan Theorem:

Nn/(n+1) = mN/kloglogN

=> n = m/(kloglogN - m) = number of tiles till m-th prime factor. Sum of lengths (N/x(x+1) each) of these n tiles (=O(loglogN)) till m-th prime factor finds approximate m-th prime factor. Constructing wavelet tree for concatenated unsorted string of tile segments of epsilon radius around approximate factors and doing select(N) on the wavelet string finds atleast one factor location. Geometric intuition: Tile summations are hyperbolic pixel polygons and if vertices of this polygon - endpoints of concatenation - coincide with factor points on x-y axis grid, factors are found. Accuracy depends on precision of tile summation and this is only an optimization. If epsilon radius centered around each approximate prime factor is e, total length of concatenated unsorted tile for all prime factors is: O(2*e*loglogN). Sequential Construction and select() for wavelet tree for this string is:

O(2eloglogN * log(2eloglogN)) + log(2eloglogN) = O(e*loglogN * log(e*loglogN)) + log(e*loglogN)

For sequential polylogarithmic factorization, e has to be upperbounded by O((logN)^l).

---

507. (THEORY) Tournament Graph Election, Coloring, Intrinsic Merit, Partitions and Bell Number - 8 February 2018

Tournament Graph of n vertices has n(n-1)/2 edges (complete). Each vertex of the tournament graph is uniquely colored by total of n colors.
Each edge (v1,v2) is the contest between vertices v1 and v2. Each edge of the tournament graph is progressively colored by the color of winning vertex of the edge endpoints. Thus at the end of the tournament, n(n-1)/2 edges are partitioned into n monochromatic sets of edges. Number of
all possible partitions of a set is the Bell number. Vertex corresponding to Color of the biggest part in this edge set partition is the final winner. Any partition which has unequal parts has biggest part. Thus number of partitions of edges having atleast one biggest part = BellNumber - 1 because there is exactly one partition of n(n-1)/2 edges into n sets of equal size (n-1)/2. Tournament Graph Election is an alternative social choice function and does not involve Majority voting. Win in contest between two vertices of an edge can be defined as the vertex having greater Intrinsic Merit (In social networks this is the standard Intrinsic Fitness of a vertex). This formalizes the intrinsic merit usecases mentioned in NeuronRain Documentation and FAQ in http://neuronrain-documentation.readthedocs.io/en/latest/ . One additional usecase where majority voting falters and Intrinsic Merit is a necessity is the indispensability of experimental evidences in exact sciences. For example, in archaeology/history/mythology, "an artefact existed few millenia ago" is quite presumptive hypothesis which cannot be proved by mere majority voting in contemporary era. In tournaments involving knock-out(s), each elimination removes (n-1) edges and n/2 rounds decides winner.

References:

507.1 Tournament Trees - Minimum Comparison Selection - [Don Knuth] - The Art of Computer Programming - Volume 3
- Page 207-209 - Section 5.3.3 - Kislitsyn Theorem - In a knockout tournament tree of n players, minimum number o
f comparisons required to find second best player is n - 2 + ceil(logn). For first best player minimum number of comparisons required is n - 1.

---
508. (FEATURE-DONE) Streaming Analytics Abstract Generator Update - Socket Streaming Datasource Added - 8 February 2018

1.Streaming Abstract Generator facade/generator pattern implementation has been updated for a new "Socket Streaming" data storage/data source.
2.Constructor Datasource arg is the remote host and port is hardcoded to 64001 (one more than kernel_analytics driver streaming port)
3.Python socket has been imported and socket client code has been added to __iter__()
4.As example, Hyper LogLog Counter Streaming implementation has been updated to read streaming data from remote streaming webservice host
5.Logs for this have been committed to testlogs/
6.Example webserver commandline:
```
        nc -l 64001
        ><data>
```

---
509. (FEATURE-DONE) Scheduler Analytics - Socket Streaming Server - Decorator pattern implementation - 9 February 2018
---
1. New Socket Streaming Server for Scheduler Analytics has been implemented in python-src/software_analytics/SchedulerAnalytics_WebServer.py
2. This invokes get_stream_data() function in python-src/software_analytics/DeepLearning_SchedulerAnalytics.py

3. get_stream_data() function is decorated by a Decorator class implemented in a new file python-src/webserver_rest_ui/NeuronRain_Generic_WebServer.py
4. SocketWebServerDecorator implemented in python-src/webserver_rest_ui/NeuronRain_Generic_WebServer.py is generic and overrides
 __init__() and __call__() functions. __call__() invokes the decoratee function get_stream_data() specific to Scheduler Analytics.
5. SocketWebServerDecorator can decorate any other streaming datasource, not just scheduler analytics - decoratee function has to be implemented
 accordingly.
6. Logs for Socket Streaming Server have been committed to python-src/software_analytics/testlogs/SchedulerAnalytics_WebServer.log.9February2018 and example Socket Streaming Client HyperLogLogCounter logs are at python-src/testlogs/Streaming_HyperLogLogCounter.log.9February2018
7. Global configs for socket streaming server host and port (64001) have been set in a new config file python-src/software_analytics/SchedulerAnalytics_Config.py
8. Known issue: There seems to be a random iterator disconnect because of psutil process_iter() process statistics streaming delay

---
510. (FEATURE-DONE) Scheduler Analytics Socket Streaming Decorator - Psutil iterator frequent disconnects resolution - 11 February 2018
---
1. SocketWebServerDecorator frequently and periodically throws "NoneType object not callable" exception.
2. This exception happens after every 215 or 217 processes (not sure what this magic number is)
3. Because of this try/except error handling has been added throughout decorator code
4. Socket listen queue size has been increased to 100
5. datasourcefunc() is re-called once an exception is thrown in a loop - a palliative cure
6. After this 215 barrier is breached. Logs for 558 processes streamed by Socket Server Decorator and received by Streaming Abstract
Generator client have been committed to python-src/software_analytics/testlogs/SchedulerAnalytics_WebServer.log.11February2018 and python-src/testlogs/Streaming_HyperLogLogCounter.log.11February2018

---
511. (FEATURE-DONE) Approximate 3SAT Solver Randomized Rounding Update - NumPy random choice() replacing permutation() - 27 February 2018
---
1. CNFSATSolver.py - function creating random 3SAT instances has been updated to invoke NumPy random choice() instead of
permutation() without replacement - equivalent to nP3 per clause for n variables.
2. Logs for 16 variables - 16 clauses and 18 variables - 18 clauses have been committed to testlogs/CNFSATSolver.16variables16clauses.log2.27February2018 and testlogs/CNFSATSolver.18variables18clauses.log.27February2018
3. Numpy random choice() is based on Uniform distribution.
4. It is remarkable to note that observed probability average and Random Matrix probability 1/sqrt(m*n)
 are strikingly equal - a confirmation of the least squares randomized rounding and relaxation SAT solver's accuracy
5. Logs for 16*16 and 18*18 have few hundreds and thousand plus random 3SAT instances which are solved 100%

---
512. (FEATURE-DONE) ConceptNet5 Update - REST endpoints changed - 27 February 2018
---
1. ConceptNet5 rest_client.py has been imported which already wraps REST methods and endpoints
2. REST endpoints for search,lookup and query have been updated for ConceptNet 5.5 which were 5.4 earlier

---
513. (FEATURE-DONE and THEORY) Approximate SAT Solver and ConceptNet REST client updates - 21 variables,21 clauses LSMR and Common Ancestor Distance in ConceptNet5 - 28 February 2018

1.Least Square SAT solver has been updated to print more readable debug messages and highlight MAXSAT approximation ratio
 average for SATs solved thus far
2.Number of variables and clauses is set to 21,21 and MAXSAT ratio is 98% for 34 random 3SATs
3.Logs for 21,21 has been committed to testlogs/CNFSATSolver.21variables21clauses.log.28February2018 which shows the observed literal
probabilities and random matrix 1/sqrt(m*n) literal probability agreeing well.
4.This is because of uniform unbiased (mostly) choice() when number of clauses = number of variables causing 1/sqrt(m*n) = 1/n
5.For unequal clauses and variables numbers, an epsilon biased PRG implementation in linux kernel is necessary (random.c might have to be
 rewritten)

6.ConceptNet5 client has been updated to include a related() function wrapper which invokes /related REST endpoint for finding similar concepts.
7.ConceptNet5 finds related concepts by word embeddings implementation of word2vec which maps each word to a vector and similar words/concepts
are clustered together in this vector space. More precisely, word distances are represented as a word-word 2 dimensional matrix/graph
and sum of path edge weights between word pair is the distance.
8.For finding distance between two concepts a common ancestor algorithm has been implemented in new conceptnet_distance() function which recursively invokes related() to deep learn concepts and grows paths between the two concept endpoints. This recursion stops when paths grown from both ends meet i.e when there is a common ancestor (inspired by Savitch and USTCONN in logspace theorems).
9.Common ancestors are printed in end and distance between the concepts is also printed
10.Some example conceptnet_distance() invocations have been captured in ConceptNet/testlogs/ConceptNet5Client.log.28February2018

References:

513.1 ConceptNet 5.5: An Open Multilingual Graph of General Knowledge - [Robert Speer,Joshua Chin,Catherine Havasi-Luminoso Technologies] - https://arxiv.org/pdf/1612.03975.pdf
513.2 ConceptNet blog - https://blog.conceptnet.io/
513.3 Microsoft Concept Graph - Probase - https://concept.research.microsoft.com/Home/Introduction - similar to ConceptNet

---
514. (THEORY) ConceptNet, Graph Tensor Neuron Network Intrinsic Merit, Word2Vec Word Embeddings - 2 March 2018 - related to 412, 513

ConceptNet5 /related REST endpoint retrieves concepts similar to a concept and ranks them by word2vec word embedding similarity score. Recursiv
e Lambda Function Growth algorithm mentioned previously, grows lambda functions for random walks in a text definition graph created by Recursiv
e Gloss Overlap and word-word similarity/relevance is formalized by Neural Tensor Network. ConceptNet /related word2vec word embedding word-word similarity scores are perfect fit for Neuron Tensor Network entity relation potentials. Per random walk lambda function composition tree composes the neuron tensor potentials of all relations in the random walk. Composition trees are ranked by these composed potentials and Lambda composition tree of maximum potential best approximates the meaning of the Natural Language Text. This implies ConceptNet itself is one huge Graph Tensor Neuron Network Monolith.

Rationale for creating random walks in text definition graph is: Process of Human Text Comprehension is simulated by connecting concepts in text in various possible random walks of finite lengths, creating a tree of their meaning as lambda function composition and choosing the most rewarding random walk ranked by composed potentials. Most rewarding random walk must contain the word vertices of high core numbers (or centralities/classes of the text) because most number of paths in the graph go through these hub vertices (Section 5.16 in https://tac.nist.gov/publications/2010/participant.papers/CMI_IIT.proceedings.pd
f proves this notion for Recursive Gloss Overlap graph in terms of WordNet relations). Cycles in the definition graph have also been experimentally found to approximate the text well. Random walks/Cycles in the definition graph have close resemblance to Word-Chain Theory in Psycholinguistics which postulates Brain stores the information as chain of words. Word-Chain Theory was dismissed by the Deep Structure theory of Chomsky. Lambda Function Composition of Random Walks/Cycles combines Word-Chain and Deep Tree Structure into one - converts list of words into a meaning tree.

References:

514.1 Neural Tensor Network - [Socher-Chen-Manning-Ng] - https://nlp.stanford.edu/pubs/SocherChenManningNg_NIPS2013.pdf - Section 3.1 - "...The goal of our approach is to be able to state whether two entities (e1, e2) are in a certain relationship R. For instance, whether the relationship (e1, R, e2) = (Bengal tiger, has part, tail) is true and with what certainty. ... The Neural Tensor Network (NTN) replaces a standard linear neural network layer with a bilinear tensor layer that directly relates the two entity vectors across multiple dimensions. ..."
514.2 The Language Instinct - [Steven Pinker] - Chapter 4 - How Language Works - Word Chains and Chomsky
514.3 Chomsky-Norvig Debate on Algorithmic Versus Statistical Learning - http://daselab.cs.wright.edu/nesy/NeSy13/norvig.pdf - 98% learning models are probabilistic/statistical while 2% are algorithmic. Chomsky's viewpoints:

<quote>

- Statistical language models have had engineering success, but that is irrelevant to science.
- Accurately modeling linguistic facts is just butterfly collecting; what matters in science (and specifically linguistics) is the underlying principles.
- Statistical models are incomprehensible; they provide no insight.
- Statistical models may provide an accurate simulation of some phenomena, but the simulation is done completely the wrong way; people don't decide what the third word of a sentence should be by consulting a probability table keyed on the previous two words, rather they map from an internal semantic form to a syntactic tree-structure, which is then linearized into words. This is done without any probability or statistics.
- Statistical models have been proven incapable of learning language; therefore language must be innate, so why are these statistical modelers wasting their time on the wrong enterprise?

<unquote>

Recursive Gloss Overlap and Recursive Lambda Function Growth are Algorithmic Language Learning Models.

514.4 Gold's Theorem - www.lps.uci.edu/~johnsonk/Publications/Johnson.GoldsTheorem.pdf and limit on learnability - language L has infinite elasticity if there exist subsets T1 in T2 in ... ad infinitum such that Ti in Li and T(i+1) not in Li for set of languages Li and Lim Ti = L . Languages of infinite elasticity are not learnable.
514.5 Power Law Distribution, Scale-Invariance, Small Word Phenomenon in Roget's Thesaurus and WordNet -Section 4.3 - http://web.mit.edu/cocosci/Papers/03nSteyvers.pdf - Roget's Thesaurus and WordNet follow Pareto's Power Law Distribution i.e P(k) = k^(-e) where P(k) is the probability of vertices having k incoming links and e is the scale exponent. Power Law implies large number of nodes have small incoming links and small number of nodes have high incoming links. Both Roget's Thesaurus and WordNet have scale exponent ~3, high sparsity, very short average distances among words(5.6 and 10.56), single connected component containing almost all words, high local clustering.
514.6 Pareto's 80/20 rule, Scale-Invariance in phase-transition - Chapter 6: Sixth Link - Linked - [Albert Laszlo Barabazi]

---
515. (THEORY and FEATURE-DONE) Recursive Lambda Function Growth - ConceptNet5 support, Per random walk
Lambda Function Composition Tree Graph Tensor Neuron Network Intrinsic Merit - 3 March 2018 - related to 514

Recursive Lambda Function Growth implementation has been updated to print per random walk lambda function
composition tree and its Graph Tensor Neuron Network Intrinsic Merit. Similarity has been made configurable by a flag to choose between 2 Antology Nets - WordNet and ConceptNet. Logs for this have been committed to testlogs/RecursiveLambdaFunctionnGrowth.log.GraphTensorNeuronNetwork.3March2018. Logs show a striking similarity between Maximum Intrinsic Merit Random Walk Composition and Centralities/Classes of the definition graph (excerpts below):
...
grow_lambda_function3(): Maximum Per Random Walk Graph Tensor Neuron Network Intrinsic Merit : (u'(present(etc,(infrastructure(country,(urban(area,(Mumbai(city,largest)))))))', 3.790350877192983)
...
==========
Unsupervised Classification based on top percentile Core numbers of the definition graph(subgraph of WordNet)
==========
This document belongs to class: Mumbai ,core number= 12
This document belongs to class: infrastructure ,core number= 9
...
which is an evidence to the fact intrinsic merit peaks for random walks through high core numbered hub word vertices. Another aspect of ConceptNet's relevance to Graph Tensor Neuron Network is the strong resemblance of word2vec word embeddings to Connectomes in Computational Neuroscience (mentioned in references). Connectomes are the wiring diagrams for neurons in brain. Word2Vec clusters similar words of a text in vector space forming a clique. These word cluster cliques are wired together into a text graph. Thus Graph representation of Text and Neuron Cliques and Connectome Information Propagation Among Cliques of Neurons have uncanny similarities.

References:

515.1 Cliques and Cavities in Human Brain - [Ann Sizemore, Chad Giusti, Ari Kahn, Richard F. Betzel, Danielle S. Bassett] - https://arxiv.org/abs/1608.03520

515.2 Algebraic topology and Computational Neuroscience - https://www.technologyreview.com/s/602234/how-the-mathematics-of-algebraic-topology-is-revolutionizing-brain-science/
515.3 Brain Graphs of Women are better expander graphs than men - http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130045 - Brain Graphs (or) Connectomes of Brain have been analyzed in computational neuroscience as Graph Expanders. Better Expander implies the Brain Graph or Graph of Neurons have better epsilon-Expansion trait i.e each vertex subset S of brain graph has epsilon*|S| incident edges. - "...Expander graphs and the expander-property of graphs are one of the most interesting area of graph theory: they are closely related to the convergence rate and the ergodicity of Markov chains, and have applications in the design of communication- and sorting networks and methods for de-randomizing algorithms [25]. A graph is an ε-expander, if every—not too small and not too large—vertex-set S of the graph has at least ε|S| outgoing edges (see [25] for the exact definition). Random walks on good expander graphs converge very fast to the limit distribution: this means that good expander graphs, in a certain sense, are “intrinsically better” connected than bad expanders. It is known that large eigengap of the walk transition matrix of the graph implies good expansion property [25]. We have found that women’s connectomes have significantly larger eigengap, and, consequently, they are better expander graphs than the connectomes of men. For example, in the 83-node resolution, in the left hemisphere and in the unweighted graph, the average female connectome’s eigengap is 0.306 while in the case of men it is 0.272, with p = 0.00458..." - High expansion implies least energy by [Barabasi-Bianconi] Bose-Einstein condensation and vertices are able to attract high number of links. In the context of random walks in Recursive Lambda Function Growth previously, assuming the lambda function tree as logical connectome - because role of neurons is simulated by the lambda functions - random walks on high expanding, least energy definition graph should converge faster theoretically.
515.4 Mapping Neurone Synapses to Connectomes - https://neurophys.gu.se/digitalAssets/1553/1553052_richard_dybowski.pdf - Workflow for Segmentation of fMRI images, Identifying membranes, Inferring Connectomes by Deep Learning.
515.5 Measuring Intrinsic Fitness in WWW and Scientific Publications (Bianconi-Barabasi Model) - http://networksciencebook.com/6#bianconi-model - Sections 6.3 - Case studies of journals (Nature,PRL,PNAS,Science etc.,) and Bose-Einstein Condensation in some networks - Section 6.4
515.6 Analysis of Brain Graphs or Connectomes - https://arxiv.org/pdf/1105.4705.pdf - Network Motifs - "...Similarly, human structural connectivity between brain regions shows small-world properties with a small-worldness S of 10.6 (Text S2, Hagmann et al., 2008). For human functional connectivity between brain regions, the clustering coefficient is 53% (22% in random networks) and the path length 2.5 (2.3 in random networks) (Achard et al., 2006)..."

---

516. (FEATURE-DONE) SAT Solver - verbose print and iterations reduced - 4 March 2018

Least Squares LSMR SAT solver implementation has been updated to print LSMR internal calculations. Maximum iterations, conlim, atol, btol parameters have been set to least values for reducing latency. But this reduces MAXSAT accuracy to ~94% which is still above 88% (7/8 approximation). SciPy Sparse dsolve.spsolve() was tried for solving least squares by csc_matrix and there were nan errors.Finally, LSMR looks to be the best of the lot. Logs for few random 3SATs of (21 clauses, 21 variables) have been committed to testlogs/CNFSATSolver.21variables21clauses.log.4March2018. Observed average probability of choosing a literal again agrees quite well to 1/sqrt(m*n) Random Matrix theoretical probability as gleaned from logs.

---

517. (THEORY) Majority+Voter Composition Hardness Amplification Lemma, Sensitivity Conjecture - Updated Hardness Bound - related to 318,355 - 5 March 2018

Hardness Amplification for Majority+Voter Composition has been described earlier by adapting XOR Lemma to Majority function as:

Hardness of Maj+voter composition = [c/sqrt(n*delta)] + [sum(column2 error entries)] - [sum(column3 no error entries)]

Hardness of voter function delta

which is based on the sudivided BP* error scenarios matrix mentioned earlier and described again below:

x	f(x) = f(x/e)	f(x) != f(x/e) Noise
x in L, x/e in L	No error	Error
x in L, x/e not in L	Error	No error if f(x)=1,f(x/e)=0 else Error
x not in L, x/e in L	Error	No error if f(x)=0,f(x/e)=1 else Error
x not in L, x/e not in L	No error	Error

x	f(x)
Randomized Decision tree evaluation	No error	Error

This matrix picturises the various false positives and false negative errors possible in majority voting. Language L denotes voting pattern for candidate 0 to win (and its complement is voting pattern for candidate 1). Correlation x/e is the flip in voting pattern. Previous matrix divides BP* errors into 2 dimensions based on 1) whether the voting pattern is valid 2) Noise sensitivity/ stability . For example a voting pattern x for candidate 0's win is correlated/flipped to x/e so that candidate 1 wins and input to Majority function f(), but yet majority function treats them with equal outcome - in matrix this error scenario is: x in L, x/e not in L and f(x)=f(x/e). Other scenarios follow this convention. Based on this error matrix, Hardness

amplification ratio derived previously can be rewritten as:

$$\frac{\text{Hardness of Maj+voter composition}}{\text{Hardness of voter function}} = \frac{[\text{sum(column2 entries)}] + [\text{sum(column3 entries)}] - [\text{sum(all no error entries)}]}{\text{delta}}$$

But Stability(f(x)) = Pr(f(x)=f(x/e)) - Pr(f(x) != f(x/e)) from definition of Noise stability => Pr(f(x)=f(y)) = (1+Stability)/2.
Sum(column2 entries) = Pr(f(x)=f(y)) = (1+Stability)/2.
Sum(column3 entries) = NoiseSensitivity
=> Hardness of Maj+Voter composition = (1+Stability)/2 + NoiseSensitivity - [sum(all no error entries)]
For majority function, Stability = (2/pi)*delta + O((delta)^1.5) and NoiseSensitivity = O(1/sqrt(n*delta))
=> Hardness of Maj+Voter composition = 1 + (2/pi)*delta + O((delta)^1.5) + O(1/sqrt(n*delta)) - [sum(all no error entries)]
For large n, previous hardness tends to 1 + (2/pi)*delta + O((delta)^1.5) - [sum(all no error entries)]

From definition of BP*, Probability of No error entries >= 2/3 => Sum(all no error entries) >= 2/3
=> Hardness of Maj+Voter composition <= [pi + 2*delta + k*pi*(delta)^1.5]/2*pi - 2/3
=> Hardness Amplification for Majority+Voter Composition <= [3*pi + 6*delta + 6k*pi*(delta)^1.5 - 2] /[6*pi*delta]
=> Hardness Amplification for Majority+Voter Composition <= 1/delta*(1/2-1/3*pi) + 1/pi + 6k*pi*(delta)^0.5
which is huge amplification for small hardness of voter boolean functions.
=> weak voters are hardened by majority
=> Computing majority by a circuit is increasingly hard as voters become weak (intuitively obvious because if voter circuits err, majority circuit errs, an indirect proof of Margulis-Russo Threshold and Condorcet Jury Theorem).

Sensitivity Conjecture polynomially relates sensitivity and block sensitivity of boolean functions as: s(f) <= bs(f) <= poly(s(f)). So far only Noise Sensitivity has been applied as sensitivity measure throughout for quantifying BP* error in Majority voting because it is a probability measure. But Sensitivity and Block Sensitivity are maximum number of bits and disjoint blocks of bits sensitive to output. Noise sensitivity probability can be written as:

$$\frac{\text{for\_all\_length\_e(Number of correlations x/e for which [f(x) != f(x/e)])}}{\text{for\_all\_length\_e(Number of correlations x/e for which [f(x) == f(x/e)] + Number of correlations x/e for which [f(x) != f(x/e)])}}$$

Sensitivity and Block Sensitivity are maximum values of e(which can be number of bits or number of blocks) in summations of numerator and denominator. Block Sensitivity in Majority+Voter composition implies voter boolean functions are interdependent (correlated majority) and swayed by peer opinions en masse (Herding). Hardness of voter boolean function and majority are indirectly related to sensitivity and block sensitivity through summations in Noise Sensitivity probability.

NOTE: Here hardness of voter boolean function is approximately assumed to equal NoiseSensitivity of Voter Boolean Function and delta=epsilon. To be precise, delta = epsilon +/- error by BP* versus NoiseSensitivity/Stability Scenarios Matrix conventions. This +/- term is ignored. Accounting for this +/- error:

=> Hardness Amplification for Majority+Voter Composition <= 1/(delta + or - error)*(1/2-1/3*pi) + 1/pi + 6k*pi*(delta + or - error)^0.5

---
518. (THEORY and FEATURE-DONE) Hardness of Majority Voting, SAT Solver Update - Compressed Sensing and Moore-Penrose Pseudoinverse
- Conflicts - 6 March 2018 - related to 517

Approximate CNF SAT Solver has been updated to invoke Moore-Penrose Pseudoinverse function to compute approximate inverse of random 3SAT matrix and is multiplied as A^-1*b to find assignments x. Compressed Sensing implementation already uses pseudoinverse to decompress image from sketch which is similar to this SAT solver. Performance is almost similar to LSMR and MAXSAT approximation ratio hovers around 97-98% for 306 random 3SAT instances of 21 variables and 21 clauses. Randomized rounding threshold has been increased to 0.5 from 0.1 (a literal is set to 0 if it is < 0.5 and to 1 if >= 0.5). Observed per literal probabilities are equal to random matrix expected probability 1/sqrt(m*n). This reduces SAT solving to Compressed Sensing. There are some conflicts between the findings of SAT Solver and Hardness of Majority:

(#) Hardness of Majority+Voter Boolean Function Composition in both directions - bottom-up Voter-Majority and top-down MajorityInverse-VoterInverse have been described earlier (bottom-up hardness has been derived) and top-down inversion is considerably harder (#P-complete counting problem) and a likely choice for an one-way function and strong pseudorandom generator implying P != NP (or does it imply something more than P != NP because top-down inversion is in #P?).In oracle notation, top-down inversion is #P^#P assuming VoterInverse() assignments are #P oracles to MajorityInverse() #P problem. Toda's theorem implies PH is in P^#P. If P^#P is in #P^#P, then top-down inversion is harder than PH and hardness of top-down inversion implies more than just P != NP.

(#) SAT Solver random matrix analysis per-literal probability agrees with observed findings so far in small number of variables and clauses and MAXSAT approximation ratio for (21,21) and less number of variables-clauses combinations is almost 97% in most of the Solver executions - SAT solver solves the equal number of clause-variable instances exceeding 7/8-approximation similar to Karloff-Zwick semidefinite programming randomized algorithm but in deterministic polynomial time. Solving unequal clause-variables requires choice in a non-uniform distribution. Solving large number of variables-clauses in the order of millions might further confirm this trend but is cpu-intensive. So it still remains open if this more than 7/8-approximation implies P=NP. It is unusual that a simple rounding of real solutions to 0 or 1 from linear system of (boolean) equations, is able to solve most number of clauses per random SAT which is more obvious compared to other Randomized Rounding/Relaxation techniques like Semi-definite programming.

(#) If Hardness of Majority implies something stronger than P != NP e.g an one-way function or PRG in #P^#P then there is no conflict from SAT Solver >7/8-approximation in deterministic Polynomial time and P=NP.

---
519. (THEORY and FEATURE-DONE) SAT Solver Update - pinv2() - 1000 variables and 1000 clauses - 6 March 2018

SAT Solver has been updated to use pinv2() pseudoinverse function which has been documented to be faster. After removing some code causing redundant bottleneck, 1000 variables-1000 clauses random SATs have been solved by pinv2() and matrix multiplication to find real assignments which have been rounded. This means solving a million entry matrix (1000*1000). MAXSAT

approximation ratio is 97%-98% again. Probability per literal average observed coincides with theoretical value of 1/sqrt(m*n). It has to be noted that Random matrix expectation for solving SAT translated to linear system of equations is an average case solution and is not worst case. This >7/8 approximation could imply AverageP=AverageNP or PromiseP=PromiseNP (for equal number of variables-clauses) if not P=NP. This is the likely reason for observed per literal probability being 97% asymptotically less than theoretical per literal probability 100%.

---
520. (THEORY and FEATURE-DONE) SAT Solver Update - 2500 clauses and 2500 variables - LSMR,LSQR and PseudoInverse Benchmarks - related to 518 - 9 March 2018
---
1. SAT Solver class has been changed to accept the algorithm as parameter which can be function names for LSMR,LSQR,PseudoInverse,SPSOLVE, SOLVE etc.,
2. 2500 variables and 2500 clauses combination of SAT instances have been benchmarked by invoking LSMR, LSQR and PseudOInverse(pinv2) separately.
3. LSMR benchmark has 405 random 3SAT instances and is the fastest, LSQR and PseudoInverse(pinv2) benchmarks have only few random 3SAT instances.
4. LSQR and PseudoInverse(pinv2) are equally slower compared to LSMR by many orders of magnitude.
5. But approximation ratio for LSMR is 91-92% while LSQR and PseudoInverse trend at 96-97% though for small number of iterations. This probably implies an accuracy versus speed tradeoff: LSMR is less accurate but fast while LSQR and PseudoInverse(pinv2) are more accurate but less fast.
6. Observed probability per literal for all three coincide with random matrix 1/sqrt(m*n) uniform distribution probability per literal
7. These numbers are only representative figures and number of iterations for LSQR and PseudoInverse(pinv2) are too meagre.
8. Logs have been committed to:
  python-src/testlogs/CNFSATSolver.2500clauses2500variables.LSMR.log.9March2018
  python-src/testlogs/CNFSATSolver.2500clauses2500variables.LSQR.log.9March2018

python-src/testlogs/CNFSATSolver.2500clauses2500variables.PseudoInverse.log.9March2018

---
521. (THEORY and FEATURE-DONE) SAT Solver Update - Non-Uniform Choice - For both equal and unequal number of clauses and variables - related to 520 - 11 March 2018

SAT Solver has been updated to include a new Non-Uniform Choice function for creating random SAT instances by choosing a literal by probability = 1/sqrt(m*n) which works for both equal and unequal number of clauses. This new function chooses non-uniformly. This function creates a submatrix of dimension (sqrt(m),sqrt(n)) chosen uniformly from larger matrix of (m clauses * n variables) by invoking choice() function to choose sqrt(m) rows from m rows and sqrt(n) columns from n columns. This uniformly chooses sqrt(m)*sqrt(n) entries from m*n entries of random matrix. This simulates the random matrix probability 1/sqrt(m*n) = sqrt(m*n)/m*n. From this submatrix 3 literals are chosen in succession to create a random 3SAT clause. Replace flag in choice() was set to False and True while creating random clause and there was not much difference and has been set to True.Logs for following 3 variables-clauses combinations have been committed to testlogs/ for ~100 random 3SAT instances each:
      1000 variables, 1000 clauses - CNFSATSolver.1000variables1000clauses.NonUniformChoice.log.11March2018
      1100 variables, 1200 clauses -

CNFSATSolver.1100variables1200clauses.NonUniformChoice.log.11March2018
      1200 variables, 1100 clauses -
CNFSATSolver.1200variables1100clauses.NonUniformChoice.log.11March2018
From logs it is evident non-uniform choice function defined as previously approximately simulates the bias and almost matches theoretical 1/sqrt(m*n) random matrix expected per literal probability. LSMR algorithm has been chosen and MAXSAT approximation ratio is 91-93% for 100 iterations for each of the 3 variable-clause combinations. For equal variable-clause combinations, non-uniform choice function is equivalent to choice() and expected and observed probabilities per literal agree fully. Previous benchmarks have been done to verify the random matrix theory for Satisfiability problem which is a probabilistic average case P?=NP problem mentioned in hardness amplification conflicts previously. There is also a special case conflict with PARITYSAT Hastad-Linial-Mansour-Nisan Theorem circuit size counterexample mentioned in 53.15. Also unequal variable-clause SATs can be simulated by equal variable-clause SATs by setting redundant variables to 0 in disjunction of the literals in clauses or redundant clauses in conjunction to 1. Thus non-uniform choice may not be necessary.

---

522. (THEORY and FEATURE-DONE) SAT Solver Update - Non-Uniform Choice - 3200 variables and 3100 clauses and Alpha=4.26 - related to 521
- 12 March 2018

SAT Solver has been updated for some aesthetics - literal selection in non-uniform choice has been parametrized: sequential or simultaneous
 based on how three random literals are chosen per random 3SAT clause - sequentially as 3*nP1 or simultaneously as nP3. But no difference was observed between these two choices. Number of variables is set to 3200 and clauses to 3100. LSMR iteration has been increased to 5 and conlim reduced to 10. For few random 3SATs, observed MAXSAT approximation ratio average stands at ~95%. This accuracy has been observed to increase proportional to number of LSMR iterations. Commercial SAT solvers solve million variable-clause combinations in few seconds but in exponential time. Two other logs have been committed which solve the notorious Alpha=4.26 Clause/Variable ratio, which is known to be hardest subset of SAT and
where a phase transition occurs from easy-to-hard. Number of unsatisfiable formulae increase for alpha > 4.26. MAXSAT approximation ratio observed for few random 3SAT instances is ~91% for number of variables 300 and 1000 and alpha=4.26. Observed per literal probability is skewed and substituting it in random matrix MAXSAT approximation ratio m^2*n^2*p^4 shows an anomaly. Another fact observed: per literal probability does not change at all for any number of iterations and stays put and because of this observed MAXSAT ratio converges within few iterations itself.

References:

522.1 SAT Solvers and Phase Transition at Alpha=4.26 [Clause/Variable Ratio] - [Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman] - https://www.cs.cornell.edu/gomes/pdf/2008_gomes_knowledge_satisfiability.pdf

---

523. (THEORY and FEATURE-DONE) SAT Solver Update - Non-uniform Choice 2 - 5000 variables and Alpha=4.267 and some intuition - related to 522
- 15 March 2018

SAT Solver has been updated for a new nonuniform choice function nonuniform_choice2() which has following algorithm:
      - Each permutation nPn of length n is elongated to n*n*alpha length array by replicating a non-variable X of large index (n*n)

- Literals are chosen from this new array as (n*n*alpha)P1 or (n*n*alpha)P3 in a while loop till a valid literal which is not equal to n*n is found
- This creates a probability of fraction n/(n*n*alpha)=1/(n*alpha) for per literal choice.

But this implementation was found to be no different from nonuniform_choice() which chooses a submatrix in per literal probability and therefore has been commented at present. MAXSAT approximation ratio observed was 88% for nonuniform_choice2() versus 90% for nonuniform_choice(). 5000 variables and Alpha=4.267 have been solved for few iterations and MAXSAT approximation ration is 90-91%. In all the benchmarks done so far, it has been observed that equal variables-clause combinations almost always converge to ~95% MAXSAT approximation ratio and Alpha=4.267 almost always converge to ~90-91%. Though these numbers are not conclusive and comprehensive, previous emerging pattern is too striking.

Here the intuition on how LSMR/LSQR works has to be mooted - Each binary assignment string can be thought of as a step function plotted as binary value versus variables. For example, assignment 001101 is plotted as:

where troughs represent 0s and peaks 1s. LSMR/LSQR finds a set of real valued points on a sinusoidal polynomial which approximates this step function by minimizing sum of distances/errors between troughs-peaks of polynomial and those of step function. By rounding the peaks and troughs of this LSMR polynomial to 1s and 0s thus converts the sinusoid to a step function. Minimizing the sum of squares distance error implies, this polynomial is almost unique(though there can be multiple satisfying assignments) and is able to extract atleast one satisfying assignment.

Probability of an LSMR/LSQR real-to-binary round off assignment failing to satisfy a random 3SAT:

For each binary variable xi, LSMR/LSQR creates a real value which is rounded off as:

```
xi   -    0  if  xi < 0.5
xi   -    1  if  xi > 0.5
```

This round off can fail if:

```
xi   -    1  if  xi < 0.5
xi   -    0  if  xi > 0.5
```

Lovasz Local Lemma Analyses described earlier further explain the clause-variable dependency graph(Factor graph) scenarios but only lowerbound the MAXSAT approximation ratio for various values of dependency. But Random Matrix Least Squares Error Partial Derivative Analysis forbids values of assignments which cause such large deviations of real solutions from 0s and 1s to maximum possible extent,by providing a mechanism on how to choose a literal - 1/sqrt(m*n) probability - reverse-engineered a posteriori probability. In other words, real solutions are almost in proximity to 0 or 1 - sinusoid approximates step-function near-perfectly. Any other probability distribution of per literal choice is theoretically inferior.

---

524. (THEORY and FEATURE-DONE) SAT Solver Update - Variables and Alpha as parameters and two Alpha Versus MAXSAT ratio graph plots - related to 523 - 16 March 2018

SAT Solver has been updated to accept commandline parameters of number of variables and Alpha (Clause/Variable ratio) as:

```
$python CNFSATSolver.py <number_of_variables> <alpha>
```

Some debug prints have been changed. Following sets of SATs were solved and variation of MAXSAT approximation ratio versus Alpha has been captured as graph (for nonuniform_choice()):

Number of Variables - 600 (after atleast 15 random 3SAT iterations)

Alpha	Observed MAXSAT Approximation Ratio
1	94.5%
2	92.7%
3	91.5%
4	90.74%
4.267	90.45%
5	90.28%
6	90.25%

Number of Variables - 300 (after atleast 15 random 3SAT iterations)

Alpha	Observed MAXSAT Approximation Ratio
1	95%
2	93%
3	92%
4	90.6%
4.267	90.6%
5	90.49%
6	90.73%

Logs for an iteration of 5000 variables and Alpha=4.267 has been committed to python-src/testlogs/CNFSATSolver.5000variablesAlpha4.267.log.16March2017 which also has MAXSAT approximation ratio of 90-91%. Gradual nosedive of MAXSAT approximation ratio as Alpha increases is notable and it stabilizes to 90-91% after Alpha >= 4 for both 300 and 600 variables implying phase transition.

SAT Solver has been updated for a new nonuniform_choice3() function based on an Epsilon Bias Test Snippet EpsilonBiasNonUniformChoice.py which has been added to repository. This is similar to nonuniform_choice2() but Probabilities of nonuniformly chosen literals are computed separately and updated within the while loops also. Algorithm is similar to nonuniform_choice2() which elongates a permutation by replicating a skew variable. Average probabilities of all variables other than skew variable almost match the random matrix 1/sqrt(m*n) probability. Logs which print these probabilities for 1000 variables and Alpha=1/Alpha=4.267 have been committed to testlogs/CNFSATSolver.1000variablesAlpha1.log.16March2018 and testlogs/CNFSATSolver.1000variablesAlpha4.267.log.16March2018.

---

525. (FEATURE-DONE) SAT Solver Update - nonuniform_choice3() updated for constant multiple in expansion of the variables array
- 19 March 2018 - related to 524

---

1. CNFSATSolver.py nonuniform_choice3() has been changed to expand the variables array by following relation:
        n + x = n*sqrt(alpha)
        x = n*(sqrt(alpha)-1)
        => array of n variables is expanded to array of size n*(sqrt(alpha)-damp) for alpha = number_of_clauses/number_of_variables
2. damp variable has been hardcoded to 2 than 1 which approximates theoretical 1/sqrt(m*n) probability well because of round-off.
3. This expansion creates an approximate nonuniform probability 1/n*sqrt(alpha) per literal excluding replicated skew variable which is

filtered in the while loops
4. Example SAT Solver logs for 1000 variables and Alpha=4.267 has been committed to python-src/testlogs/CNFSATSolver.1000variablesAlpha4.267.log.19March2018
5. MAXSAT Approximation ratio again is 90-91% for 20+ random 3SATs and nonuniform per literal probability almost matches 1/sqrt(m*n)
(till penultimate decimal)

---

526. (THEORY) Counterexample in Majority Voting, Timeline Evolution of Belief Propagation, Intrinsic Merit - related to 507 - 20 March 2018

Previous sections described a counterexample in Majority Voting for a question:
"Did an artefact exist millenia ago?"
There are two possible answers: Yes and No which depend on intrinsically determining truth of answer to this question. Both possible answers are belief propagated as trees which evolve over time following a gossip protocol. Root of either of the trees propagates its belief to its children and so on recursively for all other internal nodes. Each edge in either of the trees has a belief potential for "Yes" and "No". Parent node propagates its belief to the children proportional to this potential. If potential is below a threshold, belief propagation stops and trees are pruned. Thus there are two trees which correspond to two sets of people who believe "Yes" and "No" by potentials percolated from Roots. Only the root node has most authentic direct intrinsic evidence to either Yes or No truth value to the question. Question is: How can intrinsic absolute truth of answer be determined? Does size of the trees imply truth i.e Can Majority Voting determine intrinsic truth value at the root? This problem becomes all the more non-trivial when a chronologically ancient portion of tree (e.g Root is extinct) is not available and only nodes closer to leaves remain. Value of belief potential is a function of intrinsic truth viewed by the roots of either tree.

---

527. (THEORY and FEATURE-DONE) Hardy-Ramanujan Approximate Ray Shooting Queries for Factors - Optimization in Computational Geometric
Factorization - 21 March 2018 - related to 486

---

1. Ray Shooting Queries based on Hardy-Ramanujan Theorem for Normal Order of number of factors of an integer has been implemented as a
new function.
2. Each approximate factor is queried by a ray from origin of slope tan(m*pi/(2*k*loglogN)) intersecting hyperbolic arc bow and approximate factors are:

sqrt(N/[tan(m*pi/(2*k*loglogN))])-1 for m=1,2,3,...,kloglogN

3. Two integers have been factorized using local tile search and comparison between approximate factors and actual factors have been logged in:

python-src/testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.HardyRamanujanRayShootingQueries.log.21March2018

python-src/testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.HardyRamanujanRayShootingQueries2.log.21March2018
4. Approximate Ray Shooting could be useful for integers having large value of factor multiplicity BigOmega (sum of prime factor powers)
5. Constant k has been hardcoded presently and has to be heuristically found.
6. Approximate Factors are helpful for sieving huge integers and binary search can be localized based on these approximate factors as beacons.It is not necessary to search the entire pixelated hyperbolic arc.

---

528. (FEATURE-DONE) Unsorted Search Update - Streaming Abstract Generator File Datasource Support, Prefix-Suffix substrings hashtables
- 23 March 2018

---

1. Hardcoded File name in Streaming Abstract Generator has been replaced by a file name variable and file contents are tokenized and
stripped of leading and trailing whitespaces and yielded.
2. Padding of '#' in Unsorted Search has been replaced by '0'. Revised Unsorted Search Algorithm in
https://github.com/shrinivaasanka/Grafit/blob/master/course_material/ComputerScienceMiscellaneous/ComputerScienceMiscellaneous_CourseNotes.txt has been implemented by creating hash tables for all integer-string prefixes and suffixes (though not for all substrings) than just for single digits.
3.New function to print contents of all hashtables has been added.
4.Function create_prefix_suffix_hashtables() initializes prefix and suffix hashtables for strings from Streaming Abstract Generator
5.search_number() function has been rewritten to search for all prefixes and suffixes hashtables and match True/False is written
6.New input file First100Primes.txt has been created.
7.Primes and Non-Primes are lookedup and logs has been committed to testlogs/UnsortedSearch.log.23March2018

---

529. (THEORY) Consensus, Pareto Optimality, Intrinsic-Perceived Value/Merit Equilibrium, Fair Division, Nultiple Agent Resource Allocation(MARA),MAXSAT ranking of texts - related to 432,440,513,526 and all Intrinsic Fitness/Merit sections - 26 March 2018,27 March 2018

Intrinsic Fitness/Merit and Perceived Merit Equilibrium has been described earlier as an approximate consensus measure for agreeing on value or
merit of an entity. Consensus problem for absolute intrinsic merit arises in the following context of Fair Division where set of resources have
to be allocated amongst multiple agents fairly (also known as MARA-Multi Agent Resource Allocation). Fairness in allocation is usually expressed by Pareto Optimality which is defined as:
For a utility function u:(Good,Agent)->Happiness, set of agents (x1,x2,x3,...,xn) and goods/services/resources (r1,r2,r3,...,rm) allocated by u with a Happiness Vector V=(a1,a2,a3,...,an) , there is no other alternative allocation Happiness Vector V'=(a1',a2',a3',...,an') for which u(ai') >= u(ai) for all i=1,2,3,...,n and there exists an agent i such that u(ai') > u(ai).
Most obvious application of Pareto Optimality is Pricing or Determining Value of Goods. Allocation is Envy-Free if no agent feels its own resource allocation as inferior to others. Equating Intrinsic Value and Intrinsic Merit reduces the Multi Agent Resource Allocation or Fair Division problem to Fair Judgement of Intrinsic Merit. Preferences or Desires of an agent are specified by Logic clauses - Ceteris Paribus - all others being equal agent prefers (P1 /\ !P2) to (!P1 /\ P2). Translating the Intrinsic Value MARA problems to Intrinsic Merit MARA problem is non-trivial. For example, if the agents are web URLs and merit has to be fairly apportioned to the Web URLs (Ranking), to ensure no URL feels envious requires defining per URL preferences. This is precisely where MAXSAT representation of per text/URL preferences finds its utility - number of clauses satisfied per text/URL determines its merit i.e text/URL prefers some clauses over others or it has certain special qualities not in others. Pareto Optimality is NP-Complete by reduction from Set Packing problem(Problem of finding disjoint subsets from a collection of subsets).

Traditional Ranking methodology followed is to compute ranking scores per website URL independent of other URLs i.e envy is not accounted for which vitiates fairness. By MARA based ranking, Ranking Fairness is strengthened and Total Cumulative Intrinsic Merit = 1.0, is viewed as Pie/Cake cutting problem and websites are allocated pareto-optimal fractional merit by an envy-free

protocol. Thus Sum(rank(url))=1.

An example envy-free cake cutting algorithm is a hash function which distributes the values nearly equally onto the per-key buckets of the hash table - all buckets have equal number of values or resources are equidistributed. Varying the hash functions changes the per-key bucket number of values. Each key is an agent requiring certain number of values in its bucket and Choosing a hash function which satisfies this constraint is a MARA protocol.

There exists a bijection between a discrete probability distribution and per-bucket sizes of hashtables (or) parts in a set partition. If each discrete random variable xi has probability p(xi), sum(p(xi)) = 1. Similarly, if each bucket xi of a hash table (part in a set partition/exact cover) has size b(xi), sum(b(xi)) = N (total number of values). Choosing an apt probability distribution does fair allocation e.g If fair allocation constraint is a gaussian, mapping discrete points in normal probability distribution (p(xi)) to size of hash table buckets fairly allocates. Expressing the fair allocation constraints as 3SAT and solving it by Random Matrix LSMR/LSQR Approximate SAT Solver yields real solutions to the constraint variables if rounding is overlooked. As described earlier, this random matrix SAT solver approximates the assignment step function by a polynomial traversing these real assignment points. Riemann Sum Discrete Integral of points on this polynomial should be theoretically equal or close to parity of the binary assignment extracted by LSMR/LSQR.

References:

529.1 MARA Survey - https://staff.science.uva.nl/u.endriss/MARA/mara-survey.pdf

---
530. (THEORY and FEATURE-DONE) Recursive Lambda Function Growth - Graph Tensor Neuron Network Intrinsic Merit for Random Walks
- Analysis for different text - 26 March 2018
---
1. Text input for Recursive Lambda Function Growth has been updated
2. Graph Tensor Neuron Network Intrinsic Merit for Lambda Function Composition of Random Walks of Definition Graph have been captured in
log
testlogs/RecursiveLambdaFunctionGrowth.log.GraphTensorNeuronNetwork.26March2018
3. Maximum Graph Tensor Neuron Network Intrinsic Merit occurs for Random Walk Composition:
grow_lambda_function3(): Maximum Per Random Walk Graph Tensor Neuron Network Intrinsic Merit :
(u'(housing(protective,(certain(zone,(something(target,(part(include,(urban(area,(Chennai(city,formerly))))))))))))', 6.023684210526316)
4.Top percentile Unsupervised Classes of this text by Dense Subgraph Discovery (Core Numbers) are:
==========
Unsupervised Classification based on top percentile Core numbers of the definition graph(subgraph of WordNet)
==========
This document belongs to class: arsenic ,core number= 16
This document belongs to class: Greenwich_Village ,core number= 14
This document belongs to class: state_of_matter ,core number= 14
This document belongs to class: order ,core number= 14
This document belongs to class: include ,core number= 10
This document belongs to class: part ,core number= 10
This document belongs to class: exploitation ,core number= 9
This document belongs to class: area ,core number= 9
This document belongs to class: three ,core number= 9

This document belongs to class: collector ,core number= 8
This document belongs to class: housing ,core number= 8
This document belongs to class: January ,core number= 8
This document belongs to class: trey ,core number= 8
This document belongs to class: Chennai ,core number= 7
This document belongs to class: urban ,core number= 7
This document belongs to class: planning ,core number= 6
This document belongs to class: zone ,core number= 6
This document belongs to class: free-base ,core number= 6
This document belongs to class: travel ,core number= 6
This document belongs to class: one ,core number= 6
This document belongs to class: target ,core number= 6
This document belongs to class: something ,core number= 6
This document belongs to class: republic ,core number= 6
This document belongs to class: government ,core number= 5
This document belongs to class: legal_power ,core number= 5
---
which reasonably coincide with the vertices of the maximum intrinsic merit random walk and capture the Legal/Gubernatorial/Urban planning nature of the text though there are WordNet anomalies printing 'arsenic' etc., as classes. These anomalies are caused because usually Urban planning is related to exploitation and Solid Waste Management which in turn are connected to Toxic substances and Pollutants. WordNet contains all these relations. Some other Net like ConceptNet5 might probably do better which are word2vec based.

Essence:

Meaning of a text is approximated as:
	- Create a Definition Graph Representation of Text from some Ontology
	- Do random walks/Hamiltonian on the graph to simulate the human text comprehension
	- Get classes of the text from Recursive Gloss Overlap Unsupervised Classifier
	- Compute Lambda Composition Trees for all random walks in definition graph
	- These random walk lambda composition trees are approximations of all possible meanings of the text
	- Compute Graph Tensor Neuron Network Intrinsic Merit for all random walk lambda composition trees
	- Lambda Composition Tree of Maximum Graph Tensor Neuron Network Intrinsic Merit is the most likely approximate meaning of the text
	- This is obvious because this maximum merit tree has Neuron Tensor Network Relations of maximum similarity/truth value and these similarities are composed and belief propagated

---
---
531. (FEATURE-DONE) Recursive Lambda Function Growth - Simple Cycles and Rich Club Coefficient - 27 March 2018

1.Recursive Lambda Function Growth implementation has been updated to loop through cycles in the definition graph of text and to choose between Cycles and Random Walks by a boolean flag ClosedPaths.
2.Closed Paths or Cycles simulate the meaning better and the tree lambda composition obtained from cycle vertices has been experimentally found to approximate meaning more closely.
3.Logs for this has been committed to testlogs/RecursiveLambdaFunctionGrowth.log.GraphTensorNeuronNetwork.27March2018
4.Rich Club Coefficients of the Definition Graph has been printed for each degree as a measure of connectivity of high degree vertices(rich club):
Rich Club Coefficient of the Recursive Gloss Overlap Definition Graph: {0: 0.004865591072487624, 1: 0.016526610644257703, 2: 0.018738738738738738, 3: 0.02396259497369959, 4: 0.024154589371980676, 5: 0.023809523809523808, 6:

0.028985507246376812, 7: 0.032679738562091505, 8: 0.022222222222222223, 9: 0.0, 10: 0.0}
5.Top core number classes from Unsupervised Recursive Gloss Overlap Classifier:
==
============================
Unsupervised Classification based on top percentile Core numbers of the definition graph(subgraph of WordNet)
==
============================
This document belongs to class: incorporate ,core number= 4
This document belongs to class: environment ,core number= 4
This document belongs to class: regional ,core number= 4
This document belongs to class: `in ,core number= 4
This document belongs to class: component ,core number= 4
This document belongs to class: exploitation ,core number= 4
This document belongs to class: useful ,core number= 4
This document belongs to class: relating ,core number= 4
This document belongs to class: unit ,core number= 4
This document belongs to class: particular ,core number= 4
This document belongs to class: making ,core number= 4
This document belongs to class: process ,core number= 4
This document belongs to class: sphere ,core number= 4
This document belongs to class: something ,core number= 4
This document belongs to class: goal ,core number= 4
This document belongs to class: farm ,core number= 4
This document belongs to class: urbanization ,core number= 4
This document belongs to class: strategic ,core number= 4
This document belongs to class: ' ,core number= 4
This document belongs to class: increase ,core number= 4
This document belongs to class: comforts ,core number= 4
This document belongs to class: city ,core number= 4
This document belongs to class: district ,core number= 4
This document belongs to class: farming ,core number= 4
This document belongs to class: urban ,core number= 4
This document belongs to class: way ,core number= 4
This document belongs to class: plan ,core number= 4
This document belongs to class: do ,core number= 4
This document belongs to class: contain ,core number= 4
This document belongs to class: see ,core number= 4
This document belongs to class: state ,core number= 4
This document belongs to class: region ,core number= 4
This document belongs to class: proposal ,core number= 4
This document belongs to class: life ,core number= 4
This document belongs to class: decisiveness ,core number= 4
This document belongs to class: lives ,core number= 4
This document belongs to class: metropolitan ,core number= 4

----
coincide reasonably well to the Maximum Merit Cycle as below and the meaning of the text can be inferred from the composition tree of the word chain and the composed tensor potential is printed for this maximum merit cycle:

Cycle : [u'particular', u'regional', u'region', u'something', u'see', u'make', u'plan', u'goal', u'state', u'city', u'urban', u'area', u'exploitation', u'land', u'farm', u'unit', u'assembly', u'parts', u'environment', u'sphere']
Cycle Composition Tree for this cycle : (u'particular((region(regional, (see(something,(plan(make,(state(goal,(urban(city,(exploitation(area,(farm(land, (assembly(unit,(environment(parts,sphere))))))))))))))))))', 7.250082923612336)
maximum_per_random_walk_graph_tensor_neuron_network_intrinsic_merit=
(u'(regional(particular,(`in(region,(pronounce(way,(see(certain,(add(make, (important(increase,(plan(strategic,(state(goal,(urban(city, (administrative(relating,(unit(agency,(land(farm,(useful(exploitation, (proper(necessitate,(metropolitan(person,

(environment(lives,sphere))))))))))))))))))))))))))))))))', 14.322488296017706)
==========
grow_lambda_function3(): Graph Tensor Neuron Network Intrinsic Merit for this text: 3804.34147246
6.NetworkX library has been upgraded to recently released version 2.1

---

532. (THEORY-FEATURE-DONE) Least Squares SAT Solver Benchmarks - 1000 variables - Non-uniform choice 3 - different values of Alpha - 29 March 2018 - related to 524

---

1. SAT Solver (LSMR) has been benchmarked for 1000 variables and for varying values of Alpha as below. MAXSAT approximation ratio after
atleast 20 iterations of random 3SAT for nonuniform_choice3() are:
Alpha = 1 - Observed - MAXSAT-APPROXIMATION Ratio - Moving Average Percentage of Clauses per CNF satisfied so far: 94.3032258065
Alpha = 2 - Observed - MAXSAT-APPROXIMATION Ratio - Moving Average Percentage of Clauses per CNF satisfied so far: 92.2115384615
Alpha = 3 - Observed - MAXSAT-APPROXIMATION Ratio - Moving Average Percentage of Clauses per CNF satisfied so far: 91.1458333333
Alpha = 4.267 - Observed - MAXSAT-APPROXIMATION Ratio - Moving Average Percentage of Clauses per CNF satisfied so far: 90.196679346
Alpha = 5 - Observed - MAXSAT-APPROXIMATION Ratio - Moving Average Percentage of Clauses per CNF satisfied so far: 89.7233333333
Alpha = 6 - Observed - MAXSAT-APPROXIMATION Ratio - Moving Average Percentage of Clauses per CNF satisfied so far: 89.8702898551
Alpha = 7 - Observed - MAXSAT-APPROXIMATION Ratio - Moving Average Percentage of Clauses per CNF satisfied so far: 89.8266233766

2.As usual for equal variable-clauses MAXSAT ratio is the maximum at 94-95% and decreases for increasing alpha. For Alpha=4.267, MAXSAT ratio is 90-91% similar to previous benchmarks(which were done on nonuniform_choice()).

3.Log excerpts have been captured in testlogs/CNFSATSolver.1000variablesDifferentAlphasBenchmarks.29March2018

---

533. (THEORY) Packing/Filling/Tiling problems, Complement Functions/Diophantine Equations, Ramsey Coloring,
Exact Cover, Matiyasevich-Robinson-Davis-Putnam Theorem - related to 461,462,490 - 19 April 2018

As mentioned in previous sections, complement functions are subsets of set of Diophantine Equations defining
each diophantine set in an exact cover - this generalizes definition of complement functions to exact cover of size greater than 2. Exact Cover is a special scenario of Bin Packing/Space Filling/Tiling plane (e.g Pentominoes tiling of a Chess board, N-Queens Problem etc.,). Complement Diophantine Equations for sets in an exact cover uniquely determine each element in tiling and thus create a bijective map between Diophantines and Tiles(i.e sets in exact cover).

Reference:

533.1 Pentominoes Tiling Exact Cover for Chess board - https://en.wikipedia.org/wiki/Exact_cover - Each pentomino tile can be defined by a Diophantine Equation because Exact Cover is NP-complete recursive set, from MRDP theorem. These diophantine equations for tiles are complement functions of the exact cover.
533.2 Euclidean Ramsey Theory - http://www.math.ucsd.edu/~ronspubs/pre_Euclidean.pdf - [RL Graham] - Euclidean

Ramsey Theory pertains to the coloring/partitioning of euclidean plane E into tiles Ci i.e E = U Ci and there exists set of subsets E' of E such that for every x in E', x is exactly contained in one Ci.

---

534. (THEORY and FEATURE-DONE) Complement Diophantine and Factorization - Pell Equation Solver Update -
20 April 2018

---

1. Complement Function Implementation has been updated to invoke sympy Pell Equation solver for some N and D
in x^2 - D*y^2 = N
2. Factorization reduces to integer solutions of Pell Equation but the converse is harder - factors must be known
a priori as pq = N from some factoring algorithm. Then following equations can be solved:

```
        x + sqrt(D)*y = p
        x - sqrt(D)*y = q
```

for x and y.

---

535. (THEORY and FEATURE-DONE) Complement Diophantine and Factorization - Pell Equation Solver Update 2 -
20 April 2018

---

1. Complement Function Implementation has been updated to invoke sympy Pell Equation solver for some N and D
in x^2 - D*y^2 = N for D=1
2. Following are the factors p,q of N:

```
  x + y = p
  x - y = q
```

for Pell equation solutions x and y. This is exponential in number of bits of N and equivalent to most
Number Field Sieve algorithms for Factorization in vogue. Computational Geometric Factorization in NC
implies x and y can be found in parallel polynomial time (which is quite surprising given the antiquity and
notorious difficulty of this ~1500 year old problem).
3. Pell's Equation for Factoring and Prime polynomials (e.g Matiyasevich, Jones-Sato-Wada-Wiens) are thus
complement diophantines representing the exact cover {Primes,Composites} of the Set of Natural Numbers

---

536. (THEORY) Complement Functions, Complement Graphs, Ramsey coloring, Intrinsic Merit of Text-Graph -
related to 2,338 - 21 April 2018, 25 April 2018

Relations between the concept of complement functions and complement graphs have been described earlier in the
context of Perfect Graph Theorem. This notion of complement graphs has direct application in graph representation of texts (text-graph) and resulting Intrinsic Merit qualities of the text-graph. For example, complement graph dual of a text-graph has edges amongst word vertices absent in its primal. Also cliques in primal graph G become independent sets in complement dual graph G' and vice-versa. Complement graphs have spawned vast literature of theoretical results foremost being Perfect Graph Theorem. Recursive Gloss Overlap and Recursive Lambda Function Growth algorithms map text to Graph of word vertices

and composition of Lambda Functions - a language between Context Free Grammar and Context Sensitive Grammar. Composition of lambda functions by random walks/cycles/girth in text graphs makes it a Church-Turing-equivalent model which is an overlap of two fields - Formal Languages Theory and Graph Theory. It is an open question as to what Complements of text-graph and their Chromatic number imply: Does a complement of text-graph negate the meaning of text? As opposed to Coloring a linear sentence by Part-of-speech tagging, what does coloring of text-graph imply? E.g The sentence "It rained heavily today" is colored by PoS tags as Pronoun-Verb-Adjective-Object which is a Ramsey coloring of text construed as a sequence of words. Alphabet coloring and arithmetic progressions of alphabet positions in text has been already described in 2.10,2.11 and 2.12. In text-graphs coloring is usually edge coloring and not vertex coloring where each edge color corresponds to a relation (Is-a, Has-a, Is-part-of etc.,) between word vertices. All theorems related to Edge Coloring therefore apply to text graphs. Thus Complement Functions/Complement Graphs are useful in deep structure analysis of text graphs. If the text-graph is complete, Ramsey Theorem for edge k-coloring applies (relations are colors), order emerges and there exists a c-homogeneous subgraph edge-colored by same relation c.

There is a known theorem which states that if a graph G is disconnected its complement G's is connected and vice-versa. This can be proved by elementary arguments. If x-y is not an edge in G, x-y have an edge in G'. If x-y is an edge in G, x and y are in same component in G. If z is a vertex in some other disconnected component of G, then there are no x-z and y-z vertices in G but they exist in G' and there is x-z-y path in G' and G' is connected. In the context of WordNet relations, variant of this theorem has been proved as condition for meaningfulness and Intrinsic Merit in Section 5.16 of https://tac.nist.gov/publications/2010/participant.papers/CMI_IIT.proceedings.pdf . This qualitatively answers the previous question on meaningfulness of text-graph complement. If text-graph G is disconnected (i.e less meaningful) its complement G' is connected (i.e more meaningful). Here two meanings of primal and its complement text graph need not negate each other. Meaning of complement could be totally different from primal and may be in some other class.

WordNet Synsets for words have noun,verb etc., suffixes (.n,.v,...) which is tantamount to vertex coloring of the text-graph as nouns,verbs,adjectives etc.,. Set of vertices of same color class constitute independent sets or partition the word vertices into monochromatic independent sets. Perfect Graph Theorem applies if text-graph G and its complement G' are perfect (i.e if every induced subgraph of G or G' has its Chromatic Number equal to Maximal Clique Number).

References:

536.1 Introduction to Graph Theory - [Douglas B.West] - Perfect Graph Theorem for complement graphs, Ramsey Theorem for Graphs - Pages 226,380
536.2 A New Measure of Word Semantic Similarity based on WordNet Hierarchy and DAG Theory - https://pdfs.semanticscholar.org/d3a0/c2a70d3d39a5b41b4ef8c4890a481307259f.pdf

---
537. (FEATURE-DONE) ConceptNet Client Upgrade to 5.6 - REST endpoints for Emoticons - 23 April 2018
---
1. ConceptNet Client has been updated for new REST endpoints in ConceptNet 5.6
2. Function for querying emotions has been added for new emoji endpoints in ConceptNet 5.6
3. logs for this have been committed to testlogs/ConceptNet5Client.log.23April2018

---
538. (THEORY) Shell Turing Machines, Word2Vec and Intrinsic Merit - 24 April

2018, 25 April 2018 - related to 42

Shell Turing Machines described earlier are experimental enhancements to Turing Machine definition which
introduces dimension as a parameter in addition to tapes, alphabets, head etc., Adding dimension to Turing
Machines has immense applications and simulates lot of real world computations which span across multiple
dimensions. For example a Turing Machine defined in dimension d+1 has more computational power than a machine
defined in dimension d i.e L(T(d)) is in L(T(d+1)). This is a dimensional hierarchy as opposed to Time and Space
hierarchies of Turing Machines. This has striking applications in graph representation of texts. Each word
vertex in a definition graph G1 can be represented as a vector in a space of dimension d. Word2Vec embeddings
(e.g ConceptNet) already implement this by mapping each word to a vector in space of some dimension.
Then another definition graph G2 of words defined on a vector space of dimension d+1 can elicit more meaning
from text than G1. Recursive Lambda Composition Trees for G1 and G2 are two Turing-equivalent computation models
 approximating the meaning - G2 approximates better than G1. Essentially a Turing Machine/Lambda Function is
extracted from natural language text by growing Lambda Function by recursive composition. In other words intrinsic merit computed for G2 is more accurate than G1 for same text. Present algorithms for Recursive Gloss Overlap and Recursive Lambda Function Growth do not affix dimension information for word vertices.

---

539. (THEORY) Recursive Lambda Function Growth for sentences - related to 385,412,530,536 - 26 April 2018

Recursive Lambda Function Growth algorithm described so far depends on a text graph and walks/cycles in the text
graph are deemed as sentences of word vertices connected by WordNet/ConceptNet/Ontology relations which are converted to lambda function composition tree. This section explains how to construct a lambda function composition from linear sentences with no text-graph representation.

Approximate algorithm to create a lambda composition tree from sentence:

---

*) PoS/NER tag the sentence into Nouns,Verbs,Adjectives,Adverbs,Pronouns etc.,
*) Make all words tagged as verbs into lambda functions and noun/object for the verb as respective arguments
for the lambda function. This is a thumbrule because verbs in the sentences are the crux/actions on the nouns and objects.
*) Most natural languages are self-similar in the sense, the simple sentence template Noun-Pronoun-Adverb-Verb-Adjective-Object is recursively substituted to create large sentences. For example Object in Noun-Verb-Object template can itself be another template of the form Noun-Verb-Object.
*) Adjectives/Adverbs qualifying nouns/verbs/objects are created as wrapper lambda functions for nouns/verbs/objects.
*) This algorithm for extracting Lambda functions/Turing Machines from Natural Language Texts is the maximum theoretical limit attainable if Church-Turing Thesis is true. In other words, this algorithm creates Turing machines recognizing Recursively Enumerable sets.

Example sentence: "It rained heavily today"
The example sentence is PoS tagged as Pronoun-Verb-Adjective-Object. Verb

"rained" is marked as lambda function (after normalizing to "rain" if preferred.). Adjective "heavily" qualifying Object "today" is the wrapper function "heavily(today)". "It" is a separate lambda function for pronoun. Verb function "rained" thus accepts "It" and "heavily(today)" as arguments creating the tree:

    rained(It, heavily(today))

This algorithm is independent of natural language of the text. Lacuna in this composition from flat sentence is the deep structure - connections/paths between words - is not exploited which is remedied by the random walks/cycle traversal in text-graph and maximum merit composition is chosen.

----

540. (THEORY and FEATURE-DONE) Recursive Lambda Function Growth - Update - WordNet Relations Deep Structure
- 1 May 2018

----

1. Recursive Lambda Function Growth python implementation has been updated to define a new function to expand the
Random Walks and Simple Cycles in Recursive Gloss Overlap graph to include the deep structure relations between the pairs of words.
2. This invokes path_between() function implemented in WordNetPath.py which is python alternative implemented in NeuronRain to WordNet::Similarity in Perl CPAN
3. Logs for this have been committed to testlogs/RecursiveLambdaFunctionGrowth.log.GraphTensorNeuronNetworkDeepStructure.1May2018
4. This is the deepest possible graph structure which can be extracted from a Natural Language Text for an ontology.
5. Maximum merit lambda function composition printed abstracts the concept of expansion of a Megapolis as Hyponyms/Hypernyms "event","geographic area","region","definite quantitty","land" etc.,:
grow_lambda_function3(): Maximum Per Random Walk Graph Tensor Neuron Network Intrinsic Merit : (u'event((social_event(affair,(property(abstraction,(property(abstraction,(property(abstraction,(property(abstraction,(resource(abstraction,(resource(abstraction,(kind(abstraction,(abstraction(concept,(concept(kind,(information(abstraction,(abstraction(evidence,(evidence(information,(entity(abstraction,(entity(measure,(abstraction(plan,(abstraction(abstraction,(district(entity,(region(location,(entity(physical_entity,(municipality(district,(municipality(district,(region(district,(region(district,(physical_entity(location,(event(entity,(exploitation(abstraction,(use(activity,(entity(psychological_feature,(abstraction(event,(property(land,(land(abstraction,(abstraction(property,(point(entity,(farm(location,(entity(physical_entity,(location(point,(physical_entity(farm,(definite_quantity(entity,(definite_quantity(entity,(assembly(entity,(artifact(physical_entity,(assembly(entity,(geographical_area(physical_entity,(region(location,(geographical_area(physical_entity,(region(location,(entity(physical_entity,(situation(abstraction,entity))', 36.399953525147026)

----

541. (THEORY and FEATURE-DONE) Recursive Lambda Function Growth - Update - Postorder Traversal of AVL Lambda Composition Tree - 2 May 2018

----

1. Text file extracting lambda function composition tree has been updated to increase the size of the text
and more precisely represent the concept of "Expansion of Chennai Metropolitan Area".
2. Lambda Function Composition AVL Tree is evaluated as a Postfix expression (Postorder Traversal of AVL tree)
3. Maximum Merit composition from logs:

grow_lambda_function3(): Maximum Per Random Walk Graph Tensor Neuron Network Intrinsic Merit : (u'physical_entity((artifact(road,(object(event,(entity(way,(abstraction(living_thing,(entity(causal_agent,(entity(planner,(physical_entity(means,(abstraction(property,(entity(entity,(entity(entity,(abstraction(definite_quantity,(abstraction(one,(entity(auditory_communication,(music(measure,(digit(property,(relation(abstraction,(part(part,(event(abstraction,(relation(entity,(event(abstraction,(entity(relation,(politics(investigation,(investigation(politics,(entity(relation,(entity(Security_Service,(abstraction(abstraction,(group(abstraction,(international_intelligence_agency(intelligence,(international_intelligence_agency(group,(Security_Service(intelligence,(entity(abstraction,(unit(region,(entity(location,(entity(physical_entity,(district(district,(physical_entity(location,(grasping(control,(event(entity,(region(embrace,(activity(abstraction,(entity(event,(grasping(psychological_feature,(clasp(abstraction,(embrace(abstraction,(abstraction(group,(abstraction(control,(clasp(abstraction,(unit(division,(clergyman(leader,(living_thing(entity,(causal_agent(entity,(object(living_thing,(curate(leader,(clergyman(curate,(object(person,(entity(physical_entity,(thing(physical_entity,(physical_entity(external_body_part,(abstraction(entity,(abstraction(district,(entity(entity,(head(causal_agent,(group(region,(entity(physical_entity,(abstraction(abstraction,(entity(event,(abstraction(increase,(event(action,(event(abstraction,(expansion(entity,(expansion(abstraction,(action(fact,(proposal(abstraction,(information(abstraction,(abstraction(plan,(abstraction(entity,(event(plan,(entity(increase,(mathematical_relation(abstraction,(relation(entity,(location(region,(function(entity,(location(region,(entity(concept,(given(physical_entity,(physical_entity(idea,(given(entity,(abstraction(abstraction,(people(entity,(concept(abstraction,(abstraction(location,abstraction)))', 63.80089169516415)

4. This maximum merit lambda function composition has lambda functions "expansion", "mathematical_relation","region" which grasps the leitmotif of the text: "Expansion" is a "mathematical relation" on the "region" implying numeric increase in area from 1189 sqkms to 8848 sqkms
5. logs have been committed to testlogs/RecursiveLambdaFunctionGrowth.log.GraphTensorNeuronNetworkDeepStructurePostFix.2May2018
6. Classifier Core number for "expansion" is also maximum at 4.
7. Text for this is a mix of news articles from following websites:
   https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Chennai_Metropolitan_Area.html
   https://en.wikipedia.org/wiki/Chennai_metropolitan_area
   http://www.thehindu.com/todays-paper/tp-national/tp-tamilnadu/metropolitan-planning-area-expanded/article22639607.ece
   https://www.newstodaynet.com/index.php/2018/02/06/chennai-to-grow-from-metropolis-to-megapolis/

----
542. (THEORY and FEATURE-DONE) SAT Solver Update - Parity of the Assignment - 3 May 2018
----
1. solve_SAT2() function of CNFSATSolver.py has been changed to compute binary parity (sum of 1s) after rounding and real parity (sum of real values for variables before rounding)
2. Rationale is both the parities should be approximately close enough which is another measure of efficiency of MAXSAT approximation. Real parity is nothing but Riemann Sum Discrete Integral of the polynomial through real assignment points.

3. Notion of real parities is non-conventional and is introduced as proprietary NeuronRain SAT Solver feature.
4. This instance solves 1000 variables and Alpha=1.0 and MAXSAT approximation ratio is ~97%. LSQR solver
has been invoked instead of LSMR because of absence of LSMR in SciPy 0.9.0
5. Logs for this have been committed to
testlogs/CNFSATSolver.1000variablesAlpha1AssignmentParity.3May2018

---
543. (FEATURE-DONE) SAT Solver Update - SciPy Upgrade to 1.1 for restoring LSMR - 1000 clauses and Alpha=1.0
- 16 May 2018
---
1. SciPy has been upgraded to SciPy 1.1 by installing scipy 1.1 wheel file as "easy_install scipy==1.1"
2. LSMR invocation has been uncommented and invoked restoring status quo ante
3. Almost 150, 1000 variables and 1000 clauses random SAT instances have been solved and MAXSAT approximation
ratio is > 94 pegged at 94.5% which coincides with Alpha=1.0 pattern of ~95% for all executions so far.
4. Logs for this have been committed to testlogs/
5. Scipy 1.1 has initial guess parameter for x (SAT assignment) for faster convergence which can be utilized in
future versions of SAT Solver.

---
544. (FEATURE-DONE) Computational Geometric Factorization - Spark 2.3 Upgrade - Single Core - 30-bit integer
- 17 May 2018

1.Spark has been upgraded to 2.3 (oracle jdk 1.8) and Hyperbolic local tile search optimization for Computational
Geometric Factorization has been executed for a 30-bit integer on Single Core Processor.
2.Parallelism wont be perceptible in Single Core. This is just a dependency upgrade.
3.Logs have been committed to testlogs/

---
545. (FEATURE-DONE) SAT Solver Update - Initial Guess for LSMR - 1000 variables and Alpha=4.267 and 1.0
- 18 May 2018
---
1. SAT Solver has been updated to invoke lsmr() function with an initial guess assignment vector - hardcoded
to a decimal value < 1 and close to 0 for all variables.
2. Binary and Real parities (Riemann Discrete Integral Sum) are printed along side the assignment vector for each random SAT
3. SAT Solver has been executed for 1000 variables - Alpha=1.0 and 1000 variables - Alpha=4.267 for this
initial guess and logs have been captured in
testlogs/CNFSATSolver.1000variablesAlpha1.0.log.initial_guess.18May2018 and
testlogs/CNFSATSolver.1000variablesAlpha4.267.log.initial_guess.18May2018
4. As usual Alpha=1.0 converges to ~95% and Alpha=4.267 to ~90% for 150 and 20 random SAT iterations respectively.
5. Non-uniform choice probability for Alpha=4.267 is alomst close to theoretical random matrix probability 1/sqrt(m*n)

----
546. (THEORY) Depth-2 Recursive Majority Circuit and a Contradiction - related to 14,319,430 - 20 May 2018
- Updates to
https://sites.google.com/site/kuja27/IndepthAnalysisOfVariantOfMajorityVotingwithZFAOC_2014.pdf
----

Depth-2 Recursive Majority Circuit consists of two levels of Majority circuits composed as:

    Maj(Maj1,Maj2,Maj3,...,Majn)

for majority functions Maj1,Maj2,...Majn for each constituencies c1,c2,c3,...,cn respectively.This kind of social
choice function is followed by most real world democracies. For Multiway Contests binary indices 0,1 for candidates are replaced by large integers. Following is a schematic for hash table representation of depth-2 Majority voting function for 2 candidates:

```
      c1:   ########################################
            @@@@@@@@@@@@@@@@@@@

      c2:   ####################
            @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

      c3:   ################################
            @@@@@@@@@@@@@@@@@@@@
            ...
```

for constituencies c1,c2,c3,... and for candidate0 and candidate1. Votes for Candidate0 are collated in bucket marked # and Votes for Candidate1 are collated in bucket marked @ per constituency by a theoretical electronic voting machine using Locality Sensitive Hashing. Winners are:

    For c1, Candidate0 (#)
    For c2, Candidate1 (@)
    For c3, Candidate0 (#)

and in 2 of 3 constituencies, Candidate0 wins and therefore attains depth-2 majority. But if the constituency gerrymandering vanishes and votes in all constituencies for Candidate0 and Candidate1 are summed up, Candidate1 wins- Total @(s) outnumber Total #(s). Previous hash table representation of depth-2 Majority function can be translated to depth-2 Majority circuit composition Maj(Maj1,Maj2,...,Majn). In terms of Majority depth-2 composition circuit this can be written as:

    leaf: Sum(parity(c1,0),parity(c2,0),...) > Sum(parity(c1,1),parity(c2,1),...)
    depth-1: Sum(parity(c1,0),parity(c2,0),...) < Sum(parity(c1,1),parity(c2,1),...)

In other words, Total number of 0s > Total number of 1s at leaf of the Majority depth-2 circuit but Total number of 0s < Total number of 1s at depth 1 of the composition. Depth-2 majority is a direct consequence and application of Axiom of Choice in ZF Set Theory which states that there exists a choice function which chooses an element from each set in a set of sets possibly infinite.

References:

546.1 Boots and Socks Metaphor for Axiom of Choice - [Bertrand Russell] - https://en.wikipedia.org/wiki/Axiom_of_choice#cite_note-30 - AoC is necessary to choose a sock but not to choose a shoe.

----
547. (FEATURE-DONE) Graph Mining - GSpan Algorithm Implementation Update - for mining patterns in EventNet
DOT files - 21 May 2018

1.GSpan GraphMining implementation has been updated to read EventNet DOT file as input
2.This enables mining EventNet Event-Partakers Graph for patterns
3.EventNet has been implemented in NeuronRain AsFer in C++ and Python for persisting general purpose Cloud Event-Actors conversations as Graph.
4.EventNet is also an Economic Trading Network/Money Flow Market for KingCobra Fictitious Neuro Electronic Currency and Buyer-Seller networks in Neuro Money Flow Markets e.g AsFer-KingCobra Cloud Move of Neuro Currency writes out an EventNet edge in a DOT file from Buyers to Sellers
5.Mining this Economic Network EventNet requires Frequent SubGraph Mining(GSpan)
6.log for this has been committed to testlogs/GraphMining_GSpan.log.21May2018

---
548. (FEATURE-DONE) ThoughtNet - Contextual Multi Armed Bandit Evocation - Update - 21 May 2018

1.Python implementation DeepLearning_ReinforcementLearningThoughtNet.py has been updated to Classify the input
text by Recursive Gloss Overlap Definition Graph Core Number Classifier and lookup the classes in the ThoughtNet
Hypergraph
2.ThoughtNet Hypergraph has been recreated having classes mostly belonging to terrorism, economics, social progress etc.,
 extracted from chronologically older thought edges.
3.Input to ThoughtNet Reinforcement Learning has been updated to have recent events/thoughts/stimuli
4.Logs for the evocation of the new thoughts on the ThoughtNet hypergraph is in ThoughtNet/ThoughtNet_Hypergraph_Generated.txt
5.Logs show a recent terrorist event evoking an older event of similar class among others.

---
549. (FEATURE-DONE) ThoughtNet Reinforcement Learning - Contextual Multi Armed Bandit Evocation Update
- 22 May 2018
---
1. RecursiveGlossOverlap_Classifier.py has been updated to return both the definition graph NetworkX object
and Recursive Gloss Overlap Intrinsic Merit
2. RecursiveLambdaFunctionGrowth.py has been updated to return all intrinsic merits from grow_lambda_function3():
	2.1 Korner Entropy of Definition Graph
	2.2 Bose Einstein Intrinsic Fitness
	2.3 Graph Tensor Neuron Network Instrinsic Merit
	2.4 Definition Graph Density
	2.5 Definition Graph Recursive Gloss Overlap Intrinsic Merit
	2.6 Maximum Per Random Walk Graph Tensor Neuron Network Intrinsic Merit
3.ThoughtNet/Create_ThoughtNet_Hypergraph.py has been updated to make a choice for merit criterion by choosing between
sentiment scoring and graph tensor neuron network intrinsic merit for ThoughtNet HyperGraph edges
4.ThoughtNet/ThoughtNet_Hypergraph_Generated.txt has been recreated based on Graph Tensor Neuron Network Intrinsic Merit and the sorted order of hyperedges for most vertices is different compared to
sentiment scoring earlier
5.Logs for previous ThoughtNet creation has been committed to ThoughtNet/testlogs/Create_ThoughtNet_Hypergraph.GraphTensorNeuronNetwork.log.22May2018

6.Logs for ThoughtNet Contextual Multi Armed Bandit Evocation has been committed to testlogs/DeepLearning_ReinforcementLearningThoughtNet.log.22May2018

---

550. (THEORY and FEATURE-DONE) Computational Geometric Factorization - Single Core - Hardy-Ramanujan Ray Shooting Queries - 23 May 2018

---

1. Hard-Ramanujan Approximate Ray Shooting Queries function has been refactored to compute the angle and tangent
of the ray simply as functions of multiples of N/k*loglogN.
2. Precisely angle between the rays must be a function of spacing/gap between prime factors of N which is
at present assumed to be equal.
3. Cramer's bound for Spacing between primes is O(sqrt(p)*log(p)) assuming Riemann Hypothesis is true while
the best known modern bound proved in 1995 and 2001 by [Baker-Harman-Pintz] is p^(0.525) which is unconditional.
4. Prime number theorem implies prime counting function ~ N/logN. Probability that a prime below N is a prime factor of N is roughly: (loglogN*logN)/N - ratio of Hardy-Ramanujan estimate and Prime Counting Function.
5. Even with this equal spacing between prime factors, two experimental logs committed herewith show
previous approximate ray shooting is exact, passing through an exact factor found by tile binary search. Both logs have an exact match for one factor - approximate and exact ray shooting coincide.
6. By replacing equal spacing by a function of Cramer and Baker-Harman-Pintz Prime Gap estimate might accurately fine tune the location of a factor further.

References:

550.1 Gaps between Consecutive Primes - [Harald Cramer] - http://matwbn.icm.edu.pl/ksiazki/aa/aa2/aa212.pdf
550.2 Gaps between Integers with Same Prime Factors - [Cochrane-Dressler] - https://pdfs.semanticscholar.org/0dea/870aecd2513323c6f7df6b27fc31660b215c.pdf
550.3 Gaps between Consecutive Primes - [Baker-Harman-Pintz] - https://academic.oup.com/plms/article-abstract/83/3/532/1479119

---

551. (THEORY) Computational Geometric Factorization - Hardy-Ramanujan and Prime Number Theorems, Gap between
consecutive prime factors of an integer, Ray shooting queries based on these - 24 May 2018

From [Baker-Harman-Pintz] estimate Gap between primes p(n) and p(n+1) is approximately (p(n))^0.525. Combining
Hardy-Ramanujan and Prime Number Theorem following estimate of number of primes between two prime factors of an
integer N can be found:

Approximate number of primes between 2 prime factors of N = (Number of primes less than N) / (Number of prime factors of N) ~ N/((logN)*(loglogN))

If pf(n) and pf(n+1) are two consecutive prime factors of N, then from Prime Number Theorem number of primes between pf(n) and pf(n+1) = (pf(n+1)/log(pf(n+1)) - pf(n)/log(pf(n))) which is equivalent to Hardy-Ramanujan and Prime Number Theorem estimate N/((logN)*(loglogN))

=> pf(n+1)/log(pf(n+1)) - pf(n)/log(pf(n)) = N/((logN)*(loglogN))
=> pf(n+1)/log(pf(n+1)) = pf(n)/log(pf(n)) + N/((logN)*(loglogN))

Previous relation finds the next prime factor pf(n+1) in terms of pf(n) as a logarithmic ratio. Similar recurrence can be arrived at based on [Baker-Harman-Pintz] estimate too. If p1,p2,p3,...,pm are primes between pf(n) and pf(n+1):

```
p1 = pf(n) + (pf(n))^0.525
p2 = p1 + p1^0.525
p3 = p2 + p2^0.525
...
pf(n+1) = pm + pm^0.525
```

Expanding this recurrence yields a continued nested exponentiation of 0.525 for pf(n+1) in terms of pf(n) which must be equal to Hardy-Ramanujan-Prime Number Theorem estimate N/((logN)*(loglogN)). This estimate for gap between two consecutive prime factors of an integer can be substituted for computing angles for prime factor points ray shooting queries in Computational Geometric Factorization in lieu of equal spacing.

---

552. (FEATURE-DONE) Computational Geometric Factorization - Ray Shooting Queries - Hardy-Ramanujan-Prime Number Theorem and Baker-Harman-Pintz Gap between Prime Numbers - 24 May 2018

1.DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been updated to include two new
functions for ray shooting queries - 1) based on Hardy-Ramanujan-Prime Number Theorem estimate for number
of primes between two prime factors 2) based on Baker-Harman-Pintz Gap between prime numbers.
2.These two functions are quite nascent and might be refined in next commits.
3.Two numbers have been factored and all three ray shooting query algorithms print the approximate location of
factors.
4.Of the three, Hardy-Ramanujan query is very simple and finds the factor exactly in most executions (logs show an exact match for one factor in both integers).
Hardy-Ramanujan-Prime Number Theorem and Baker-Harman-Pintz estimates are too skewed. But theoretically latter two estimates are tighter approximations compared to Hardy-Ramanujan ray shooting.
5.Logs for these two integer factorizations have been committed to testlogs/

---

553. (THEORY and FEATURE-DONE) Computational Geometric Factorization - Ray Shooting Queries and Cramer Prime Gap Updates - 29 May 2018

1.DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been updated to include a new ray shooting query function for Cramer Prime Gaps (sqrt(p)*log(p))
2.All Four Ray Shooting Query functions have been executed on two integers and factorization logs have been
captured in testlogs/ (Single Core - Spark 2.3)
3.Both logs don't show an exact shoot (but are sufficiently close)
4.Of the Four, Hardy-Ramanujan ray shooting is still the best and estimates are close to exact factors and sometimes exact.
5.Baker-Harman-Pintz, Hardy-Ramanujan-PrimeNumberTheorem Ray Shooting functions have been slightly changed for
initial prime factor, constant for Prime Counting Function normal order etc.,

---

6.Analysis:

These ray shooting queries are Monte Carlo algorithms with an error bounded by the spacing between two prime

factors. Ray shooting to find a prime factor by Hardy-Ramanujan succeeds with a probability loglogN/N which is
 loglogN/2^logN=loglogN/(2^(2^loglogN)) which is exponentially small. But real benefit of ray shooting queries is in the following factorization algorithm:
	- Sequentially ray shoot O(loglogN) queries in O(loglogN) time to create beacons
	- Distance between any two such beacon in the hyperbolic arc bow is O(N/loglogN)
	- This creates O(loglogN) hyperbolic pixelated segments of length O(N/loglogN)
	- Pixelation of O(N/loglogN) sized segment requires Parallel RAM sorting (e.g. Using Cole's Parallel Sort each pixelated segment of length O(N/loglogN) can be sorted in O(log(N/loglogN)) on PRAMs.)
	- Each pixelated segment can be binary searched in log(N/loglogN) time
	- Sequentially this requires O(loglogN)*O(log(N/loglogN)) time for all segments
	- Total factorization time = O(loglogN) + O(loglogN)*O(log(N/loglogN)) + O(loglogN)*O(log(N/loglogN)) < O(loglogN) * O(logN)
	- This PRAM algorithm does factorization in at most O(loglogN*logN) time compared to O((logN)^2) or O((logN)^3) PRAM time algorithms using segment trees and sorting described earlier.

---
554. (FEATURE-DONE) Scheduler Analytics Update - Rewrite of scheduler analytics webserver functionality and Spark log mapreduce of "perf sched script" - 30 May 2018

1.SparkKernelLogMapReduceParser.py has been rewritten to remove hardcoded filenames and patterns - new function log_mapreducer() has been defined for passing log file name and pattern as arguments.
2.software_analytics/DeepLearning_SchedulerAnalytics.py and software_analytics/SchedulerAnalytics_WebServer.py h$
been refactored and get_stream_data() (@Generic Socket Web Server) decorated function has been moved to software$
3.software_analytics/DeepLearning_SchedulerAnalytics.py invokes log_mapreducer() defined in SparkKernelLogMapRed$
 for new scheduler performance file perf.data.schedscript and pattern sched_stat_runtime.
4.New perf.data file has been created from perf utility by "perf sched record" commandline
5.New perf.data.schedscript has been created from perf.data by "perf sched script" commandline for printing
scheduler context switches and runtime in nanoseconds
6.logs for both have been committed to software_analytics/testlogs/DeepLearning_SchedulerAnalytics.log.30May2018 and software_analytics/testlogs/SchedulerAnalytics_WebServer.log.30May2018
7.Perf utility support has been added for potential clockticks-to-processes dynamic hash table creation later.

---
555. (THEORY) Non-majority social choice, Majority social choice - Boolean Composition of Majority+SAT Versus SAT Oracles, Margulis-Russo Threshold , KRW Conjecture, Condorcet Jury Theorem and a conflict - 4 June 2018 - related to 129, 317, 355, 358, 400, 429 and all other Majority Voting related sections
---
In the theoretical drafts in this document, relativizing proofs involving oracle access to Voter SAT have been
neglected in favour of boolean function composition/communication complexity proofs based on composition of
Majority and Voter SAT boolean functions. But a recent result has proved BQP^A

is not contained in PH^A relative
to some oracle A. This has direct relation to Condorcet Jury Theorem circuit for Majority Voting having SAT oracle access. P(Good) binomial series (or Condorcet Jury Theorem) equates Non-majority and Majority social choice based on correctness of electorate group decision vis-a-vis individual decision correctness. This is tantamount to equating goodness of following two algorithms in different complexity classes:

*) LHS: An algorithm having access to PRG making a random choice of a boolean function(voter) - could be in BPP/BQP/RP/BPNC/RNC

*) RHS: An algorithm which does majority voting by composing Majority function and Individual voter SATs - could be in PH (DC-uniform) if the number of variables m, per Voter SAT are same for all and size of the electorate is n=2^m - exponential fanin - and depth restricted (unrestricted is in EXP).

Assuming both LHS and RHS have equal goodness (1-error) of 0.5 (by Margulis-Russo Threshold and Condorcet Jury Theorem), which happens when each individual votes with correctness p=0.5 (homogeneous voters), Oracle version of the previous equality is (for some oracle Voter SAT A which has goodness p=0.5):

*) LHS: BPP^A / BQP^A / RNC^A / BPNC^A

*) RHS: PH^A / EXP^A / Other larger classes

Caveat: This assumes equal error/goodness implies lowerbounds and there are complete problems for respective classes.

Since goodness/error is equal, if one side of the equality is a C-complete problem, the other side devours class C. E.g:

*) if LHS is BQP^A-complete and RHS is PH^A, BQP^A is in PH^A - this is in direct conflict with 555.1 which separates BQP and PH relative to some oracle A.

*) if RHS is PH^A-complete and LHS is BQP^A, PH^A is in BQP^A

Resolving this conflict requires:

*) LHS and RHS have different Oracles (Pseudorandom Oracle A1 for LHS and SAT Oracle A2 for all voters in RHS) - but in real-world voting nothing prohibits all voters from making a random choice and thus there is always a possibility electorate votes at random (Therefore Both Oracles are Pseudorandom Generators and A1=A2. This is exactly Balls-and-Bins problem described in sections on theoretical EVMs based on LSH and separate chaining hashtables elsewhere in this draft).

*) ruling out existence of BPQ^A-complete problems for Voter SAT oracle A separating PH and BPQ.

*) assumption of equal goodnesses of LHS and RHS implying lowerbounds is wrong.

*) Oracle A which separates (BQP^A is not in PH^A) has goodness > 0.5 or < 0.5 violating Margulis-Russo threshold.

But if LHS is Promise-BPP-Complete which are known to exist and RHS is PH^A, Promise-BPP is in PH^A.

References:

555.1 Oracle Separation of BQP and PH - [AvishayTal-Raz] - https://eccc.weizmann.ac.il/report/2018/107/download
555.2 Margulis-Russo Threshold for Majority at p=0.5 - [RyanODonnell] - Pages 224-225 - http://www.cs.tau.ac.il/~amnon/Classes/2016-PRG/Analysis-Of-Boolean-Functions.pdf
555.3 Promise-BPP-Complete problems - [OdedGoldreich] - http://www.wisdom.weizmann.ac.il/~oded/PSX/prpr-r.pdf

---

556. (THEORY and FEATURE-DONE) Computational Geometric Factorization - Ray Shooting Queries - Yitang Zhang Prime Gap Estimate - 5 June 2018

1.DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been updated to include a new function
for ray shooting queries which implements recent breakthrough result in Prime Gap estimation - there are
infinitely many twin primes separated by gap < 70000000 - by [Yitang Zhang] which refines
[Goldston-Pintz-Yildirim] Sieve. This might have effect only for finding primes between prime factors of
large integers and summing up the gap for spacing between any two prime factors.
2.Constants k and l for multiplying estimation bounds have been changed slightly.
3.logs for this have been committed to testlogs/ for an 8-digit 26-bit integer.
4.None of the five ray shooting queries in the bouquet shoot exactly. But Hardy-Ramanujan is still the best and
Baker-Harman-Pintz estimate is also close enough to actual factors.
5.Ray Shooting's success rate depends on the number and multiplicity of the prime factors - Integers of large Big Omega and Small Omega would have lot of prime factors taking a point-blank hit. In other words large multiplicative partition (Factorisatio Numerorum) number of an integer is a pre-requisite for exact ray shooting. Finding the values of the constants for estimation functions which are normal orders, is presently done by trial and error. Correct values for these constants are also crucial for exact shooting.
6.Another weird observation is the convergence of the Hardy-Ramanujan-PrimeNumberTheorem log ratio (p/log(p), also known as Merit) estimate after some iterations and this converged integer sometimes matches closely with actual factors. This has happened in earlier logs too.

References:

556.1 Bounded Gaps between Primes - [Yitang Zhang] - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.308.998&rep=rep1&type=pdf
556.2 Goldston-Pintz-Yildirim Sieve - [K.Soundararajan] - https://arxiv.org/pdf/math/0605696.pdf - there are infinitely many twin primes such that lim sup (p(n+1) - pn) / log(pn) ~ 0 - there exist prime tuples which are arbitrarily close apart
556.3 Primes in Tuples - [Goldston-Pintz-Yildirim] - https://arxiv.org/pdf/math/0508185.pdf

---

557. (THEORY and FEATURE-DONE) Computational Geometric Factorization - Ray shooting queries - correction - 6 June 2018

1.DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been updated to correct the
number of primes estimate between two factors - constant for prime counting function (merit) O(p/logp) has been
moved to numerator which was earlier in denominator incorrectly.
2.Two integers have been factored and logs have been committed to testlogs/. Both show exact ray shoot finding
factors correctly matching with actual factors. These are on Single Core + Spark 2.3 with local[4]. This
simulates quadcore on single core and some factors are printed out of order probably because of Spark Driver
Executor Threads.

---

558. (THEORY and FEATURE-DONE) Computational Geometric Factorization - Hardy-Ramanujan-PNT Ray Shooting - Correction - 7 June 2018

---
1. Mysterious convergence of merit p/logp has been investigated.
2. This estimate equates the Prime number theorem merit difference between 2 prime factors and ratio of Prime
counting function and Hardy-Ramanujan prime factors estimate as below:
l*pf(n+1)/log(pf(n+1)) - l*pf(n)/log(pf(n)) = l*N/(logN * k*loglogN)
3. Earlier the constant multiples k and l were missing in some places which has been rectified.
4. Even after this, previous ratio converges after lot of iterations for l=10 and k=6. So suitable value of l
is still elusive.
5. Two integers with large Omega (usually numbers ending in 5 have lot of factors) have been factored. While
one integer has exact shoot from Hardy-Ramanujan, the other has close shave, but approximate factors are quite
in proximity to actual factors.
6. This establishes utility of ray shooting queries as an Oracle access implementation which can be used both
by classical P and quantum period-finding BQP (P^A versus BQP^A) Factoring Turing Machines which at most requires
O(loglogN) queries.

---
559. (THEORY and FEATURE-DONE) Computational Geometric Factorization - Ray Shooting Queries - Constant Changes
- 8 June 2018

1.Constants in all ray shooting query functions have been changed so that gaps are reduced and possibility
of exact hit increases. Specifically constant l for Cramer, Baker-Harman-Pintz, Zhang has been made small to
widen xrange().
2.2 integers have been factored and approximate factors for all 5 queries are printed after these new constant values. Zhang and Cramer queries were overshooting previously beyond N. None of the queries exactly shoot but are quite close to actual factors.

---
560. (THEORY and FEATURE-DONE) SAT Solver Update - nonuniform_choice3() damp - 2000 variables and Alpha=4.267
- 12 June 2018

1.CNFSATSolver.py - damp variable in nonuniform_choice3() function has been fine tuned to match the
theoretical probability of 1/sqrt(m*n) - observed probability almost matches with theoretical
2.random 3SAT instances for Number of
variables=100,200,300,400,500,600,700,800,900,1000,1100,1200,1300,1400,1500 and 2000 for alpha=4.267
have been solved and observed probability closely matches the random matrix probability
3.Logs for 2000 variables and Alpha=4.267 have been committed to
testlogs/CNFSATSolver.2000clausesAlpha4.267.log.12June2018.
4.Observed MAXSAT approximation ratio for first few iterations hovers around 90-91-92% for all previous
variable combinations.
5.Theoretically, if damp variable is perfectly tuned to create required bias in nonuniform choice, then MAXSAT
ratio observed for unequal variable-clause combinations should also be in the range of 94-95% if not 100%.

---
561. (FEATURE-DONE) Scheduler Analytics Update - New Convolution Network for analyzing stream of 2-dimensional graphics performance image data - 14 June 2018

Pictorial Performance Pattern Mining by Convolution Network:

Input to this Convolution Network is an image bitmap matrix obtained from some standard graphic
performance analyzers - runqueuelat, BPF/bcc-tools, perf etc., Following example image is created
by "perf timechart" on perf.data which captures the state,timeslice etc., information of list of
processes in scheduler at any instant in the form of a gantt chart. Backpropagation-Convolution is
then applied to this performance snapshot and convolution, maxpooling, final neuron layers are
computed as usual. This convolution is iterated periodically for stream of performance snapshot bitmaps.
Advantages of mining graphic images of performances captured periodically are:
- number of dimensions of a performance snapshot image is 2 while psutils creates 1 dimensional
array stream of data per process id.
- 2 dimensional performance snapshots are system wide and capture all process id(s) at a time
and not process id specific.
- Recent advanced performance analyzers like BPF/bcc-tools provide a histogram equivalent of
timechart in perf (runqueuelat) which is also an image bitmap. So support for graphic performance
data analytics is futuristic.
- Processing stream of performance snapshot image bitmaps by convolution to extract patterns is
more universal/holistic than applying convolution to stream of process structure data
- Specifically if remaining_clockticks-to-processes dynamic hash tables are available as
stream of histogram images, applying convolution on this stream creates frequent recurring
patterns of queueing in OS Scheduler (Described in GRAFIT course material:
-
https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt,
-
https://gitlab.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt
)

NOTE on linux kernel versions: All the NeuronRain commits being done (in SourceForge and GitHub) for the past 2 months since April 2018 are based on executions/tests on linux kernel 3.x.x + Single Core. Berkeley Packet Filtering(BPF)/bcc-tools performance analyzer which includes runqlat is recent addition to mainstream linux kernel from version 4.x.x onwards. Commits prior to April 2018 were done on a linux kernel 4.13.3 + Dual Core development environment which was sabotaged by a probable severe Cybercrime in first week of April 2018 causing catastrophic damage to partitions and hardware failure too making recovery impossible. This incident caused total destruction of all dev and build setups necessitating full checkouts from GitHub and SourceForge

repositories and continued development on another base linux kernel of lesser version which does not support BPF/bcc-tools. Previous provision for graphic performance data pattern recognition is mainly targeted towards supporting BPF/bcc-tools in future.

---

562. (THEORY) Computational Geometric PRAM-NC Factorization, Ray Shooting Query Oracles, Quantum NC, Quantum Factorization, Classical-Quantum Transition and Knot Theory - related to 34,351,555 - 15 June 2018 and 18 June 2018

Earlier sections define various Ray Shooting Query functions on Hyperbolic curve to find approximate factors - Hardy-Ramanujan, Prime Number Theorem, Baker-Harman-Pintz, Zhang, Cramer etc.,These Ray shooting functions are utilized as supporting preprocessing oracles to k-mergesort, local tile search, segment trees based PRAM factorization algorithms - oracles locate prime factors approximately and sometimes exactly in sequential time of O(loglogN). If an oracle ray shooting query succeeds, factorization is over in O(loglogN) steps with no necessity for further pixelation of hyperbolic curve and mergesort/search for factors in PRAM-NC - of parallel time O((logN)^2) or O((logN)^3). If an oracle ray shooting query is approximate, PRAM k-mergesort/search phase of factorization is taken up using the approximate factors queried by ray shooting as beacons - search in vicinity of approximate factor should easily locate the exact factor. Therefore a factorization involving both ray shooting queries and k-mergesort/segment trees/tile search is a problem in complexity class NC^A where A is a ray shooting query oracle in P or NC (if O(loglogN) parallel ray shooting queries are made on O(loglogN) PRAMs in O(1) time). Similar to classical NC, Quantum NC is the class of problems solvable in polynomial size quantum circuits (controlled-U gates, controlled-not gates, Hadamard gates) in polylogarithmic time. There exists an oracle A such that P^A=BQP^A and NC^A is contained in P^A. Also BQP is contained in AWPP i.e. any quantum computation can be simulated by a GapP function f(x) computed by a NPTIME Turing Machine (= number of accepting paths of x - number of rejecting paths of x) such that:

Pr(M(x) accepts] = f(x)/5^(2*t(x))

for a Turing Machine M(x) in BQP. Since NC is contained in QNC, PRAMs in NC Factorization can be termed as decohered quantum gates in corresponding QNC algorithm. Since Every quantum computation can be transformed into a braid evaluated by Jones Polynomial, an oracle A of Jones Polynomial can be non-constructively shown to exist such that quantum part in a BQP Turing Machine is mapped to a deterministic computation. Theorem 6.1 in reference 562.4 defines such a Jones polynomial PTIME braid oracle A so that BQP is contained in P^A.

Computational Geometric Factorization involves both Parallel and Sequential phases - Pixelating hyperbolic arc bow and k-mergesorting/segment-tree them requires parallel RAMs whereas binary search of k-mergesorted/segment-tree tiles can be done by sequential binary search with no necessity for PRAMs. If each PRAM is replaced by a Quantum Gate, computational geometric factorization can be done in QNC instead of NC which is contained in BQP. Following algorithm maps the mergesort and search of hyperbolic tiles as below and does away with sorting and searching.

Pixelated Hyperbolic arc bow as superposition of quantum state pixels

---

Set of pixel coordinates in pixelated hyperbolic arc bow can be construed as superposed pixel quantum states each of some amplitude:

|(p,q)> = B1|(x1,y1)> + B2|(x2,y2)| + ... + Bn|(xO(N),yO(N))>

for complex amplitudes Bi of O(N) coordinate states in pixelated hyperbolic arc. This superposition implies all the coordinates of hyperbolic arc simultaneously have certain probability of being the factor of N. Factorization reduces to maximizing probability of decoherence of this superposition to correct classical factor state |(xl,yl)> i.e xl*yl=N.

Number of distinct prime factors of N = kloglogN for some constant k from Hardy-Ramanujan Theorem. Set of all factors = Set of all multiplicative partitions of N. Average Probability of a coordinate in pixelated hyperbolic arc to be the factor of N = O((Multiplicative Partition Number of N) / N). But square of complex amplitude Bi^2 is the classical probability of a pixel coordinate state |(xi,yi)>. Each state |(xi,yi)> is created by a quantum toffoli multiplication gate and requires 2*logN qubits. Amplitude Bi for each state |(xi,yi)> can be defined as the complex number function of coordinate:

Bi = f(xi,yi) + i*g(xi,yi)

Classical probability of being a factor of N = Bi^2 which is 1 for factor points. Solving Bi^2 = 1, yields f and g.

Classical probability ((modulus(Bi))^2) of pixel coordinate state |(xi,yi)> - assumed to be normalized so that
Sum(Bi^2) = 1 - can be defined as below:

(modulus(Bi))^2 = f(xi,yi)^2 + g(xi,yi)^2 = floor(1 - abs(xi*yi/N - 1))

which is 1 only if xi*yi = N, the factor point, and is 0 if xi*yi/N > 1 or xi*yi/N < 1 , the non-factor points on pixelated hyperbolic arc bow, because floor() makes the enclosing fraction to 0.

Quantum NC equivalent of Classical PRAM-NC Computational Geometric Factorization:

- Input to the quantum NC circuit is the equation for hyperbola, xy=N
- Length of pixelated hyperbolic arc bow is O(N) or approximately 1.5N-1
- For factorization it suffices if each pixel (p,q) in the pixelated hyperbola is input to a quantum gate and it is verified by all gates in parallel if pq=N
- Succeeding multiplication gates with pq=N find factors p,q in parallel. This is similar to multiplication in modular exponentiation subroutine of Shor Factorization (or) a quantum multiplication circuit similar to reference 562.7
- Pixelated Tiles and their pixel coordinates are found in parallel by tiling equation N/(x(x+1))
- Each multiplication gate for p,q has 2*logN bits inputs - |p| + |q|
- Number of multiplication gates required < 1.5N-1 (adheres to QNC definition if input size is N and not logN)
- No sorting and search is necessary
- Modular multiplication of pixel coordinates (p,q) = pq mod N and verifying it to be 0 is sufficient to conclude p,q are divisors/factors of N. Succeeding Quantum Modular Multiplication Gates have pq mod N = 0 and each modular multiplication gate is of polylogarithmic depth
- Exact sum of tiles of pixelated hyperbolic arc bow:

N/(1.2) + N/(2.3) + ... N/((N-k)(N-k+1)) = N*{1/1 - 1/2 + 1/2 - 1/3 + 1/3 - 1/4 + ... + 1/(N-k) - 1/(N-k+1)}

= N*(1 - 1/(N-k+1)) = N*(N-k)/(N-k+1)

But tiling ends when N=(N-k)(N-k+1) or N=N^2 - 2Nk + N + k^2 - k

=> k^2 - k(2N+1) + N^2 = 0

(or) k = [(2N+1) +/- sqrt(4N^2 + 4N + 1 - 4N^2)]/2

=> k = [(2N+1) +/- sqrt(4N + 1)]/2

=> k ~ N - sqrt(N) + 0.5 = Approximate Number of tiles(arrays of pixels)

=> Sum of tile lengths = N*(N-N+sqrt(N)-0.5)/(N-N+sqrt(N)+0.5)

=> Sum of tile lengths = N*(sqrt(N)-0.5)/(sqrt(N)+0.5) ~ 0.999999...*N for

large N
      - Tiling phase also requires Quantum multiplication gates e.g Division in N/(x(x+1)) is converted to a multiplication circuit ([BeameCookHoover] - division reduces to multiplication)

References:

562.1 Quantum NC - https://arxiv.org/pdf/quant-ph/9808027.pdf
562.2 Shor's Factorization simulated on GPUs (comparison to Microsoft LIQuiD Quantum simulator) - https://arxiv.org/pdf/quant-ph/9808027.pdf
562.3 Complexity limitations on Classical Computation - [Fortnow-Rogers] - https://arxiv.org/pdf/cs/9811023.pdf - BQP is in AWPP (Lemma 3.2) and P^A=BQP^A (Theorem 4.1)
562.4 Approximate Counting and Quantum Computation - [M. Bordewich, M. Freedman, L. Lovasz, D. Welsh] - http://web.cs.elte.hu/~lovasz/morepapers/additive1.pdf - "...A classical polynomial time algorithm can convert a quantum circuit for an instance of such a problem, into a braid, such that the probability that the output of the quantum computation is zero, is a simple (polynomial time) function of the Jones polynomial of the braid at a 5th root of unity. For an exact statement of this see Freedman, Kitaev, Larsen and Wang [5], or the more detailed papers by Freedman, Kitaev and Wang [6], and Freedman, Larsen and Wang [7, 8]. ..." and "...The Kitaev-Solvay theorem, [K, S] together with the density theorem ,Freedman, Larsen and Wang [7, 8], yields an algorithm for an approximation of any gate by a braid of length polylog (1/e) under a Jones representation. We take the number of strings m, as the size of the input, however the number of crossings in the link is the same as the number of gates in the BQP computer, which is bounded by a polynomial in the input size, hence an algorithm will be polynomial with respect to both measures (or neither)..." - Theorem 6.1 - "Let A(L) be an oracle that returns the sign of the Jones polynomial of the link L evaluated at e^2πi/5 . Then BQP ⊆ P^A"
562.5 Jones Polynomial, Knot Theory and Equivalence of Quantum and Classical Computation - https://en.wikipedia.org/wiki/Jones_polynomial - related to 562.4 - Quantum part of any computation can be replaced by approximate evaluation of Jones Polynomial of certain braid under a Jones representation.
562.6 Shor Quantum Factorization - [Peter Shor] - (1995) and (1996) - https://arxiv.org/abs/quant-ph/9508027 - modular exponentiation and Quantum Fourier Transform of an integer state |a>
562.7 Quantum Karatsuba Integer Multiplication - https://arxiv.org/pdf/1706.03419.pdf
562.8 Quantum Multiplication Modulo N Gate Example - https://www.math.ksu.edu/reu/sumar/QuantumAlgorithms.pdf - Depth of multiplication gates is polylogarithmic in N.
562.9 Log Depth Circuits for Division and Related Problems - [BeameCookHoover] - https://pdfs.semanticscholar.org/29c6/f0ade6de6c926538be6420b61ee9ad71165e.pdf
562.10 Quantum Machine Learning - HHL Algorithm - https://www.scottaaronson.com/papers/qml.pdf - HHL solves system of linear equations Ax=B in quantum logarithmic time. Quantum RAM loads the classical amplitudes of all states - Quoted excerpts - "...If the HHL algorithm could be adapted to the case where (say) a single bi is 1 and all the others are 0, then it would contradict the impossibility of an exponential speedup for black-box quantum search, which was proved by Bennett, Bernstein, Brassard, and Vazirani...". This is relevant in the context of pixel state superposition above i.e if exactly one Bi corresponding to a factor point is made to 1 and all others are 0, factorization immediately follows. HHL algorithm fits well in the context of least squares SAT solver also which maps 3SAT clauses to system of linear equations and finds assignments approximately by solving random matrix equation, Ax=B.

---
563. (THEORY and FEATURE-DONE) Random 3SAT Approximate LSQR/LSMR Solver update - Increase in Accuracy
- 19 June 2018

1.CNFSATSolver.py has been updated to increase the accuracy of least squares approximation of SAT assignments
by setting atol=btol=conlim=0 as prescribed by SciPy documentation in :
https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.sparse.linalg.lsmr.html
Also set the damp to a low value close to zero to solve the 3SAT as regularized least squares problem which is
a generalization of least squares.
2.Random 3SAT instances of 1000 variables and Alpha=1.0 have been solved and MAXSAT approximation ratio for this
converge to ~97% for few iterationsA
3.This increase in accuracy slows down LSMR. But by better least squares optimization which is parallelizable on cloud viz., LSRN - http://web.stanford.edu/group/SOL/software/lsrn/ which is recent parallel algorithm for
solving least squares (LSRN paper - http://web.stanford.edu/group/SOL/papers/lsrn-sisc-2014.pdf) this slow down can be remedied. Invoking LSRN instead of LSMR implies SAT Solver can be executed on cloud.

---

564. (THEORY and FEATURE-DONE) SAT Solver Update - Randomized Rounding and other linear system of equation solvers - 21 June 2018

1.SAT Solver has been updated to include a special if condition to check if Alpha=1 and directly invoke
linear system of equations solvers instead of least squares regression solvers. Various system of equations
solvers like minres,cg,solve and their variants in ScipPy were tried for number_of_clauses==number_of_variables
 and minres was found to be the best equalling lsmr's accuracy. But they have been commented and lsmr() is
invoked with no regularization damp for Alpha=1
2.Randomized Rounding of the fractional real assignments computed from least squares has been refined. Earlier
the rounding threshold was set to 0.5 which has been generalized to the midpoint of the assignment array ((min + max)/2)
3. This midpoint computation is important because it finds the halfway inflexion point of the polynomial traversing the real assignment points. This polynomial is approximated to a step function of 1s and 0s based on whether the assignment for a variable is above or below half-way mark.
4. Polynomial through the fractional assignment points is sinusoidal and it crosses this midpoint in as many places as there are 0-to-1 and 1-to-0 inflexions.
5. Another randomized rounding scheme is also implemented by computing a pseudorandom fraction between 0 and 1 as
threshold. Midpoint threshold is quite accourate and is mostly 0.5 which is on the expected lines, while values
other than 0.5 are caused because of negative values to some variables in least squares. This follows the usual
randomized rounding convention (e.g for Set Cover).
6. Few Random SAT instances of 1000 variables and Alpha=1 have been computed and MAXSAT approximation ratio has been found to be 97-98% which is captured in logs. In some other executions, this ratio touched even 98.5%
7. Accuracy of randomized rounding threshold is crucial to maximizing MAXSAT ratio.
8. Quantum HHL algorithm solves system of equations Ax=B in logarithmic time while previous classical solvers
are polynomial in clauses-variables. Theoretically this SAT Solver invoking HHL on a quantum computer should be
able to solve SAT NP-Complete problem approximately in logtime and thus in BQ-P (or BQ-NC).

---
565. (FEATURE-DONE) Query Index and Ranking - Sorting by merit - 22 June 2018

1.QueryIndexAndRank.py has been changed to sort the results queried from LSH and ThoughtNet indices by
descending order of Graph Tensor Neuron Network Intrinsic Merit. Earlier sorting was missing. Two
dictionaries have been created for LSH and ThoughtNet index rankings and are sorted descending.
2.Locations of ThoughtNet Hypergraph have been updated in Indexing/
3.Logs for the ranking have been captured in
testlogs/QueryIndexAndRank.log.22June2018

---
566. (THEORY) Exact Learning of Boolean Functions(e.g absolute CNF) and Intrinisic Merit - 22 June 2018
- related to 529

Intrinsic Merit problem has been defined as MAXSAT problem earlier assuming there exists an absolute universal
CNF which accepts merit variable values from texts/AVs. Merit variables in this universal CNF are boolean
 based on threshold: if a merit value > threshold, merit variable is 1 and 0 otherwise. For example if graph
tensor neuron network intrinsic merit of a text exceeds a threshold, corresponding variable "intrinsic_merit"
in universal CNF is set to 1. Texts/AVs are ranked by descending order of percentage of clauses satisfied in this
universal CNF which is MAXSAT problem. Problem is how to learn this universal CNF if it exists. There are
approximate learning algorithms like PAC learning which can learn a concept class approximately correctly
with a bounded error. Traditional exact learning model uses halving algorithm based on membership and
equivalence queries to gradually shrink the concept class to 1 element. Number of queries required are log|C|.
Membership queries ask an oracle if x is in C and equivalence queries ask an oracle if f(x)=h(x). This halving
algorithm is analogous to interval halving proof of Bolzano-Weierstrass Theorem which states that every infinite sequence has a convergent monotone subsequence. Applying this halving procedure to set of all texts and AVs couldexactly learn a universal CNF in logarithmic number of Angluin model membership and counterexample/equivalence
queries.

References:

566.1 Oracles and Queries That Are Sufficient for Exact Learning - [Nader H. Bshouty, Richard Cleve, Ricard Gavaldah, Sampath Kannan and Christino Tamon] - https://ac.els-cdn.com/S002200009690032X/1-s2.0-S002200009690032X-main.pdf?_tid=735940ad-7047-40d2-a9a7-2575422672c3&acdnat=1529665709_798fd8cba3f740826f4c3ce299cdb6c1
566.2 Bolzano-Weierstrass theorem and its application to MARA - https://en.wikipedia.org/wiki/Bolzano%E2%80%93Weierstrass_theorem

---
567. (FEATURE-DONE) Streaming Analytics - Facebook Graph API - Python 3.4 and facebook-sdk - 29 June 2018

1.Similar to Streaming_<datasource>.py for LinkedIn and Twitter,
Streaming_FacebookData.py has been added to
NeuronRain AsFer repositories in SourceForge,GitHub and GitLab.
2.This imports facebook-sdk for python 3.4. This is the first python
implementation on python 3.4.
3.This upgrade to python 3.4 from python 2.7 is necessitated because:
	3.1 facebook-sdk on python 2.7 requires python 2.7.9
	3.2 But importing facebook-sdk on Python 2.7.9 raises errors related to
_ssl and _hashlib (md5,sha etc.,)
which are not by default supported by python 2.7.9
	3.3 Enabling SSL flags in Modules/Setup.dist for python 2.7.9 non-system
builds also does not resolve this error and ssl is not built along with python
	3.4 Facebook sdk supports only python 3.4 and 3.7 in 3.x
	3.5 Python3.4 has been installed from a non standard PPA
	3.6 VirtualEnv, EasyInstall3, PIP also do not solve the above SSL import
issue
	3.7 In python3.4 too, lot of facebook-sdk dependencies have to be git-
cloned,built and installed (requests, idna, certifi, chardet etc.,) for facebook
import to work
4. Example temporary access token has been created from from
https://developers.facebook.com/tools/explorer/
5. Simple name and id REST query has been done using requests and logs committed
to testlogs/

---
568. (FEATURE-DONE) Streaming Analytics - Facebook Graph - 2 July 2018

1.Streaming_FacebookData.py has been parametrized to accept user name as
commandline argument
	$python3.4 Streaming_FacebookData.py <facebook_user>
2.Wall Posts and Friends of a user are obtained by facebook-sdk REST
3.Friends generator object is iterated and printed

---
569. (THEORY and FEATURE) SAT Solver - Discrete Fourier Transform (FFT) of the
real assignment,
Parseval/Plancherel Theorems, Semidefinite Programming Approximation and Least
Squares - 3 July 2018
---
1. SAT Solver has been updated to compute Fast Fourier Transform of the real
assignment points for each
random 3SAT.
2. Discrete Fourier Transform of set of points is:
	X(N) = Sum(x(n) * e^(2*pi*n*k/N)), k=0,1,2,...,N-1
X(i) are frequencies and x(i) are discrete points in real assignment vector
computed from Least Squares
3. Mapping the real assignment points to frequency domain enables finding
periodicity within the real assignments and
applying Parseval/Plancherel Theorems
(https://en.wikipedia.org/wiki/Discrete_Fourier_transform)
4. Parseval Theorem equates the inner product of two vectors to inner product of
Fourier Transform Frequencies.
5. Inner product of two assignment vectors (exact binary and real approximate -
for values of variables)
is a measure of distance between the exact (if known) and approximate
assignments
6. Least Square Approximation solves the system of linear equations AX = B for
random 3SAT while Semidefinite

Programming (SDP) solves the Convex Optimization Problem:
maximize/minimize: Double_Summation(a(i,k)*xi * yk)
Subject to Constraints: Double_Summation(cl*xi*yk) < bl
7. In other words, SDP solves:
maximize/minimize: A*X
Subject to Constraints: C*X < B
X is a Positive Semidefinite Gramian Matrix having entries from inner product space of vectors
(or v*X*v^T is non-zero for real vectors v)
8. [Goemans-Williamson] MAXCUT Approximation algorithm - http://www-math.mit.edu/~goemans/PAPERS/maxcut-jacm.pdf -Journal of the Association for Computing Machinery,vol. 42, No.6, November1995 - solves MAXCUT in ~88% by SDP.
9. Any symmetric matrix can be approximated to nearest positive semidefinite matrix (e.g cov_nearest - http://www.statsmodels.org/dev/generated/statsmodels.stats.correlation_tools.cov_nearest.html)
10. There is an SDP variation of least squares, Semidefinite Least Squares, which solves the convex program:
minimize: |X-C|^2
Subject to: AX=B for positive semidefinite matrix X.
which is similar to least squares solution for variable vector X. X is positive semidefinite because for any real
vector v and some assignment to X, v*X*vT is positive. Minimizing |X-C|^2 for some binary assignment C
finds closest approximation of X to binary step function characterized by C. (Example: https://ljk.imag.fr/membres/Jerome.Malick/Talks/08-louvain.pdf)
11.logs for 2000 variables and alpha=1 have been committed to testlogs/ which show usual MAXSAT approximation
ratio convergence to >= 97%

---

570. (THEORY and FEATURE) Computational Geometric Factorization, Ray Shooting Query Oracles, ABC Conjecture
- related to 486, 527, 550, 558 and all other sections on factorization - 6 July 2018

ABC Conjecture implies there exist finite number of coprime integer triples a,b,c as solutions to diophantine
equation c=a+b constrained by:

$$\text{quality} = \frac{\log(c)}{\log(\text{rad}(abc))} < 1 + \epsilon \text{ for all } \epsilon > 0$$

radical(N) or rad(N) is the product of distinct prime factors of N (square-free).

Gap between numbers c and a having same prime factors has been conjectured in reference 570.1 as:
c - a > C(epsilon)a^(0.5-epsilon) which is derived from ABC Conjecture. This is contrasted against the observed
convergence of merit difference in Hardy-Ramanujan-PrimeNumberTheorem ray shooting query oracle:

$$l*pf(n+1)/\log(pf(n+1)) - l*pf(n)/\log(pf(n)) = l*N/(\log N * k*\log\log N)$$

which is the approximate gap between merit of n-th and (n+1)st prime factors pf(n) and pf(n+1) of an integer N.
An example factorization and ray shooting query log has been committed which shows actual and ray shooting estimates of factors in 4 of 5 algorithms (Hardy-Ramanujan, Hardy-Ramanujan-PNT, Cramer, Baker-Harman-Pintz) which are in close agreement (off by small distance for most factors) and Hardy-Ramanujan-PNT estimate converges. If c=N1 and a=N2 are two integers having same prime factors, previous Hardy-Ramanujan-PNT ray shooting estimate of gap between prime factors for N1 and N2 is:

l1*pf(n+1)/log(pf(n+1)) - l1*pf(n)/log(pf(n)) = l1*N1/(logN1 * k1*loglogN1)
l2*pf(n+1)/log(pf(n+1)) - l2*pf(n)/log(pf(n)) = l2*N2/(logN2 * k2*loglogN2)
rewriting N1 and N2 in terms of merit difference of prime factors:
(pf(n+1)/log(pf(n+1)) - pf(n)/log(pf(n)))*logN1*k1*loglogN1 = N1 = pf1^s1 * pf2^s2 * ... * pfn^sn
(pf(n+1)/log(pf(n+1)) - pf(n)/log(pf(n)))*logN2*k2*loglogN2 = N2 = pf1^t1 * pf2^t2 * ... * pfn^tn
=> N1 - N2 = c - a > C(epsilon)*N2^(0.5-epsilon) if ABC conjecture is true
=> (pf(n+1)/log(pf(n+1)) - pf(n)/log(pf(n)))*(logN1*k1*loglogN1 - logN2*k2*loglogN2) = N1 - N2 > C(epsilon)*N2^(0.5-epsilon)
=> (pf(n+1)/log(pf(n+1)) - pf(n)/log(pf(n))) > C(epsilon)*N2^(0.5-epsilon)/(logN1*k1*loglogN1 - logN2*k2*loglogN2)
which is the approximate lower bound on number of primes between two prime factors of N1 and N2 having same prime factors if ABC conjecture is true.

References:

570.1 Gaps between integers having same prime factors - [Cochrane-Dressler] - http://www.ams.org/journals/mcom/1999-68-225/S0025-5718-99-01024-8/S0025-5718-99-01024-8.pdf - Conjecture 2 and Theorem 1 - "...Conjecture 2. For any epsilon > 0 there exists a constant C(epsilon) such that if a<c are positive integers having the same prime factors, then c − a ≥ C(epsilon)(a^(0.5-epsilon))..." and "...Theorem 1. The abc conjecture implies Conjecture 2...."
570.2 Pillai's Conjecture and ABC Conjecture - [Subbayya Sivasankaranarayana Pillai] - https://webusers.imj-prg.fr/~michel.waldschmidt/articles/pdf/abcEnVI.pdf - Difference of consecutive perfect powers tends to infinity - Pillai’s conjecture as a consequence of the abc Conjecture : if x^p != y^q, then |x^p - y^q| >= c(e)*max{x^p , y^q}^(k-e) with k = 1 - 1/p - 1/q· Factoring Perfect powers is special case of factorization and in previous ray shooting algorithm N1=x^p and N2=x^q and both N1 and N2 have same prime factors.

---
571. (FEATURE) Streaming Facebook Analytics - Exhaustively print all wall posts of a user - 6 July 2018
---
1. Loop has been added to Streaming_FacebookData.py for printing all pages of user wall posts in json
2. logs for this have been committed to testlogs/Streaming_FacebookData.log.6July2018

---
572. (THEORY) Merit Versus Fame, an example definition of Intrinsic Fitness/Merit in social networks, some contradictions - 9 July 2018

Intrinsic Merit-Versus-Fame/Image discrepancy is a well-studied problem in social networks and is the leitmotif of Intrinsic/Absolute Choice versus Majority voting theoretical complexities discussed at length in many sections of this draft. The populist engineering facet of this includes various definitions of intrinsic merit and fame in different disciplines(sports,science etc.,). Bianconi-Barabasi Bose-Einstein condensation phenomenon in social networks occurs when links (particles) accrue in some prominent fittest hub vertices (least energy levels). Defining what least energy implies in social networks is still an open problem. An example definition of Intrinsic Merit function for Bianconi-Barabasi least energy social network profile is described:

Most link-attractive social profiles fall into three disjoint categories:

- Wealthy: Affluent/Businesses/Political (W),
- Educative: Oratory/Academic/Science/Research (E),
- Valorous: Media/Entertainment/Sportspersons/Heroics (V)

Log-Normal Fitness Attachment (LNFA) Function of previous three, f(W,E,V) = g(W)*h(E)*l(V) formed by multiplication of individual merits in three categories. When a social profile has overlaps with 2 or all 3 of these categories, its ability to attract link increases manifold implying protean all-round nature of profile. E.g Superheroes in fiction/history were versatile and had two or all three. Log Normal distribution is a sum of these random variables and tends to Gaussian by Central Limit Theorem. It remains to define merits in individual categories:

- Wealth: Income, Gravity Model based on GDP
- Education: Publications, Citations, Degrees, Awards
- Valour: Elo Rating, IPR

Metrics like HDI and SPI already incorporate mixture of these variables.

Log Normal summation of these three quantities therefore is factor causing links to gravitate towards a vertex thereby relating to least energy as:

Energy of a Social Profile Vertex ~ 1/(logW + logE + logV)

=> When the log normal sum of merit random variables tends to maximum/infinity, energy of a vertex is at its nadir. On a side note, these three quantities are interdependent:

- Education causes Wealth i.e W is a function of E mostly though there are exceptions
- Valour causes Wealth i.e W is a function of V mostly though there are exceptions

In references below, it is surmised that Fame increases linearly or exponentially with merit. The two contrarian cases when 1) Fame occurs with no merit 2) Merit does not cause Fame, complement this educated guess and is equivalent to group decision correctness probability (p) defined by Condorcet Jury Theorem and Margulis-Russo Threshold when majority function has a sharp transition at p=0.5. For majority to decide correctly (or) for Fame to coincide with merit p has to be atleast 0.5. Margulis-Russo Formula equates derivative of expected value of boolean function for bias p to the Ratio of Sum of Influences of individual variables (voters) and Standard Deviation. In simple terms, this theoretical bound implies Fame or Majority Voting opinion depends on how influential the voters are (i.e how flipping individual preference affects group decision outcome). For majority voting function, this first derivative of Pr[MajorityDecision(N) == Good] or influence is O(sqrt(number_of_voters)) at p=0.5. Both of the previous contrary scenarios occur when bias p < 0.5 which is obvious because the group is prejudiced because all its constituents are prejudiced. Therefore probability of this bias being less than 0.5 has to be quantified:

Probability of per voter bias p < 0.5 for all N voters = (0.5)^N because random variable for p has two values(p >= 0.5 and p < 0.5 both equally probable) and voting is uncorrelated which is exponentially meagre though not impossible. This implies previous Fame-Merit mismatch error occurs with (0.5)^N probability and is infact a converse of Margulis-Russo.

References:

572.1 The quantitative measure and statistical distribution of fame - [Ramirez-Hagen] - https://arxiv.org/pdf/1804.09196.pdf - Fame is measured by Google Hits, Google News references, Wikipedia edits, Wikipedia views.

572.2 How Famous is a Scientist - Famous to those who know us - [James P. Bagrow, Hern´an D. Rozenfeld, Erik M. Bollt, and Daniel ben-Avraham] - http://people.clarkson.edu/~dbenavra/paper/118.pdf - Fame increases near-exponentially with achievement/merit i.e Fame = c(Achievement)^(epsilon). Achievement or Intrinsic merit in science is defined as number of publications.

572.3 Untangling Performance from Success - [Burcu Yucesoy and Albert-László Barabási] - http://barabasi.com/f/908.pdf - defines a least square fit of wikipedia references to a tennis player versus a function of his/her ATP rank,

Tournament value, number of matches, rank of the best opponent, Career length. - "...For example, only $▫.▫ million of Roger Federer’s $▫▫.▫ million reported ▫▫▫▫ income was from tournament prizes [▫▫], the rest came from endorsements tied to the public recognition of the athlete. Yet, Novak Djokovic, who was better ranked than Federer during ▫▫▫▫, received over $▫▫.▫ million prize money but only $▫▫ million via endorsements, about a third of Federer’s purse ..."
572.4 Chess Players' Fame Versus Their Merit - [M.V. Simkin and V.P. Roychowdhury] - https://arxiv.org/pdf/1505.00055.pdf - defines a linear and exponential fit of Fame versus Elo IPR score of a Chess player. - "...This suggests that fame grows exponentially with Elo rating rather than linearly. ..."
572.5 Only 15 Minutes? The Social Stratification of Fame in Printed Media - [Arnout van de Rijt, Eran Shor, Charles Ward, and Steven Skiena] - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.888.1629&rep=rep1&type=pdf - "... We measure a person’s fame as the number of appearances of that person’s name in newspaper records. More precisely, for every day, we count the number of distinct newspaper articles in which a name appeared ..."
572.6 Log Normal Fitness Attachment - [Shilpa Ghadgea, Timothy Killingback, Bala Sundaram and Duc A. Tran] - https://www.cs.umb.edu/~duc/www/research/publications/papers/ijpeds10.pdf - "...we consider it reasonable that in many complex networks each node will have associated to it a quantity, which represents the property of the node to attract links, and this quantity will be formed multiplicatively from a number of factors...."
572.7 p-biased boolean functions analysis, Margulis-Russo Formula - http://www.cs.tau.ac.il/~amnon/Classes/2016-PRG/Analysis-Of-Boolean-Functions.pdf - Section 8.4 - Equation 8.8 - Page 225 - Example 8.49 and Question 8.23 (Condorcet Jury Theorem)
572.8 Sybils,Collusions and Reputations - GRAFIT course material - https://gitlab.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt - PageRank majority function which is non-boolean peer-to-peer converged markov chain random walk voting is a generalization of Boolean Majority function Margulis-Russo Threshold. Boolean majority function is equivalent to PageRank computed for 2 candidate websites on (n+2)-vertex graph (n user websites choose between 2 candidate websites - Good(1) and Bad(0), choice is fraction of outdegree from a user website) normalized to 0 (smaller pagerank) and 1 (bigger pagerank). Margulis-Russo threshold d(Pr(Choice=Good=1))/dp on this PageRank version of Majority function is ratio of sum of influences of variables (user websites) to standard deviation which has a phase transition to Good(1) at per-voter website bias p >= 0.5. It is obvious that bias p is inversely proportional to probability of a website being a Sybil - increased bias implies user votes more correctly and probability of it being a Sybil decreases.
572.9 Cascading and Thresholds in Social Networks - Chapter 8 - Six Degrees - [Duncan J.Watts] - Cascading is the phenomenon of deluge of acceptance of new concept/innovation in Social Networks. Every social profile has a threshold above which he/she accepts an innovation spread as a meme/fad. Probability of a user acceptance/voting for A increases sharply after more than a threshold of its neighbours accept/vote for A implying Herding.
572.10 Bradley Effect and Noelle-Neumann Spiral of Silence - https://en.wikipedia.org/wiki/Spiral_of_silence - Minority opinion is silenced by Majority - "...which stipulates that individuals have a fear of isolation ... social group or the society in general might isolate, neglect, or exclude members due to the members' opinions. This fear of isolation consequently leads to remaining silent instead of voicing opinions...". Bradley effect in psephology is problem of opinion/exit polls going wrong because of this spiral of silence.
572.11 Achievement of a Nobel Laureate defined in terms of Fame - https://www.tandfonline.com/doi/full/10.1080/13504851.2016.1200176
572.12 Fame Versus Merit in Economic Networks - CPI(Corruption Perception Index)/Global Integrity Versus Real Gross Domestic Product - https://arxiv.org/pdf/0705.0161.pdf, https://arxiv.org/pdf/0710.1995.pdf -

CPI(Transparency International) which ranks perceived corruption (not absolute) in countries based on various surveys (WEF etc.,) is the counterpart of Fame/Reputation ranking for countries similar to WWW/Social Networks. GDP has been accepted as intrinsic standard to measure wealth of a nation. These articles describe a power law distribution between CPI and Foreign Direct Investment/GDP.
572.13 Methodology of TI Corruption Perception Index - https://www.stt.lt/documents/soc_tyrimai/KSI_methodology_2007.pdf - Density Function for each Normalized Variable - Matching Percentiles and Beta Transformation - Samples, Perception and Reality - an example of Fame/Infamy correlating to Merit/Demerit.
572.14 Strength of Weak Ties - [Mark Granovetter] - https://www.cs.cmu.edu/~jure/pub/papers/granovetter73ties.pdf - Social Networks have clusters of strong ties among themseleves (Cliques) and Strength of weak tie between individuals in two separate groups determines diffusion of influence.
572.15 Linked - Chapter 10 - Viruses and Fads - Viruses (Biological and Computational), Pandemics, Epidemics, Memes/Fads, Scale Free Topology (few nodes have high degree - number_of_nodes(degree x) is proportional to 1/((constant)^x))

---
573. (FEATURE) Streaming Facebook Analytics - Comments - 10 July 2018

For each wall post, comments object is retrieved by requests and printed

---
574. (FEATURE) DeepLearning - Reinforcement Learning - Recommender Systems - Recursive Lambda Function Growth Merit - 12 July 2018

1.RecommenderSystems/Create_RecommenderSystems_Hypergraph.py has been changed similar to ThoughtNet/ to rank the
hypergraph edges by both Sentiment and Graph Tensor Neuron Network Intrinsic Merit
2.RecommenderSystems/RecommenderSystems_Hypergraph_Generated.shoppingcart3.txt has been recreated and rankings for hyperedges through hypergraph stack vertices is different from sentiment scoring
3.DeepLearning_ReinforcementLearningRecommenderSystems.py has been changed to remove creamy layer computation for
classes and all classes are lookedup in hypergraph
4.Logs for these have been committed to RecommenderSystems/testlogs/Create_RecommenderSystems_Hypergraph.GraphTensorNeuronNetworkMerit.log.12July2018 and testlogs/DeepLearning_ReinforcementLearningRecommenderSystems.shoppingcart3.log.12July2018
5.Example input is a book on quantum physics and is lookedup in Recommender Systems Hypergraph and matches return lot of books on quantum mechanics and physics

---
575. (THEORY) Computational Geometric Factorization and Parallel Construction of Wavelet Trees - 13 July 2018
- related to 34, 486

Classical algorithm for Computational Geometric Factorization described earlier is based on storing the pixelatedhyperbolic tile segments in parallelly constructed Interval Trees/Segment Trees and doing stabbing query for factor point N=(p,q) or by k-mergesort of locally sorted tile segments and doing binary

search which are O(logN*logN) parallel time. Wavelet Trees are recently gaining prominence as succinct binary search tree data structure for bigdata algorithms like string search, indexing etc., Wavelet Trees are binary search trees having following traits:

(*) Prerequisite is a set of symbols (alphabets) and sequence of symbols (string)

(*) Alphabets are divided into two equal sized sets encoded as 0 and 1

(*) At root full string is encoded as 0-1 bitmap by previous set encoding for symbols

(*) Left subtree contains first half of the parent alphabets and Right subtree the second half and encoded as 0-1 by dividing into two sets again

(*) Previous subdivision of alphabets in each node is recursed encoded as 0 and 1 till single element per node is reached at leaves.

(*) Wavelet trees have two operators: rank(c,S,i) and select(c,S,j)

(*) rank(c,S,i) queries number of occurrences of symbol c in string sequence S till position i.

(*) select(c,S,j) queries position of j-th occurrence of symbol c in string sequence S.

(*) Both rank() and select() are logarithmic time operations traversing the tree from root to leaf.

Binary Search for Factor points N=(p,q) on hyperbolic tile segments can be mapped to wavelet tree:

(*) Set of all segments of length N/x(x+1) of y-axis endpoints (N/x, N/(x+1)) are concatenated into
a single String sequence in ascending order of y-axis unsorted

(*) Each element of this unsorted concatenated string is an integer some of which could be factor points

(*) Alphabets of this string are integers in the vicinity of N in the interval [N-epsilon,N+epsilon] and size of this alphabet is 2*epsilon << N

(*) select(N,S,j) queries index k of j-th occurrence of N in this concatenated string which is nothing but a factor point N=(k,N/k) and maximum of value of j = number of factors of N. This selection finds factor in logarithmic time.

(*) But previous algorithm requires Wavelet Tree construction on PRAMs or Multicores to be in NC and there have been many parallel algorithms for wavelet tree construction mentioned in references.

LevelWT parallel wavelet tree construction algorithm in 575.7 requires O(NlogS) work and O(logS*logN) depth (S is the size of the alphabet and N is size of input). For Computational Geometric Factorization by select() on concatenated unsorted hyperbolic tile segments into one huge string, Parallel Wavelet Tree construction requires O(log(Epsilon)*logN) depth and O(Nlog(Epsilon)) work which implies requiring O(N/logN) PRAM processors. This is far better polylogarithmic depth bound than parallel k-mergesort/interval tree/segment tree which require O(logN*logN) parallel time (depth) minimum. No necessity for sorting the hyperbolic segments is another advantage and Wavelets can store huge strings suitable for factoring very large integers - Each concatenated unsorted hyperbolic segments string is O(N).

References:

575.1 Parellel Construction of Wavelet Trees on Multicore Architectures - [Sepulveda · Erick Elejalde · Leo Ferres · Diego Seco] - https://arxiv.org/pdf/1610.05994.pdf

575.2 Parallel lightweight wavelet tree, suffix array and FM-Index construction - [Labeit et al] - https://people.csail.mit.edu/jshun/JDA2017.pdf - Parallel Cilkplus Implementation : https://github.com/jlabeit/wavelet-suffix-fm-index

575.3 Wavelet Trees for All - [Navarro] - https://www.dcc.uchile.cl/~gnavarro/ps/cpm12.pdf

575.4 Improved Parallel Construction of Wavelet Trees and Rank/Select - [Julian Shun] - https://arxiv.org/pdf/1610.03524.pdf

575.5 Wavelet Trees and RRR Succinct Data Structure for O(1) rank() - http://alexbowe.com/wavelet-trees/ - Each node in Wavelet Tree invokes RRR for

O(1) rank computation per node
575.6 RRR - Succinct Indexable Dictionaries with Applications to Encoding k-ary Trees, Prefix Sums and Multisets - [Rajeev Raman, Venkatesh Raman, S. Srinivasa Rao] - https://arxiv.org/pdf/0705.0552.pdf - Succinct Data Structure for computing rank() and select() of a bitmap in O(1) time. Bitmap is divided into blocks and sets of blocks(superblocks). For each block, rank is computed and stored in a class-offset two level lookup dictionary where class is the rank of the block and offset is index into set of permutations for that block. Superblocks add the ranks of the blocks.
575.7 Shared Memory Parallelism Can Be Simple, Fast and Scalable - [Julian Shun] - http://reports-archive.adm.cs.cmu.edu/anon/2015/CMU-CS-15-108.pdf - Theorem 34 - Section 14.4.1 and Page 10 footnote 6 on NC - "...Polylogarithmic-depth algorithms are also desirable for computational complexity reasons, as they fall in the class NC (Nick's Class) containing problems that can be solved on circuits with polylogarithmic depth and polynomial size [15] ..."
575.8 Wavelet Tree in Computational Geometry - [Meng He] - Planar Point Location - https://pdfs.semanticscholar.org/f32b/bece8f1cd11e5c50acab532b41aaac34f8e0.pdf - "...Mäkinen et al. [54] showed that a wavelet tree can be used to support orthogonal range counting and reporting for n points on an n × n grid, when the points have distinct x-coordinates (this restriction can be removed through a reduction [14]). To construct a wavelet tree for such a point set, we conceptually treat it as a string S of length n over alphabet [n], in which S[i] stores the y-coordinate of the point whose x-coordinate is i. Thus Figure 1 can also be considered as a wavelet tree constructed for 8 points on an 8 by 8 grid, and this point set is given in the caption of the figure ..." - "... Fig. 1. A wavelet tree constructed for the string 35841484 over an alphabet of size 8.This is also a wavelet tree constructed for the following point set on an 8 by 8 grid:{1, 3}, {2, 5}, {3, 8}, {4, 4}, {5, 1}, {6, 4}, {7, 8}, {8, 4}..." - Previous Computational Geometric Factorization algorithm exactly applies this and construes set of points of hyperbolic arc on 2 dimensional plane as single huge string.
575.9 Wavelet Tree for Computational Geometric Planar Range Search in 2 dimensions - https://www.researchgate.net/profile/Christos_Makris2/publication/266871959_Wavelet_trees_A_survey/links/5729c5f708ae057b0a05a885/Wavelet-trees-A-survey.pdf?origin=publication_detail - "... Therefore, consider a set of points in the xy-plane with the x- and y-coordinates taking values in {1, …,n}; assume without loss of generality that no two points share the same x- and y-coordinates, and that each value in {1,…,n} appears as a x- and y- coordinate. We need a data structure in order to count the points that are in a range [lx, rx] × [by, uy] in time O(logn), and permits the retrieval of each of these points in O(logn) time. ... this structure is essentially equivalent to the wavelet tree. The structure is a perfect binary tree, according to the x–coordinates of the search points, with each node of the tree storing its corresponding set of points ordered according to the y-coordinate. In this way the tree mimics the distinct phases of a mergesort procedure that sorts the points according to the y-coordinate, assuming that the initial order was given by the x-coordinate. ..." - In the PRAM k-mergesort version of computational geometric factorization (http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.tex/download) 2 dimensional plane which has the hyperbolic curve is divided into logN rectangles of length N/logN and hyperbolic point-sets in each of these rectangles are sorted and searched for factor points. Wavelet tree exactly applies to this rectangular range search and can retrieve a factor point from hyperbolic arc point set in O(logN) time per rectangle. For logN rectangles, time to find factors is O(logN*logN). As mentioned in the reference, Wavelet Tree mimics sorting the points. But this 2-dimensional Wavelet tree requires parallel multicore construction NC algorithms mentioned earlier for logarithmic depth.
575.10 Wavelet Tree in Computational Geometry - Range Report - [Travis Gagie, Gonzalo Navarro, Simon J. Puglisi] - https://ac.els-cdn.com/S0304397511009625/1-s2.0-S0304397511009625-main.pdf?_tid=bb9ebcde-9e40-41b4-8327-4b9ed4e2fb57&acdnat=1536661119_222f9c6a5423840eec8e09f8107d3f71 - report points in a rectangular area in logarithmic time. As mentioned in previous reference,

point in each rectangle bounding a pixelated hyperbolic point-set can be retrieved in logarithmic time e.g query factor point N=(p,q) on the point-set.
575.11 Functional Approach to Data Structures and their use in Multi-dimensional Searching - [Bernard Chazelle] -
https://www.cs.princeton.edu/~chazelle/pubs/FunctionalDataStructures.pdf - Data structure which inspired wavelet
tree in geometric search - M-Structures (Mergesort-Structures)
575.12 Space Efficient Data Analysis Queries on Grids - [Gonzalo Navarro, Yakov Nekrich, Luís M.S. Russo] - https://ac.els-cdn.com/S0304397512010742/1-s2.0-S0304397512010742-main.pdf?_tid=fa66c1a6-518d-4fd1-b6f3-59720d2736fa&acdnat=1537341599_ebf10ba8401e952f49aa06f5168192ba - Space efficient representation of point-sets on 2-dimensional grid
575.13 Succinct Orthogonal Range Search Structures on a Grid with Applications to Text Indexing - [Prosenjit Bose, Meng He, Anil Maheshwari, and Pat Morin] - https://pdfs.semanticscholar.org/3d28/169f7c1524cf87c9215b61d536939ae5841a.pdf - Orthogonal range reporting of points in a rectangle - "Lemma 8. Let N be a set of points from the universe M = [1..n]×[1..n], where n = |N|. N can be represented using n lg n + o(n lg n) bits to support orthogonal range reporting in O(k lg n/ lg lg n) time, where k is the size of the output....". This implies each hyperbolic point in rectangle of width N/logN in http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarithmicSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_updateddraft.pdf/download containing pixelated hyperbolic point-sets can be reported in sub-logarithmic time (k=1) independent of other rectangles thereby obviating k-mergesort of the tile segments (or mimicking it).
575.14 Succinct Datastructure for Multidimensional Orthogonal Range Searching - [Ishiyama-Sadakane] -
https://www.computer.org/csdl/proceedings/dcc/2017/6721/00/07921922.pdf - kd-tree example for orthogonal range reporting - Figure 1.

---
576. (FEATURE) Spark Streaming - Java 1.8 - Spark Java Upgrade to 2.3 and Some Structured Streaming
Enhancements - 17 July 2018
---
1. Spark Generic Streaming Java Implementation has been compiled on Spark 2.3 upgraded from Spark 2.1 and sparkgenericstreaming.jar has been repackaged
2. JavaPairDStream code which was commented earlier has been uncommented to return Word Count instead of Words
3. Structured Streaming usecase example in https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html for website advertisement analytics has been implemented by computing timestamp of each word occurrence along with its count
4. Word.java has been changed for doing get/set of Time,Count and Word
5. DataFrame schema has been changed to Word, Count and Timestamp instead of just Word.
6. This Structured Streaming DataFrame prints periodically changing tables of word occurrences, frequencies and timestamp as a tabular column.
7. JavaPairDStream still does not have Java Pair support but only scala.Tuple2.

8. Commandline for spark-submit has been changed for jsoup:
/home/kashrinivaasan/spark-2.3.0-bin-hadoop2.7/bin/spark-submit --jars /home/kashrinivaasan/jdk1.8.0_171/lib/jsoup-1.11.3.jar --class SparkGenericStreaming --master local[2] sparkgenericstreaming.jar "https://twitter.com/search?f=tweets&vertical=news&q=Chennai&src=typd"

9. CLASSPATH for spark 2.3 has been changed as for catalyst, scala-reflect:
/home/kashrinivaasan/spark-2.3.0-bin-hadoop2.7/jars/spark-streaming_2.11-2.3.0.jar:.:/home/kashrinivaasan/spark-2.3.0-bin-hadoop2.7/jars/scala-library-2.11.8.jar:/home/kashrinivaasan/spark-2.3.0-bin-hadoop2.7/jars/spark-sql_2.11-2.3.0.jar:/home/kashrinivaasan/spark-2.3.0-bin-hadoop2.7/jars/spark-core_2.11-2.3.0.jar:/home/kashrinivaasan/jdk1.8.0_171/lib/jsoup-1.11.3.jar:/home/

kashrinivaasan/spark-2.3.0-bin-hadoop2.7/jars/spark-catalyst_2.11-2.3.0.jar:/home/kashrinivaasan/spark-2.3.0-bin-hadoop2.7/jars/scala-reflect-2.11.8.jar:/home/kashrinivaasan/jdk1.8.0_171/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

10. Example logs for streamed news analysis has been committed to testlogs/

---

577. (THEORY and FEATURE) Computational Geometric Factorization and its Pell Diophantine (Complement Diophantine of Prime Valued Polynomials) - Updates - 18 July 2018

---

1. DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been updated to persist the factors
found to a JSON file (json.dump() and json.load()).
2. Aggregating factors found by multiple Spark executors requires defining a new Spark Broadcast Accumulator
class which appends a factor to already existing accumulator value globally
3. complement.py has been updated for solving both directions of Pell-Factoring (for D=1 and N > 1):
	3.1 Pell Equation to Factoring: Solves Pell equation x^2-y^2=N for D=1 and finds factors (x+y) and (x-y) in exponential time (or between polynomial and exponential - http://www.math.leidenuniv.nl/~psh/ANTproc/01lenstra.pdf)
	3.2 Factoring to Pell Equation: Computes all factors (p,q) of N by PRAM Computational Geometric Factorization and finds x and y in Pell Equation as: x=(p+q)/2 and y=(p-q)/2. This solves a special case of Pell Equation in classical NC for D=1 (for generic, only quantum polynomial time algorithms are known so far)
4. Logs for these have been committed to testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.JSONPersistence.18July2018 and
testlogs/complement.PellEquation.SolvedByPRAMNCFactoring.18July2018
5. Ray Shooting Oracle shows some exact and close ray shooting of factors. For simple Hardy-Ramanujan ray shooting factorization requires only O(loglogN) + O(log(N/loglogN)*log(N/loglogN)) + O(log(N/loglogN)) time to find a factor because prime factors are O(loglogN) and distance between any two prime factors is O(N/loglogN).
6. Hyperbolic arc bow for this x-axis distance is sufficient to locate a factor. K-mergesort/Segment-tree of this subset is O(log(N/loglogN)*log(N/loglogN)) and binary search is O(log(N/loglogN)).

---

578. (FEATURE) ConceptNet 5.6 Client Update - Common Ancestor Algorithm rewrite for path between concepts
- 19 July 2018

---

1. ConceptNet 5.6 provides a REST endpoint /query for finding connections between two concepts which has been
wrapped by a new function conceptnet_path()
2. Common Ancestor Algorithm for finding distance betwen two concepts has been rewritten to create a NetworkX
subgraph from the edges connecting two concepts. Function conceptnet_distance() has been renamed to conceptnet_least_common_ancestor_distance()
3. Loop iterating over cartesian product has been split into 2 loops which grow from two opposite directions
(concept1, ---) and (--- ,concept2) and intersect at common ancestors.
4. Edges recursively seen by /related REST query are collected in a list and a NetworkX DiGraph is created from
it which has all paths between concept1 and concept2
5. Shortest path on this subgraph yields the shortest distance and path between

concept1 and concept2 and thus passes via the least common ancestor between two concepts
6. Logs for this have been captured in testlogs/ConceptNet5Client.log.19July2018 which show conceptnet_least_common_ancestor_distance() computes deeper paths than /query

---

579. (THEORY and FEATURE) SAT Solver Update - some refactoring, and replace=True - 20 July 2018

1.CNFSATSolver.py has been refactored for invoking Uniform and Non-uniform choice based on Variables==Clauses
or Variables != Clauses and lsmr() invocation has been changed to remove regularized LSQR
2.replace=False has been set to replace=True in np.random.choice() for Uniform choice of random 3SAT variables
(and everywhere else). For very long time replace was set to False
3.logs for this have been committed to testlogs/CNFSATSolver.2000VariablesAlpha1.0.log.20July2018 and
MAXSAT approximation ratio for few iterations ranges from ~95% to ~99% (logs for 98.25% have not been committed)

---

580. (THEORY) Random Matrix Least Square SAT Solver, kSAT Sharp Threshold and Friedgut Theorem, ThoughtNet
Hypergraph as k-SAT problem, TextGraph and Social Network Property Testing - 26 July 2018 - related to 453

Margulis-Russo Threshold and Condorcet Jury Theorem in the context of Sharp Threshold of Majority Function have
been extensively described earlier. Random Matrix Least Squares SAT Solver has been analyzed and distribution for
choosing a literal has been derived as 1/sqrt(m*n). Similar to Majority Function Sharp Thresholds exist for
kSAT formulae from Friedgut's Theorem which is stated as:

The satisfiability property for random k-CNF has a sharp threshold, i.e. there exists p' such that for any epsilon > 0,
μp(A) → (1 if p ≤ (1 − epsilon)p', 0 if p ≥ (1 + epsilon)p')
where p' = alpha(n)*n/N.

probability of choosing a clause p is (M=Number of clauses in a random kSAT)/(N=All possible clauses in a random kSAT). Number of all possible clauses in a random kSAT = N = nCk*2^k and alpha = M/n (n is number of variables) => p = (alpha*n)/N.

Probability of choosing a random 3SAT clause in Random Matrix SAT Solver p = [1/sqrt(mn)]^3 = 1/((mn)^1.5). From Friedgut's Theorem number of satisfied clauses increase if p <= (1-epsilon)p'. This leads to inequality:

1/(mn)^1.5 <= ((1-epsilon)alpha(n)*n)/N
=> 1/(mn)^1.5 <= [((1-epsilon)alpha(n)*n)*6*(n-3)!]/(8*n!) from N = nC3*2^3
Simplifying reduces to:
1/(mn)^1.5 <= [((1-epsilon)alpha(n))*0.75*]/((n-1)(n-2)) from N = nC3*2^3

But mn=alpha(n)*n^2 substituting which reduces the previous inequality to:

[(n-1)(n-2)/(n^3*(1-epsilon)*0.75)]^0.4 <= alpha(n)

Example - For n=10 and alpha(n)=4.267 previous inequality becomes:

0.39165/(1-epsilon)^0.4 <= alpha(n)
=> epsilon <= 0.6154

implying a sharp thresold to increased number of satisfied clauses.

Related note: Sharp Thresholds in k-SAT have some applications in graph representation of texts e.g Definition graphs, ThoughtNet Hypergraph. Any random k-SAT formula can be represented as a hypergraph (Factor graph) of variables as vertices and clauses as hyperedges. ThoughtNet hypergraph is a Factor graph for some k-SAT and extracting the k-SAT from ThoughtNet translates the Contextual Multi Armed Bandit ThoughtNet to a random kSAT instance. Graph representation of texts can be queried for various properties e.g number of matchings, triangles, connected components which extract pattern in the meaning of text. For example, Margulis-Russo threshold is a graph property threshold of atleast n(n-1)/2 random edges between n vertices in a random graph where each edge is created with probability p. Similar threshold holds for Social Network Random graphs.

References:

580.1 Thresholds in Random Graphs and k-SAT - [Emmanuel Abbe] - https://pdfs.semanticscholar.org/d862/14a9e02c3749ca92515b592f2fdb38710b45.pdf - Proof of Margulis-Russo Threshold (Lemma 2), Sharp Threshold of k-SAT and Friedgut Theorem (Section 6 - Theorem 7)
580.2 Hunting for Sharp Thresholds - [Ehud Friedgut] - http://www.ma.huji.ac.il/~ehudf/docs/survey.ps - "...Much like water that freezes abruptly as its temperature drops below zero the structure of the random graph quickly changes from a primordial soup of small tree-like connected components to a graph in which the giant component, one of linear size, captures a constant proportion of the vertices...." - Erdos-Renyi edge probability threshold in random graphs has close resemblance to Bose-Einstein condensation in networks.

---
581. (THEORY) Rice Theorem, Complement Diophantines, Program Analysis, SAT Solver and Undecidability of
Non-trivial properties of Turing Machines - 30 July 2018 - related to 166, 471

Draft updates to:
---
1. Decidability of Complement Functions (2011) - Section on Rice Theorem - https://arxiv.org/abs/1106.4102v1

Rice Theorem implies every non-trivial property of Turing Machines are undecidable. Non-trivial property is defined as below.

For a non-trivial subset S of recursively enumerable languages which is a non-empty proper subset of all recursively enumerable languages RE(trivial subsets are empty-set and RE):
581.1 Turing Machine M which accepts a language in S
581.2 Turing Machine M' which accepts a language in complement of S (RE \ S)

Previous proofs of undecidability of complement diophantines by Post Correspondence Problem and MRDP theorem are devoted only to the question of existence of a complement function. Since all languages in RE are diophantine, languages which are recursively enumerable but not recursive do not have accept/reject halting Turing Machine to decide diophantine polynomial representation. But Rice Theorem applies in a different setting: Once the set S and its complement RE \ S are known, it is undecidable if an arbitrary Turing Machine M accepts a language in S or RE \ S. In terms of equivalence of RE and Diophantine Equations, it is undecidable if an arbitrary Diophantine Equation represents a set in S or its complement in RE \ S i.e. Given a (complement) diophantine D and 2 complementary sets in RE (disjoint set cover of RE) it is undecidable to decide which set in S or RE \ S are values of diophantine D which is the converse direction of existence of complement diophantine.

PhD Thesis Proposal (2011) - Sections on Program Comprehension - https://sites.google.com/site/kuja27/PhDThesisProposal.pdf - Theoretical Answers to some questions in it:

For the same reason, Comprehending a Program behaviour is a non-trivial property X which asks "Does the Code do X?". Translating Code to Boolean Formula and applying a SAT Solver (as done in SATURN Program Analyzer) is also a non-trivial property: "Does the Code Satisfy the 3SAT?". Similarly automated debug analytics of code is undecidable too. Software solutions to these are only approximations.

A circular timerwheel hashmap of dynamic sets of timeout clockticks to queues of processes - Survival Index Timeout - has been applied to OS Scheduler in https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt which estimates a non-trivial undecidable property approximately - Worst Case Execution Time of a process.

Licensing Violation is a great problem in Open Source Software (advertently or inadvertently) and Software Similarity Detection provides solutions to identify similar source codes. Mostly available Similarity Detectors parse function and variable names in two source trees and compare similarity but rarely compare the control flow graphs.But Frequent Subgraph Mining of SATURN Control Flow Graphs (static) and Valgrind/Callgrind/KCachegrind Call Graphs (dynamic) in NeuronRain AsFer takes it further solving similarity detection holistically. Third-party Graph Isomorphism Detectors can also be applied on these compiletime and runtime graphs. NeuronRain AsFer also provides indexing of source code and pairwise similarity by Latent Semantic Analysis.

References:

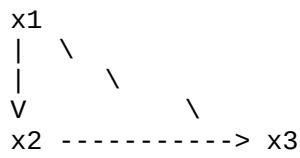
581.3 Rice Theorem applications in Programming - http://ai.cs.unibas.ch/_files/teaching/fs16/theo/slides/theory-d08.pdf - Every question on comprehending Programs is undecidable
581.4 Mortality Problem/Halting Problem Undecidability Proof of Rice Theorem - http://people.seas.harvard.edu/~cs125/fall14/lec17.pdf
581.5 Approximate Program Analysis - https://cs.au.dk/~amoeller/spa/1%20-%20TIP.pdf - Exact Program Analysis is undecidable
581.6 OSSPolice - Identifying Open-Source License Violation and 1-day Security Risk at Large Scale - [Ruian Duan, Ashish Bijlani, Meng Xu, Taesoo Kim, Wenke Lee] - https://acmccs.github.io/papers/p2169-duanA.pdf
581.7 OSSPolice - https://github.com/osssanitizer/osspolice
581.8 Bliss - Graph Isomorphism Detector - http://www.tcs.hut.fi/Software/bliss/
581.9 Program Analysis - Analysis of Control Statements - expected and variance of execution times and laplace transforms - Appendix E - Probability and Statistics, Queueing, Reliability and Computer Science - [Kishore Shridharbhai Trivedi, Duke University]
581.10 Program Analysis - JRF self-study - 2008 Course Notes - [Madhavan Mukund, CMI and Sriram Rajamani, Microsoft Research] - Pointer Analysis - Andersen and Steensgaard algorithms creating points-to graphs - https://www.cs.cmu.edu/~aldrich/courses/15-819O-13sp/resources/pointer.pdf - Graph Mining algorithms can be applied to points-to graphs (static)
581.11 Comparing Call Graphs - [Lohtak] - https://plg.uwaterloo.ca/~olhotak/pubs/paste07.pdf
581.12 Principles of Program Analysis - [Flemming Nielsen-Hanne Riis Nielsen-Chris Hankin] - https://books.google.co.in/books?id=YseqCAAAQBAJ&printsec=frontcover&dq=nielsen+nielsen+hankin&hl=en&sa=X&ved=0ahUKEwiki_-gsOXfAhUPeysKHWpmA1gQ6AEIKjAA#v=onepage&q&f=false - Interprocedural Analysis - Section 2.5.4 Call Strings as Context and Examples 2.33 and 2.37 - function call stacks or paths in dynamic call graphs can be formalized as unbounded call strings which are defined by concatenation of values of transfer functions for each entry and exit(return) of functions invoked along a path in call graph.

581.13 Survey of Tools for estimation of Worst Case Execution Time of a program - Fig. 5 Bound Calculation - CFG, Path based, IPET, Transformation rules, Structure based - http://moss.csc.ncsu.edu/~mueller/ftp/pub/mueller/papers/1257.pdf

---

582. (THEORY) Cloud, Autonomous Vehicles, EventNet Logical Clock, Relativity, Lorentz Transform, Lamport's Clock
 - related to 70-86 - 2 August 2018, 3 August 2018

Cloud of autonomus vehicles(e.g drones) in a wireless network are mostly not stationary and each node has a velocity. This relative motion between nodes of a cloud fits in the realm of relativistic time dilation. Lorentz Transform for a stationary and a moving observer with respect to a frame of reference is derived as below:



Node x1 in vehicle cloud transmits an information with speed of light c (which is fixed for all observers by Maxwell Equations as 3 * 10^5 km/s) as electromagnetic signal, which is received by both x2 and x3. x2 is stationary and x3 is moving with velocity v. Each node has its own clock t(xi). For x2, information from x1 reaches in t(x2)*c and for x3 information from x1 reaches in t(x3)*c. By this time x3 has travelled t(x3)*v distance from x2. This leads to Lorentz transform below:

```
(t(x2)*c)^2 + (t(x3)*v)^2 = (t(x3)*c)^2
(t(x2)*c)^2 = (t(x3)*c)^2 - (t(x3)*v)^2
=> (t(x2)*c)^2 = t(x3)^2*(c^2 - v^2)
=> t(x2)^2 = t(x3)^2*(1 - (v/c)^2)
=> t(x2) = t(x3)*sqrt(1 - (v/c)^2)
```

i.e clock of vehicle x3 slows down compared to x2's clock as x3 moves faster.
Though autonomus vehicles could have speed less than light, if event timestamps are in nanoseconds (C++ provides nanosecond resolution in std::chrono::duration_cast<std::chrono::nanoseconds>), relativistic time dilation could be non-trivial. Lamport's Logical Clock has the causation protocol (s,r):

sender(s) - transmits timestamp to receiver
receiver(r) - finds maximum of local timestamp and received timestamp and increments by 1

which is valid only for stationary nodes in cloud and has to be augmented by previous relativistic time dilation transform for moving nodes in cloud.

Previous time dilation is because of Special Relativity where as General Relativity predicts clocks in objects closer to a heavier mass tick slower. This causes onboard clocks in GPS satellites to tick faster than those on earth slowed down by its mass and corrections are applied. Drones using GPS navigation are directly affected by this. EventNet is a partial ordered directed acyclic graph (infinite) and causality in EventNet Logical Clock (for wireless clouds) is defined by light cone i.e an expanding circular region travelled by light plotted against arrow of time - two events P and Q are causally related if Q is in past lightcone or future lightcone of P and viceversa.

References:

582.1 On the Relativistic Structure of Logical Time in Distributed Systems - https://www.vs.inf.ethz.ch/publ/papers/relativistic_time.pdf - Section 5 - Minkowski Spacetime of (n-1) space and 1 time dimensions and Light Cone

Causality - Time is a partial order and not linear - Causal ordering between two spacetime points U and V
582.2 Atomic Clocks, Satellite GPS and Special/General Relativity - http://www.astronomy.ohio-state.edu/~pogge/Ast162/Unit5/gps.html
582.3 Feynman's Lectures on Physics - [Richard Feynman] - Volume 1 - Chapter 15 - Lorentz Transformation and Special Relativity
582.4 CAP Theorem,Spanner Database,TrueTime clock - [Breuer] - https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45855.pdf - Spanner achieves almost all three (Consistency, Availability, Partition Tolerance) - "...The short answer is “no” technically, but “yes” in effect and its users can and do assume CA. The purist answer is “no” because partitions can happen and in fact have happened at Google, and during (some) partitions, Spanner chooses C and forfeits A. It is technically a CP system. We explore the impact of partitions below..."

---
583. (FEATURE) *) Sentiment Analyzer - Empath word embeddings - multipolar sentiment analysis
*) Factorization - JSON - 6 August 2018

1.SentimentAnalyzer.py has been changed to invoke Empath lexical analyzer for various signals (sentiments)
in Markov Random Fields Belief Propagation Sentiment Analysis function.
2.Empath invocation computes objective maximum score from the dictionary returned by lexical analyzer for
each vertex in a clique
3.RecursiveLambdaFunctionGrowth.py invokes Markov Random Fields Belief Propagation Sentiment Analyzer function
for the text graph and returns the sentiment score in intrinsic_merit_dict["empath_sentiment"]
4.Logs for this have been committed to testlogs/RecursiveLambdaFunctionGrowth.log.Empath_BeliefPropagation_RGO_M

5.JSON for a sample factorization execution has also been committed

References:

583.1. Analyzing Right Wing YouTube Channels - [Raphael Ottoni,Evandro Cunha,Gabriel Magno,Pedro Bernardina,
Wagner Meira Jr.,Virgilio Almeida,Berkman Klein] - https://arxiv.org/pdf/1804.04096.pdf
583.2. Empath - http://empath.stanford.edu/

---
584. (FEATURE) Latent Semantic Analysis/Indexing (LSA/LSI) Implementation - 7 August 2018
---
1. Latent Semantic Analysis/Indexing has been implemented as python class in LatentSemanticAnalysis.py
2. This reads list of filenames from LatentSemanticAnalysis.txt and converts into term-document matrix
by invoking Pandas DataFrame and ScikitLearn Vector libraries
3. Pandas DataFrame is converted to NumPy array and Singular Value Decomposition (SVD) is done on
term-document matrix to arrive at UEV^T decomposition.
4. Documents are represented by U and terms by V. E is matrix of singular values.
5. Cosine similarity of pairs of documents in U is computed for all pairs of documents and printed in log:
testlogs/LatentSemanticAnalysis.log.7August2018

6. This class is multipurpose and can be used for variety of applications including Program Analysis.
7. Corpus for this execution is list of python files and similarity between python code is evidenced by
cosine similarity

---
585. (FEATURE) SAT Solver - CVXOPT LAPACK gels() least squares - 9 August 2018
---
1. SAT Solver has been changed to invoke LAPACK gels() least squares from CVXOPT
2. For this NumPy array is cast to CVXOPT real matrix object
3. CVXOPT gels() works only if the matrix is full-rank (number of linearly independent rows is equal to number of rows and similarly for columns) and throws ArithmeticError exception otherwise
4. Approximately 1040 random 3SAT instances for 10 variables - 10 clauses have been executed and MAXSAT
approximation ratio converges to ~97%. For larger values of variables, finding full rank matrix needs lot of
iterations but still for 20 variables - 20 clauses similar convergence to ~96% was observed.
5. Support for CVXOPT is crucial because of wide range of functionalities (including Convex programs, SDPs) present in CVXOPT
6. logs for this have been committed to testlogs/CNFSATSolver.10variablesAlpha1.0.CVXOPT.log.9August2018

---
586. (FEATURE) Latent Semantic Analysis - Low Rank Approximation - 9 August 2018
---
1. New function for computing low rank approximation of term-document matrix has been included
2. For SVD of td = UEV^T, few least singular values (square root of eigen values of td) are set to zero
(hardcoded to 10, because matrix_rank() computation is quite slow for large term-document matrices)
3. Low rank approximation of td is computed from matrix product of U * rankreduced(E) * V^T
4. logs for this have been committed to testlogs/LatentSemanticAnalysis.log.9August2018

---
587. (THEORY) ImageNet, EventNet and Recursive Lambda Function Growth and Merit of Large Scale Visuals
 - ImageGraph Algorithm - related to 249,410 - 10 August 2018

ImageNet is the pictorial WordNet and has received lot of attention in Computer Vision and Large Scale
Visual Recognition (LSVR) from Images/Videos. Deciding merit of a Video intrinsically with no reputation rankings
has been an open problem (in Fame versus Merit). Recursive Lambda Function Growth algorithm which generalizes
Recursive Gloss Overlap for text-graph creation has been so far restricted only to text analysis. ImageNet
has been applied for Object recognition/tracking based on Convolution Networks in Video datasets (e.g VID).
This section describes an algorithm to map a video to ImageNet graph of visuals and infer merit from this
image-graph:

ImageNet-EventNet ImageGraph Algorithm:

(*) Every Video is set of frames of Images continuously played out in time.

(*) Some image frames in the past causally relate to some frames in future in the Video.

(*) For each image frame:

{

(*) Each frame is recognized by a standard LSVR algorithm (e.g ConvNet) and objects are annotated by bounding boxes.

(*) Each bounded recognized object in a frame is lookedup in ImageNet and an ImageGraph is constructed per frame by computing ImageNet path between every pair of recognized objects and unifying the paths to create an ImageGraph for the frame, similar to WordNet Path s-t connectivity algorithm mentioned earlier for TextGraphs.

(*) Merit of Video in human judgement is more than mere connectedness of information conveyed - it also depends on visual appeal e.g object attributes like Angle, Resolution, Lighting, Texture, Aesthetics etc., Edge weights in frame ImageGraph has to reckon these factors.

}

(*) Previous loop creates a huge set of ImageGraphs for frames in Video.

(*) Following EventNet convention, every frame is an Event vertex of set of actors (partakers) in EventNet. Causation between two Event ImageGraphs for chronologically ordered frames F1 and F2 (E(F1) and E(F2)) is determined by causal similarity of the graphs E(F1) and E(F2) i.e if there exist ImageNet paths from vertices of E(F1) to vertices of E(F2) then F1 "causes" F2.

(*) Example: If frame F1 has recognized visual ImageGraph of a birthday party and F2 has recognized visual ImageGraph of cake-cutting, and if there exists an ImageNet path "birthday - celebration - cake - cutting " ,then E(F1) causes E(F2).

(*) Previous algorithm converts a Video into a Graph of Graphs (or) Tensor which is a matrix having matriix entries. Size of this video tensor is O(number_of_frames).

(*) If Audio is also annotated by a Speech Recognition algorithm to text, it catalyses in deciphering causation amongst sets of frames (by creating AudioGraph per set of frames from WordNet and connecting AudioGraphs by WordNet distances and this creates an audio tensor).

(*) Once these tensors for video and audio are created, merit is computed by how causally connected these tensors are from standard Graph complexity measures already defined for TextGraphs.

ImageGraph Merit as Tensor Products:

(*) Two causally related (having ImageNet path between their vertices) ImageGraphs E(F1) and E(F2) for frames F1 and F2 can be multiplied by Tensor Product which is a cartesian/kronecker product of two adjacency matrices of each ImageGraph per frame - resulting tensor product graph has:

(*) ImageNet vertex gh for g in V(E(F1)) and h in V(E(F2))

(*) ImageNet edge (a,b)-(c,d) iff edge (a,c) in E(F1) and (b,d) in E(F2)

(*) the tensor product for two frame ImageGraphs in the Video (which are connected causally by ImageNet) as E(F1) * E(F2) yields a semantic similarity measure between successive chronologically causative frames. e.g edges (birthday, congregation) in E(F1) and (cake, serving) in E(F2) are mapped to edge (birthday-cake, congregation-serving) in the product graph.

(*) EventNet Causality of ImageGraphs in a Video is represented by an adjacency matrix defined as:

(*) EventNet matrix EN for a video of N image frames has N rows and N columns

(*) entry EN(i,j) = tensor product of Event ImageGraph for frame i and Event ImageGraph for frame j if there is an ImageNet path between ImageGraphs for Frame i and j, else, 0

(*) This EventNet adjacency matrix represents a graph of causality between

frames in the video.

(*) Weight of every edge (a,b)-(c,d) (or) (ab)-(cd) in tensor product of ImageGraphs for frames F1 and F2, E(F1)*E(F2) (i.e (a,c) in E(F1) and (b,d) in E(F2)) = 1/(d(a,b)*d(c,d)) for ImageNet distance d between two vertices in different ImageGraphs E(F1) and E(F2).

(*) Previous Inverse Weight implies an edge has more weight in Tensor Product graph if distance between endpoints of edges in two different ImageGraphs are close enough in ImageNet. When either of the endpoints in two different ImageGraphs are not related by ImageNet, d is infinity and respective adjacency matrix entry for tensor product graph is 0.

(*) Applicability of EventNet to Large Scale Visual Recognition is quite unusual. Necessity of tensor product entries as edge weights in Video EventNet adjacency matrix, than simple scalar weights is because of necessity of capturing distance between every pair of vertices among 2 causally related ImageGraphs - one vertex is in E(F1) and other in E(F2)

Gist and Intuition of previous algorithm

Every Video (set of frames) can be mapped to 1) a causation graph of image frame event vertices and 2) each frame event itself is a subgraph of ImageNet - depth 2 graph-of-graphs creation. Causality between frame events is determined by ImageNet distances between vertices in ImageGraphs of the frames represented as tensor product.

References:

587.1 ImageNet Large Scale Visual Recognition Challenge - [Olga Russakovsky* · Jia Deng* · Hao Su · Jonathan Krause · Sanjeev Satheesh · Sean Ma · Zhiheng Huang · Andrej Karpathy · Aditya Khosla · Michael Bernstein · Alexander C. Berg · Li Fei-Fei] - https://arxiv.org/pdf/1409.0575.pdf
587.2 Zig-Zag Product - [Reingold, O.; Vadhan, S.; Wigderson, A. (2000)], "Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors", Proc. 41st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 3–13 - https://arxiv.org/abs/math/0406038 - Graph product of large graph G and small graph H in which each vertex in G is replaced by copy of H. In ImageGraph EventNet Tensor above, each vertex in EventNet G is replaced by ImageGraph of Frame Event.
587.3 Tensor Product of Graphs - https://en.wikipedia.org/wiki/Tensor_product_of_graphs
587.4 ImageNet publications - http://image-net.org/about-publication
587.5 ImageNet VID dataset - https://www.kaggle.com/c/imagenet-object-detection-from-video-challenge

---
588. (THEORY) Analyzing Youtube Videos, Ranking Movies and ImageGraph Tensor Products Algorithm for
Merit of Large Scale Visuals - 14 August 2018, 15 August 2018 - related to 587

Analyzing Videos in public domain and Ranking Movies based on merit as opposed to usual human review/reputation
based rankings is one of the hardest Large Scale Visual Recognition Challenge. Algorithm for Mapping Video or Movie to EventNet Graph of frame ImageGraphs and representing EventNet edge weights as Tensor Products has been described earlier. This algorithm just elicits the graph complexity of visual information contained in the Video overlooking sentimental polarities and psychological impact. Language Inquiry and Word Count (LIWC) is a dictionary of human voted words categorized by their human psychological emotions. Classifying the objects in Video or Movie of human viewership therefore demands emotional categories of words. Following changes are made to previous algorithm for quantifying emotions:

(*) Emotions are mostly verbal and Speech Recognition annotated AudioGraph

of Video or Movie Dialogue Transcript are created per frame/scene by union of WordNet paths for all pairs of words in Transcript per frame/scene

(*) Word Vertices of Each ImageGraph and AudioGraph per frame in Video or Movie are also annotated by the psychological polarity of the Word Vertex in LWIC

(*) Weight of every edge (a,b)-(c,d) (or) (ab)-(cd) in tensor product of AudioGraphs and ImageGraphs for frames F1 and F2, E(F1)*E(F2) (i.e (a,c) in E(F1) and (b,d) in E(F2)) is the ordered pair of inverse distance similarity and majority LIWC polarity = (1/(dist(a,b)*dist(c,d)), majority_LIWC_polarity_of(a,b,c,d)) for ImageNet distance dist between two vertices in different ImageGraphs or AudioGraphs E(F1) and E(F2).

(*) Merit of the Video or Movie in toto is the ordered pair = (Tensor Rank of the AudioGraph and ImageGraph EventNet Tensors, Sorted list of top percentile LIWC polarity of edge weights in ImageGraph and AudioGraph Tensor Products)

(*) Computing Tensor Rank is NP-Complete for Finite Fields and NP-hard for Rationals.

(*) Previous Merit captures both the complexity and emotional content of the Large Scale Visual.

(*) Tensor Rank is the generalization of Matrix Rank (number of linearly independent rows) = Dimension of Tensor = Minimum number of Tensors Ti to linearly express Tensor T = Sigma(Ti)

(*) Most algorithms for Video analysis are caption and statistics based whereas the previous algorithm is a graph theoretic formulation of a machine learning computer vision problem.

(*) There are recent other alternatives to LIWC e.g Empath.

(*) Intuition for choice of Tensor Rank as Merit of Large Scale Visuals:

(*) Tensor Rank of EventNet is the number of Independent n-rows/n-columns of Events in the Video

where n-columns and n-rows are hypercubes considering Tensor as multidimensional array

(*) Increased Tensor Rank implies the Events in the Video are more disconnected/independent

while decreased Tensor Rank implies the Events are more connected/dependent

(*) Therefore Video of Low Tensor Rank is heavily connected.

(*) Though applied to Video EventNet, tensor rank has wider philosophical ramifications because EventNet formalizes flow of time logically and has weird implication - EventNet Logical Time Tensor can be written as linear sum of independent tensors and therefore Time tensor essentially is a linear function of finer time subdivision tensors.

(*) An important difference between Videos and Theoretical EventNet is: Event vertices and participants/actors within an Event vertex can be created in parallel independent of other event vertices in theoretical eventnet (and causality edges between event vertices could be established later in parallel too) while a video (youtube, movie etc.,) is always a serialized set of frames (events) though it could depict concurrent events - video is non-parallel unless more than one video source is simultaneously streamed. EventNet can be formally defined by some Process calculi.

(*) Usually Like-ing Versus Merit of Videos has the following conflict: Videos or movies which induce viewers to repetitively watch them, garner huge popular support which is not necessarily a reflection of their merit. Music/Audio has an equivalent of this phenomenon known as "Earworm"

References:

588.1 Language Inquiry and Word Count - LIWC - http://liwc.wpengine.com/wp-content/uploads/2015/11/LIWC2015_LanguageManual.pdf

588.2 Exploring Tensor Rank - http://www.its.caltech.edu/~matilde/WeitzMa10Abstract.pdf

588.3 Tensor Rank is NP-Complete - [Hastad] - http://www.nada.kth.se/~johanh/tensorrank.pdf

588.4 Computational Complexity of Tensor Problems - https://www.stat.uchicago.edu/~lekheng/meetings/agtd/hillar13.pdf

588.5 Tensor Rank as Measure of its Complexity - http://web.math.unifi.it/users/

ottavian/colloquium_kias.pdf
588.6 Process Calculi - https://en.wikipedia.org/wiki/Process_calculus
588.7 Actor Model and Process Calculi -
https://en.wikipedia.org/wiki/Actor_model_and_process_calculi_history

---
589. (THEORY and FEATURE) SAT Solver - CVXOPT - gesv() least squares - 16 August 2018 - related to 424
---
1. CNFSATSolver.py has been changed to import gesv, getrs, sysv functions for least squares in cvxopt
2. Each of these were tried for 10 variables and 10 clauses combinations and gesv() has the highest accuracy
observed so far (~99% to 100%) in any of the least squares algorithms.
3. logs for 500 random 3SAT instances have been committed to testlogs/CNFSATSolver.10variablesAlpha1.0.CVXOPT.log.16August2018
4. Caveat: Though the number of variables/clauses are less which could cause repetitive choice of same 3SAT similar randomness did not attain 99% accuracy for other least squares algorithms earlier for similar variable-clause combination.
5. gesv() also throws ArithmeticError for non full-rank matrices and logs for gesv() are filtered-out random 3SAT instances which are full rank. This leaves some random 3SAT instances unsolved.
6. try...except has been changed to print exception stacktrace for ArithmeticError

CVXOPT gesv() has breached the accuracy barrier to large extent from 95-97% to > 99% coinciding with Random Matrix Analysis for 3SAT. But problem is with high number of variables and clauses which do not churn out full rank matrices for 3SAT and lot of instances are left out. So the average is only for full-rank random 3SATs. If this implies P=NP, Polynomial Hierarchy PH collapses to P implying existence of PH-complete problems.

---
590. (FEATURE) Program Analysis + Software Analytics - Valgrind/Callgrind/KCachegrind Call Graph DOT file creation and its Cyclomatic Complexity - 18 August 2018
---
1. Valgrind/Callgrind are the standard tools for call graph profiling and memory leak check of userspace code.
2. Cyclomatic Complexity is already computed for DOT files created by SATURN Program Analyzer for Linux Kernel
by VIRGO SATURN Program Analysis Driver. For userspace call graph DOT files are created by Kcachegrind complementing SATURN in userspace.(https://github.com/shrinivaasanka/virgo64-linux-github-code/blob/master/virgo-docs/VIRGO_SATURN_Program_Analysis_Integration.txt)
3. Valgrind for an example Spark program (Factorization) is invoked as:
valgrind --tool=callgrind -v --dump-every-bb=10000000 /media/Ubuntu2/spark-2.3.1-bin-hadoop2.7/bin/spark-submit ../DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py 819289
4. callgrind.out files are written to:
      callgrind.DiscreteHyperbolicFactorization_TileSearch_Optimized.out.14531 suffixed by process id.
5. callgrind.out file is viewed by kcachegrind (requires KDE) and call graph is exported as DOT and image files in:
      5.1 kcachegrind_callgraph_DiscreteHyperbolicFactorization_TileSearch_Optimized.png
      5.2 kcachegrind_callgraph_DiscreteHyperbolicFactorization_TileSearch_Optimized.dot
6. CyclomaticComplexitySparkMapReducer.py has been updated for doing Spark

wordcount of both SATURN and Valgrind/Callgrind/KCachegrind DOT Control Flow Graph / call graph files and print Cyclomatic Complexity.
7. This leverages on already implemented Convolution Net Deep Learning of Graphic Performance Patterns by Perf and FlameGraph.
8. Essentially Program Behaviour Analysis in userspace+kernelspace reduces to Graph Mining of these DOT graphs.

---
591. (FEATURE) Program Analysis + Software Analytics - Graph Mining of Call graph and CFG DOT files
- 20 August 2018

1.GraphMining_GSpan.py has been changed to check for an error condition of number of vertices per edge
2.New function graph_mining() has been included in software_analytics/CyclomaticComplexitySparkMapReducer.py for invocation of Graph Mining GSpan implementation to print frequent subgraph patterns in SATURN CFG and Valgrind/Callgrind/KCachegrind graph DOT files
3.logs for this have been committed to software_analytics/testlogs/CyclomaticComplexitySparkMapReducer.log.GraphMining.20August2018
4.This formally unifies the two ends: Program Analysis and Software Analytics (Deep Learning and Graph Mining)
5.Frequent Subgraphs in DOT files of Call graph and Control Flow Graph(CFG) extract static patterns in underlying software running on a system while the FlameGraph/Perf Graphics DeepLearning Analytics extract runtime patterns
6.Theoretically, if SAT for a Program is created by SATURN, Least Square SAT Solver in NeuronRain could be model checker for the Program

---
592. (THEORY and FEATURE) SAT Solver - CVXOPT - L1-norm Regularized Least Squares Quadratic Program l1regls()
and gesv() - 22 August 2018

1.This commit includes a special SAT Solver in CVXOPT documentation example l1regls() [https://cvxopt.org/examples/mlbook/l1regls.html] which solves the L1-norm regularized Least Squares Quadratic Program(QP):
        minimize error |AX-B|^2 + L1Norm(|X|)
which is the most accurate Least Square Solution. Geometric intuition and preference of L1 norm for least squares is depicted in Q&A:
https://www.quora.com/Why-is-L1-regularization-better-than-L2-regularization-provided-that-all-Norms-are-equivalent

2. In the context of SAT, previous minimization implies, the polynomial approximating the binary assignment
heaviside step function is as close as possible to satisfying each clause (because B is unit vector of 1s for
each clause) and sum of real fractional assignments (real parity) is minimum. This can be visualized as a
sinusoidal string pulled on both directions and string settles at an equilibrium curvature which minimizes the
QP above.

3. Prerequisite l1regls from https://cvxopt.org/_downloads/l1regls.py has to be downloaded and PYTHONPATH must be set to it.

4. CNFSATSolver.py has been refactored with if clauses for the least square algorithm invoked. For equal variable-clause "lapack()" invokes gesv() and for unequal variable-clause gels() is invoked

5. Few 1000 variables * 1000 clauses random 3SAT instances have been solved for l1regls() with MAXSAT approximation ratio 96%-97%

6. CVXOPT gesv() has been tested again for 10 variables * 10 clauses and similar MAXSAT approximation ratio of 99%-100% has been observed. gesv() throws ArithmeticError if A is singular (not invertible, determinant 0) while gels() throws ArithmeticError if A is not full-rank - gesv() is exact solver and gels() is least square solver. (NumPy solve() internally uses lapack gesv). So the logs filter out singular random 3SAT matrices.

References:

592.1 Advanced Engineering Mathematics - Tenth Edition - [Erwin Kreyszig] - https://www.safaribooksonline.com/library/view/advanced-engineering-mathematics/9780470458365/Chapter006.html#ch006-sec003 - Many Unit Heavyside step function and Laplace Transforms. Randomized Rounding of real assignments creates a many unit Heavyside step function.

---
593. (FEATURE) Audio/Music Pattern Mining - Audio to Time Series - 24 August 2018

1.Similar to image_pattern_mining/ImageToBitMatrix.py, new file AudioToBitMatrix.py has been committed to
repository for converting an audio file (e.g MP3, .wav) to time series array
2.AudioToBitMatrix.py imports librosa (audioread) Python library which returns waveform time series floating point array and sampling rate.
3.audio-to-bitmatrix function has optional arguments to specify if the time series is binary or float (defaults to float) and duration of the timeseries (defaults to full)
4.Binary time series invokes bin() on each time series point
5.An example audio MP3 file has been converted to time series and logs have been committed to testlogs/AudioToBitMatrix.log.24August2018

---
594. (THEORY) Horn3SAT and Least Squares Approximate SAT Solver - 7 September 2018

Horn3SAT is the special case of 3SAT in which each clause is Horn clause having single positive literal known
as Head and all other literals are negative. Horn3SAT is P-complete and is solved by unit propagation algorithm in polynomial time. Unit propagation algorithm has 2 repetitive phases: *) Remove all unit clauses having single literal *) Remove negated literal in all clauses. Horn3SAT can be solved by Least Squares Approximate Solver by creating random Horn3SAT matrix similar to random 3SAT matrix. This random Horn3SAT matrix is very sparse and has only one 1 per row for head literal and all other columns are 0s. Since Horn3SAT is P-complete, there exists a deterministic (and possibly parallel because P-completeness implies NC) algorithm to solve it one of which is unit propagation. Solution to this Horn3SAT random matrix Linear system of equations for equal variable-clause is deterministic polynomial time (gesv() solver can be applied for non-singular instances) while unequal variable-clauses are least squares solvable by algorithms - lsmr/gels/lsqr etc., Cloud algorithms like lsrn can solve Horn3SAT random matrix instance in NC if number of parallel processors are logdepth and polynomial size.

References:

594.1 Unit Propagation Horn3SAT Solver - https://en.wikipedia.org/wiki/Unit_propagation

594.2 Binary Least Squares and SAT Solver - http://www.mirlab.org/conference_papers/International_Conference/ICASSP%202011/pdfs/0003780.pdf - Binary Least Squares Solution to AX=B for boolean X is NP-Hard - Semidefinite Relaxation of Least Squares - Gaussian Randomization for obtaining binary values for X from feasible Semidefinite solution - Section 3 - Page 3782

---

595. (FEATURE) Music/Audio Pattern Mining - Histogram and Probability Distribution Function extraction (PDF)
- 7 September 2018

1.New function audio_features() has been added for analyzing features in audio/music signal
2.Considering audio as time series there are 2 standard ways to extract probability distribution from
time series: 1) Histogram 2) Bandt and Pompe - [Andres M. Kowalski,Maria Teresa Martin,Angelo Plastino and George Judge] - www.mdpi.com/1099-4300/14/10/1829/pdf
3.audio_features() invokes histogram() function and extracts PDF from NumPy by setting density=True
4.Comparing two music signals amounts to finding Jensen-Shannon Divergence of two probability distributions
5.Histogram for an example audio signal is printed in testlogs/AudioToBitMatrix.log.7September2018

---

596. (FEATURE) Music/Audio Pattern Mining - Jensen-Shannon Divergence of Probability Distributions of two
audio signals - 8 September 2018

1.audio_features() in AudioToBitMatrix.py returns the probability histogram and bins in a tuple
2.JensenShannonDivergence.py has been rewritten to compute normalization inline (normalize() function has been
removed), import audio functions from AudioToBitMatrix.py and compute probability histograms for two
different audio files.
3.logs for Jensen-Shannon Divergence distance between two audio files have been committed to testlogs/JensenShannonDivergence.log.8September2018 (For same audio, distance is zero)

---

597. (THEORY) Computational Geometric Sequential Factorization Optimization by Wavelet Trees, Tile Summation
Ray Shooting Queries and Error Probability - related to 506, 575 - 14 September 2018, 15 September 2018,
16 September 2018

Sequential Factorization Optimization by Tile Summation Ray Shooting Queries and searching the epsilon radius
in the vicinity of approximate factors described earlier have the following error probability in failing to
find a factor:

If epsilon radius is O((logN)^l) around an approximate factor, probability of not finding a factor =
      distance between periphery of 2 epsilon radius circles surrounding an approximate factor

```
      ------------------------------------------------------------------------
--------------
                  N/kloglogN

      = [N(m+1) - Nm] / kloglogN - 2q(logN)^l
      ---------------------------------------
                  N/kloglogN

      = N/kloglogN - 2q(logN)^l
      -------------------------
            N/kloglogN
```

For low error probability, numerator tends to 0:

```
      N/kloglogN = 2q(logN)^l
      => l = log(N/2kqloglogN)/loglogN = O(log(N/loglogN)/loglogN) which is
exponential.
```

For constant l, probability of error is non-trivial. This epsilon radius vicinity search applies to other ray shooting algorithms too (by replacing Hardy-Ramanujan estimate with the corresponding ray shooting algorithm and equating to tile summation). Advantage of tile summation is it always results in an integral value and coincides with some factor point on xy-plane.

```
------------------------------------------------------------------------------
--------------------------------
598. (THEORY) Hastad Switching Lemma, Least Squares Approximate SAT Solver,
Random Restrictions, CNFSAT
to DNFSAT - 18 September 2018
------------------------------------------------------------------------------
--------------------------------
Least Squares Approximation by Random 3SAT matrix analysis previously has so far
solved almost 99%-100% clauses
(observed) excluding singular random 3SAT matrix instances. This implies binary
assignments to some variables
are not right by randomized rounding of real assignments, falsifying some of the
clauses. Hastad Switching Lemma
implies setting values to alpha fraction of variables (restriction) in a CNF(or
DNF) formula and evaluating the restricted formula by switching it to DNF(or
CNF) and computing its decision tree abides by the inequality:
      Pr[DTdepth(f/alpha) > d] <= (10*alpha*w)^d
for s = sigma*n <= n/5, d >= 0, w is formula width. Least Squares Approximation
thus can be construed as restriction generator by following algorithm:
      (*) Solve a random 3SAT matrix by Least Squares (e.g gesv())
      (*) Variables in all satisfied clauses in least squares solution form a
random restriction - set variables
      (*) Variables in all unsatisfied clauses in least squares solution
(excluding variables in satisfied clauses) are made unset variables
      (*) Switch the restricted CNF3SAT to a DNF3SAT in unset variables
      (*) DNF3SAT is solvable in polynomial time (linear)

References:
-----------
598.1 Hastad Switching Lemma - [Johan Hastad] - PhD Thesis - Chapter 4 - Lemma
4.1 - "Let G be an AND of ORs all of size <= t and rho a random restriction from
R(p). Then the probability that G/rho cannot be written as an OR of ANDs all of
size < s is bounded by alpha^s where alpha is the unique positive root to the
equation:
      (1 + 4p/(1+p)*(1/alpha))^t = (1 + 2p/(1+p)*(1/alpha))^t + 1" and Stronger
Switching Lemma - Lemma 4.2 - "Let G = /\ Gi where Gi are ORs of fanin <= t. Let
F be an arbitrary function and p, a random restriction in R(p).Then Pr[min(G) >=
s/Fp = 1] <= (alpha)^s" - https://www.nada.kth.se/~johanh/thesis.pdf
598.2 Hastad Switching Lemma - [Vikraman Arvind] - JRF Course Notes - Complexity
2 - August-December 2010 - Institute of Mathematical Sciences, Chennai.
598.3 Randomized Rounding of Set Cover - [CR Subramanian] - JRF Course Notes -
```

Randomized Algorithms - August-December 2010 - Institute of Mathematical Sciences, Chennai. Least Squares SAT Solver rounds real assignment by midpoint of max-min values in assignment vector. Alternatively, each variable x* in real assignment vector can be rounded with respect to maximum value f in real assignment as: x = 0 if x* < 1/f else x = 1.
598.4 Hastad Switching Lemma - Course Notes - [Jonathan Katz] - Corollary 3 - "Let f:{0,1}^n - {0,1} be computed by a DNF formula (resp. CNF formula) of width at most w. Let alpha be a random s-restriction with s <= n/5. Then f/alpha can be computed by a CNF formula (resp. DNF formula) of width w except with probability at most (10sw/n)^w => Pr[DT(f/alpha) >= s] < (10sw/n)^w (or) Pr[DTdepth(f/alpha) > d] <= (10*sigma*w)^d for s=sigma*n <= n/5 and d>= 0." - http://www.cs.umd.edu/~jkatz/complexity/f05/switching-lemma.pdf
598.5 Switching Lemma - https://en.wikipedia.org/wiki/Switching_lemma - "...The switching lemma says that depth-2 circuits in which some fraction of the variables have been set randomly depend with high probability only on very few variables after the restriction. The name of the switching lemma stems from the following observation: Take an arbitrary formula in conjunctive normal form, which is in particular a depth-2 circuit. Now the switching lemma guarantees that after setting some variables randomly, we end up with a Boolean function that depends only on few variables, i.e., it can be computed by a decision tree of some small depth d. This allows us to write the restricted function as a small formula in disjunctive normal form. A formula in conjunctive normal form hit by a random restriction of the variables can therefore be "switched" to a small formula in disjunctive normal form...."
598.6 Cutting Planes - Gomory Cut - https://ocw.mit.edu/courses/sloan-school-of-management/15-053-optimization-methods-in-management-science-spring-2013/tutorials/MIT15_053S13_tut11.pdf - for Integer Linear Programs.
598.7 Chapter 14 - Cutting planes - Combinatorial Optimization - [Stiglitz-Papadimitriou]
598.8 Randomized Rounding Survey - [Aravind Srinivasan] - http://homepage.divms.uiowa.edu/~sriram/196/fall08/rr-final.pdf - Pr[Clause is satisfied] > 0.75*(sum of relaxed fractional assignments) - "...One intuitive justification for this is that if x∗(j) were “high”, i.e., close to 1, it may be taken as an indication by the LP that it is better to set variable j to True; similarly for the case where x(j) is close to 0. ..." - Least Squares SAT Solver applies this intuition for rounding.
598.9 Randomized Rounding - Randomized Algorithms - [Motwani-Raghavan] - Pages 81,96,105,106 - optimized wiring nets.

---
599. (FEATURE) Streaming Algorithms - Approximate Counting and Distinct Elements - 21 September 2018

599.1. Two new python implementations for Streaming Approximate Counting and Distinct Elements in a random subset
have been committed to repository
599.2. These are implementations of two algorithms from JRF presentation notes (on 28-29/10/2010) for course - Topics in Datamining - during August-December 2010, Chennai Mathematical Institute, Chennai.
599.3. They have been implemented because of their importance for processing Streaming Datasets e.g Spark Streaming

References:

599.4. Approximate Counting Algorithm - [Morris-Flajolet] - http://algo.inria.fr/flajolet/Publications/Slides/aofa07.pdf
599.5. Distinct Elements algorithm - [BarYossef] - https://slideplayer.com/slide/4943252/

---
600. (FEATURE) Streaming Abstract Generator - Absolute paths to Relative paths -

21 September 2018

File Datasource paths have been changed from absolute to relative paths

---
601. (THEORY) Algebraic Geometry and Computational Geometric Factorization - some definitions
- 24 September 2018

Euclidean Rings are defined as:
For a,b in ring R, both non-zero, there exist t,r in R such that a=tb + r where r >= 0. Unique Factorization Theorem is defined on Euclidean Rings.
Set of Integers are examples of Euclidean Rings.

Let R be a Ring. An Ideal I of R is a subgroup of (R,+) and for all x in I and for all r in R, x.r and r.x in I.
Set of all Integers Z divisible by an integer n is an ideal denoted by nZ.
Ideals can be generated by a subset X
of Z. Special case is when X is a singleton X={a} for some a in Z, named Principal Ideal:
      => aZ = {ar | r in Z}
      and Za = {ra | r in Z}
These ideals are left and right principal ideals generated by X={a} i.e multiples of a.

Every hyperbolic tile segment in Computational Geometric Factorization is the union of intersections of principalideals - one infinite ideal y1*Z parallel to x-axis and other k infinite ideals xi*Z parallel to y-axis in xy-grid (each intersection is a pixel in xy-grid) for xi and y1 in Z - this intersection of principal ideals is finite:
      y1*Z = {y1, y1*2, y1*3,...}
      x1*Z = {x1, x1*2, x1*3,...}
      x2*Z = {x2, x2*2, x2*3,...}
      x3*Z = {x3, x3*2, x3*3,...}
      ...
      xk*Z = {xk, xk*2, xk*3,...}

      Tile segment = (x1*Z /\ y1*Z) \/ (x2*Z /\ y1*Z) \/ ... (xk*Z /\ y1*Z)

In Algebraic Geometric terms, a hyperbolic pixelation is union of tile segments (union of union of intersections of principal ideals). Sum of lengths of first n tile segments of pixelated hyperbolic curve:
      N/[1*2] + N/[2*3] + ... + N/[n*(n+1)] = Nn/(n+1)
      N/[1*2] + N/[2*3] + ... + N/[n*(n+1)] <= N/1^2 + N/2^2 + N/3^2 + N/4^2 + ...
      N/[1*2] + N/[2*3] + ... + N/[n*(n+1)] <= (RiemannZetaFunction(s=2))*N
      N/[1*2] + N/[2*3] + ... + N/[n*(n+1)] <= 1.6449*N

An algebraic subset of Z^2 for hyperbolic variety is defined as (N is number to factorize):
      V(S) = { (x1,y1) in Z^2 | x1*y1 - N = 0 }
which is the point-set on the hyperbolic tile segments (equivalent to previous definition based on principal ideals).

Computational Geometric Hyperbolic tile segments pixelation as Variety over Finite Fields:

Algebraic Varieties are defined as set of zeros of polynomial equations. If K is

a Field and N is a positive integer, an affine N-Space over K is defined as:
A^N(K) = {(a1,a2,...,aN)| a1,a2,...,aN in K}
A subset of affine N-Space is an affine Algebraic Variety over K defined as:
X(K) = {a in A^N(K)| f1(a) = 0,f2(a) = 0,...,fr(a)=0}
if there are polynomials f1,f2,...,fr having coefficients from K[T1,T2,...,Tn]. In Computational Geometric Factorization, Hyperbolic tile segment pixelation is an affine Algebraic Variety defined over a finite field or Galois Field K of order p^n >> N (p^n is a prime power) satisfying polynomial x1*y1 - N = 0 for (x1,y1) in affine 2-Space A^2(K). Counting number of elements in finite algebraic variety is non-trivial problem - in the context of geometric factorization, this is tantamount to finding number of points in hyperbolic tile segments.

References:

601.1 Algebraic Geometry - Chapters 1 and 2 - [JS Milne] - https://www.jmilne.org/math/CourseNotes/AG.pdf
601.2 Principal Ideal - https://en.wikipedia.org/wiki/Ideal_(ring_theory)
601.3 Topics in Algebra - [Israel N. Herstein] - Pages 143-144 - Euclidean Rings
601.4 Value of Riemann Zeta Function at s=2 - https://en.wikipedia.org/wiki/Particular_values_of_the_Riemann_zeta_function - pi^2/6=1.6449...
601.5 Counting Points of Varieties over Finite Fields - [Sudhir Ghorpade, IIT Bombay, Mumbai] - http://www.math.iitb.ac.in/~srg/Talks/IRCCAwardLecture_Ghorpade_29Aug2012.pdf

---
602. (THEORY) Software Analytics - GraphFrames - GraphX - Parallel Graph Algorithms for CallGraph/CFG/SATURN Graph DOT files - 27 September 2018

1.NeuronRain till now has a native implementation of parallelized graph processing (e.g definition graph growth
implementation in InterviewAlgorithm/ for texts) and sequential NetworkX Graph objects
2.Spark has parallel graph library GraphX and a separate repository GraphFrames which internally wraps GraphX
(http://graphframes.github.io/api/python/graphframes.html)
3.This commit introduces an option to choose GraphFrames in analyzing Software Analytics CallGraph/CFG/SATURN DOT files in CyclomaticComplexitySparkMapReducer.py and invokes few parallel algorithm functions for indegrees, strongly connected components, triangle counting.
4.Number of strongly connected components, degrees, triangles in CallGraph/CFG/SATURN graphs are cyclomatic measures of code complexity (both static and dynamic)
5.logs for GraphFrames is committed to testlogs/CyclomaticComplexitySparkMapReducer.log.GraphXFrames.27September2018
6.GraphFrames can be applied not just to Program Analysis DOT files but to any other source (TextGraph, Social Networks etc.,)`
7.Spark Commandline for GraphFrames package: /media/Ubuntu2/spark-2.3.1-bin-hadoop2.7/bin/spark-submit --packages graphframes:graphframes:0.6.0-spark2.3-s_2.11 CyclomaticComplexitySparkMapReducer.py

---
603. (THEORY and FEATURE) ImageGraph creation from an Image - Keras-Theano - related to 587 and 588 -
EventNet Tensor Products Algorithm for Merit of Large Scale Visuals - 30 September 2018

1.This commit introduces a primitive but crucial implementation of EventNet Tensor Products Algorithm

for Large Scale Visuals which creates a text graph from information gained from processing of a visual (.jpg).
2.Keras is a framework for Computer Vision which supports pre-trained ImageNet datasets (VGG, ResNet) annotated
to ImageNet vertices by Convolution Neural Networks.ImageNet presently has 14 million vertices.
3.Keras has support for TensorFlow, Theano, CNTK backends. A python implementation which imports Keras and sets
the backend to Theano and predicts objects in an image from ImageNet has been committed to image_pattern_mining/ImageNet/ImageGraph_Keras_Theano.py
4.This script loads an image in keras-theano, converts it to an image array, expands its dimensions and predicts the objects in the image by ResNet50 pre-trained ImageNet dataset.
5.Predictions for objects in image are decoded into array of tuples of the form (id, imagenet_vertex, probability).
6.An example prediction for few images is described in : https://www.pyimagesearch.com/2016/08/10/imagenet-classification-with-python-and-keras/ along with probabilities for objects annotated.
7.These predicted ImageNet vertices are concatenated into a text and passed on to RecursiveGlossOverlap_Classifier.py which creates a Text Graph - ImageGraph - by invoking RecursiveGlossOverlapGraph() on this text.
8.This algorithm essentially maps an Image to a TextGraph. Presently only still images are supported - extending to Video is straightforward by getting all frames in a video and computing ImageGraphs for all frames and Tensor Products of the causally related ImageGraphs.
9.logs for few images (one of which is courtesy: https://www.aazp.in/live-streaming/ - WhiteTiger_1.jpg) have been committed to image_pattern_mining/ImageNet/testlogs/
10.Graphs printed in logs show fairly reasonable inference of prominent entities in image.

---
604. (FEATURE) Music Pattern Mining - Onset of Notes Detection - 1 October 2018 - related to 67,68,69
---
1. audio_features() function in AudioToBitMatrix.py has been updated to compute notes onset from audio signal
2. Notes Onset Detection involves finding peaks in onset strength of the signal time series.
3. Extracting notes from signal converts the music time series to a string from a language of notes alphabets.
4. Most algorithms for audio analysis are time-series (fft,statistics etc.,) based. Considering music as a Formal language opens new vistas into theoretical analysis of music. For example, finding a deterministic finite state automaton for sequence of notes in a music signal is measure of complexity in music.
5. Not just automata, variety of string complexity measures (e.g Kolmogorov complexity - MDL) can effectively
describe music notes strings.

---
605. (THEORY) Computational Geometric Factorization, Parallel Planar Point Location, Ray Shooting Queries,
Multisearch, PRAM and BSP models - 2 October 2018 - related to 34, 506, 575, 597

Factorization by Computational Geometric algorithms - k-mergesort, segment trees, wavelet trees, ray shooting
queries - described earlier implement a generic idiom of Planar Point Location which locates factor points
inside some of the polygons in the planar subdivision created by the hyperbolic pixel array segments. Planar

Point Location is a special problem in Multisearch which is defined as: Given a set of query points and set of
line segments in 2D plane, multisearch returns the line segments containing the query points which is exactly
what is required in factorization. There are lot of parallel planar point location algorithms for PRAM and BSP
model and thus in NC (references in 34). After locating a pixel array polygon by parallel planar point location, factor can be found by binary search of the segment polygon containing the factor because each hyperbolic segment is implicitly sorted - one of the ordinate axes is constant and other increases i.e each tile is an arithmetic progression. Parallel Planar Point Location algorithm in reference 34.83 can locate a polygon containing factor point in O((logN)^2) PRAM time with additional O(logN) sequential time for binary search to locate the factor within polygon.

Ray shooting query algorithms described earlier reduce the necessity of parallelism to great extent by making an
educated guess of approximate location of factor points by applying number theory results and search their circular vicinity for exact factors by k-mergesort, segment trees or wavelet trees. Probability of error in ray shooting has been derived earlier. Planar Point Location and Ray Shooting queries can be combined into one by following
algorithm for multisearch:

(*) Sequentially shoot ray queries for O(loglogN) approximate prime factors with average spacing O(N/loglogN) between rays shot from origin. This is O(loglogN) sequential time (assuming Hardy-Ramanujan estimate)

(*) Locate the segments (pixel array polygons) containing each of the approximate factors by Parallel Planar Point Location. This returns O(loglogN) segments in O((logN)^2*loglogN) parallel time. With high probability a segment containing approximate factor might also contain exact factor:

(*) Each tile segment is of variable length N/[x(x+1)] but spacing between prime factors is equalkN/loglogN for some constant k

(*) Error occurs if N/[x(x+1)] < kN/loglogN (or) loglogN < k(x(x+1))

(*) Solving for x in x^2 + x - (loglogN/k) > 0:

x > -0.5 + 0.5*sqrt(1 + 4loglogN/k)

(*) Each O(loglogN) segment(polygon) can be binary searched in O(logN*loglogN) sequential time.

(*) Total Parallel and Sequential time of previous 3 steps = O(loglogN + (logN)^2*loglogN + logN*loglogN)

(*) This is just another sequential optimization which makes some parallel computation sequential.

An average case sequential optimization for removing parallelism in Computational Geometric Factorization:

(*) Average number of tiles between two approximate prime factors m and (m+1) found by ray shooting:

= (m+1)/[kloglogN - m - 1] - m/[kloglogN - m]

because number of tiles till m-th prime factor = n = m/[kloglogN - m] for m=1,2,3,...,kloglogN

(*) Polygon (hyperbolic tile segment) containing the ray shot approximate factor can be found by equation in 2-level binary search sequential optimization described previously - Tile containing approximate factor point x can be expressed by inequality:

Nk/(k+1) < x < N(k+1)/(k+2)

k < x/(N-x)

integer round-off of k is the index of tile interval containing x. This a simple sequential planar point location identity in elementary arithmetic with no necessity for PRAMs or BSP.

(*) Each consecutive tile in N/kloglogN spacing between approximate factors can

be found by tiling equation N/[x(x+1)] - subtract y coordinate by N/[x(x+1)] and increment x coordinate by 1 or vice versa - and each of these tiles in N/kloglogN spacing can be individually binary searched in O(logN).
(*) Thus total average sequential time to sweep binary search tiles in N/kloglogN spacing between 2 consecutive approximate factors found by ray shooting:
      = O([(m+1)/[kloglogN - m - 1] - m/[kloglogN - m]] * logN)
(*) Maximum number of tiles in N/kloglogN spacing occurs by setting m=kloglogN - 2 (because of geometric intuition of hyperbolic tiling, most tiles occur within an interval in topmost or rightmost extreme of hyperbolic arc which contains kloglogN-th factor on either axes):
      = O([(kloglogN - 2 + 1)/[kloglogN - kloglogN + 2 - 1] - (kloglogN - 2)/[kloglogN - kloglogN + 2]])
      = O(kloglogN - 1 - (kloglogN - 2)/2)
      = O(kloglogN)
=> Maximum time to binary search tiles in N/kloglogN spacing = O(loglogN * logN)
=> Each of O(loglogN) spacings between approximate factors can be searched in:
      O(loglogN*loglogN*logN)
average case sequential time with no necessity for parallel processing.

References:

605.1 Truly efficient parallel algorithms: 1-optimal multisearch for an extension of the BSP model - [Armin Baumker, Wolfgang Dittrich, Friedhelm Meyer auf der Heide] - Elsevier Theoretical Computer Science - https://core.ac.uk/download/pdf/81931675.pdf - Parallel Planar Point Location by Multisearch in BSP* model - Section 1.1.
605.2 Communication Effcient Deterministic Parallel Algorithms for Planar Point Location and 2d Voronoi Diagram - [Mohamadou Diallo , Afonso Ferreira and Andrew Rau-Chaplin] - Section 2 - Hyperbolic pixelation is in a sense a Voronoi tessellation in which factor points are ensconced by the pixel array polygons.
605.3 Computational Geometric Planar Point Location on Arrangements - Section 9.9.1 - Randomized Algorithms - [Motwani-Raghavan] - Pixelated Hyperbolic tile segments are adjacent arrangements
605.4 Rectifiable Curves - Principles of Mathematical Analysis - [Walter Rudin] - Pages 136-137 - Section 6.26 and Theorem 6.27 - Pixelation of Hyperbolic arc is exactly rectification - map is defined by y = N/x and interval [1,N] is partitioned into tile segments of lengths N/[x(x+1)] and sum of lengths of tile segments is the length of polygonal path.
605.5 Rectification of Curves and Bresenham Line Drawing Algorithm - https://en.wikipedia.org/wiki/Arc_length and https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm - Rectification also called as Arc Length finds length of a curve by approximating it with line segments which
may not be axis parallel whereas Computational Geometric Factorization approximates hyperbola by only axis parallel straightline segments which is exactly similar to Bresenham line drawing algorithm - line is replaced by hyperbolic arc (illustrated in https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm#/media/File:Bresenham.svg).

---
606. (THEORY and FEATURE) VideoGraph EventNet Tensor Product Merit for Large Scale Visuals - 3 October 2018
---
1. Two new functions have been committed to ImageGraph_Keras_Theano.py for reading a Video file, extracting
frames from the video upto certain number and extract object information by Keras-Theano ImageNet predictions
for each frame.
2. imagenet_videograph() function depends on OpenCV 3.4.3 (cv2) Python for reading Videos and writing Frame
JPEG files suffixed by unique id. Each frame is mapped to a TextGraph -

ImageGraph - by imagenet_imagegraph()
3. videograph_eventnet_tensor_product() function computes tensor products of pairs of Frame ImageGraphs and
returns an EventNet Tensor from NetworkX.
4. logs, an example Video (MP4) and 2 sample extracted frames (JPEG) for EventNet Tensor representation of
a video have been committed to testlogs/ImageGraph_Keras_Theano.log.3October2018

5. From this EventNet Tensor any kind of intrinsic merit can be computed - e.g previously described ImageNet based inverse weights, Graph Edit Distance between two Frame ImageGraphs, Tensor Rank for connectedness etc.,

---

607. (THEORY and FEATURE) Video EventNet Tensor Products - Inverse Distance Merit of Large Scale Visuals
- 4 October 2018

1.New function inverse_distance_intrinsic_merit() has been included in ImageGraph_Keras_Theano.py. It
iterates through each tensor product entry of Video EventNet Tensor Products matrix and for every edge
(a,b)-(c,d) in tensor product computes inverse wordnet distance (which is basis for ImageNet):

```
        1/dist(a,b)*1/dist(c,d)
```

and populates a weights matrix per tensor product graph.
2.Each tensor product in EventNet Tensor corresponds to causality between any 2 frames in the video.
3.An example merit computation log has been committed to testlogs/ImageGraph_Keras_Theano.log.4October2018
which shows merits for 4 tensor products (2 frames * 2 frames).
4.Lot of weights in merit matrix in logs have huge values implying high causality between two frame events.
5.EventNet Tensor Product Algorithm is computationally intensive. Each ImageGraph per frame encodes relationship
between objects (actors) in the frame event.
6.Size of the EventNet Tensor is O(number_of_frames * number_of_frames * average_number_of_actors_per_frame * average_number_of_actors_per_frame) and memory intensive too.

---

608. (THEORY and FEATURE) Video EventNet Tensor Products - Emotional Merit of Large Scale Visuals
- 15 October 2018

---

1. EventNet Tensor Product representation of a Large Scale Visual just infers the connectedness of the
information in the video with no emotional or sentiment scoring.
2. To redress this, new function has been included in ImageGraph_Keras_Theano.py to lookup the edges of
tensor product graph in empath and aggregate the sentiments of the vertices as a tuple which is qualitative emotive merit of the video.
3. Two new example visuals have been added: ExampleImage_1.jpg and ExampleVideo_2.mp4 in addition to earlier
ExampleVideo_1.mp4 and images.
4. Keras has mistook ExampleImage_1.jpg to be a theatre audience which is indeed a classroom photo and annotated
it accordingly (probably because seating and ambience are similar in both). Also Empath has misjudged the alumni website content in ExampleVideo_2.mp4 to large extent excluding "college" as emotional annotation. Logs have been committed to testlogs/ImageGraph_Keras_Theano.log.15October2018

5. These Photo and Videos are courtesy (and related to self):
(*) ExampleVideo_1.mp4 - https://www.cmi.ac.in/people/fac-profile.php?id=shrinivas - Chennai Mathematical Institute - Research Scholar website - 2010-11
(*) ExampleImage_1.jpg - http://www.angelfire.com/id/95cse/album.html - PSG College of Technology - 1995-99 CSE batch alumni website album - Classroom Photo (class1.jpg)
(*) ExampleVideo_2.mp4 - https://alumni.psgtech.ac.in/profile/view/srinivasan-kannan-1 - PSG College of Technology Official Alumni website
6. Frames captured by OpenCV2 are prefixed by the Video file name.
7. THEANO_FLAGS have been set as: declare -x THEANO_FLAGS="cxx=/usr/bin/g++-6"

---
609. (THEORY) EventNet Tensor Products - Example for Movie and Youtube Merit - 16 October 2018
- related to 608

Previous videos were proof-of-concept implementations for utility of EventNet Tensor. Following is stated just as a theoretical example with no testcases because computation involved is massive requiring millions of frames and their EventNet Tensor Product - EventNet Tensor Products Algorithm can be applied to set of movie/youtube datasets to deduce the connectedness and emotions in the videos. Since most movie and youtube channels are emotions oriented which are expressed visually, comparing the emotion annotated tuples (e.g by Empath previously described) of any pair of movies/youtube videos, could rank them by emotional quotient approximately. Also the previous Empath emotion annotation creates a set of emotions and not a hashmap of emotion annotations to their number of occurrences in the Audio-Visual which is yet another indicator. Comparing emotions occurrence hashmaps of two Audio-Visuals is a ranking measure - requires sorting two hashmaps by emotions occurrence values, comparing top percentile emotions in two Audio-Visual hashmaps and finding correlation coefficient between the two probability distributions. Added advantage could be speech recognition in the AV frame-by-frame or for set of frames which is not done by previous Keras-Theano implementation and creating AudioGraph from it (there is no AudioNet library so far).

---
610. (THEORY) Summary of Large Scale Visuals and EventNet Subgraphs - related to 158,423 - 19 October 2018
and 20 October 2018

Guessing/Discovering a Larger EventNet causality Graph from Subgraphs of EventNet has been mentioned earlier.
An opposite of it is to summarize a huge EventNet Graph into a small summary EventNet subgraph without much
loss of causality information. Text Summarization is text analysis application of Graph Summarization which has been earlier described and implemented by Recursive Lambda Function Growth algorithm. Realworld application of
Graph Summary is Video Summarization by EventNet Tensor Products representation e.g a movie or a youtube video is a small duration summary (minutes or hours) of real life events (and their causalities) enacted by set of people over many years - this involves sampling an EventNet Tensor Product Graph of a Video at relevant vertices and induce a summary subgraph on it which preserves meaning. Similar to Text Summarization, Video Summary could also be approximated by choosing dense subgraphs (or high core number vertices) of Video EventNet Tensor Products. Topological Sort of Dense Subgraph of Video EventNet Tensor Product Graph gives a flattened summary of the video by some ordering (a gist of important frames).

---
611. (THEORY and FEATURE) Video Emotions Hashmap, Video Core Topological Sort Summary - 25 October 2018
- related to 611

(*) Empath emotions of the Video EventNet Tensor Products Graph have been aggregated in a hashmap. This facilitates quantitative emotional scoring of the video. Hashmap correlates the empath emotion and its cumulative score.
(*) Core (Main core) of the Video EventNet Tensor Product Graph has been computed and is sorted topologically for creating some ordering for the summary.

(*) New example MP4 video has been created from 6 JPEG photos (in which I was present) and google search related to self (ExampleVideo_3.mp4) :
Frame 1 - Self - Mahabalipuram, Chennai - 2012 (Copyright: Self)
Frame 2 - Self - photo (Copyright: Self)
Frame 3 - Self - VISA photo (Copyright: Self)
Frame 4 - Self - Family photo (Copyright: Self)
Frame 5 - Self - IEC Sun Microsystems - Bengaluru - 2004 (Copyright: Ex-Colleagues at Sun Microsystems)
Frame 6 - Self - IEC Sun Microsystems - Bengaluru - 2000 (Copyright: Sun Microsystems/Oracle)
Frame 7 and 8 - Google Search of Self
(*) Previous recording is a mix of photos of varied genre. Sentiment hashmap for this video prints (excerpts):
Sentiment Analysis of the Video: [('work', 1484.0), ('toy', 424.0), ('tool', 848.0), ('technology', 3180.0), ('social_media', 1060.0), ('shopping', 424.0), ('science', 1060.0), ('restaurant', 848.0), ('reading', 424.0), ('programming', 2544.0), ('play', 424.0), ('phone', 1696.0), ('office', 1060.0), ('negative_emotion', 0.0), ('musical', 424.0), ('music', 424.0), ('messaging', 1272.0), ('meeting', 424.0), ('internet', 2968.0), ('hiking', 424.0), ('furniture', 212.0), ('dance', 212.0), ('computer', 5088.0), ...]. This is a remarkable inference of the Video content by Keras-Theano ImageNet giving high weightage to office and technology as emotional annotations.
(*) Logs for this have been committed to testlogs/ImageGraph_Keras_Theano.log.25October2018

---
612. (FEATURE) Video EventNet Core Topological Sort Summary - Bugfixes - 28 October 2018

(*) Some errors in inverse distance merit computation of the Video EventNet Tensor have been corrected which
was earlier a 2-dimensional tensor and has been made 3-dimensional in this commit.
(*) Core Summary of the Video has also been changed to compute maximum merit for each tensor product inverse
distance merit vector and apply a threshold filter for the distance merit.
(*) Three dimensional EventNet Tensor is mapped to NetworkX Graph after previous threshold filter.
(*) Only causal frames above inverse distance threshold are included in the NetworkX graph and main core is computed on this graph
(*) self loops and parallel edges are removed and topological sorting is performed on main core.
(*) logs for this have been committed to testlogs/ImageGraph_Keras_Theano.log.28October2018

---
613. (FEATURE) People Analytics - HR Analytics Implementation and Least Energy Intrinsic Merit of a
social profile - 9 November 2018 - related to 365, 443, 572

```
--------------------------------------------------------------------------------
-------------------------
(*) People Analytics is the generalized version of HR Analytics for analyzing
any social profile vertex.
(*) This commit adds SocialNetworkAnalysis_PeopleAnalytics.py python
implementation to repository which
at present analyzes LinkedIn profiles and computes:
      (*) Graph Tensor Neuron Network Intrinsic Merit of the profile text with
no field parsing - this treats
the social profile as an en masse text with no delineations and applies
Recursive Lambda Function Growth algorithm to create lambda function composition
tree of the resume text. Core classification and Centrality measures throw some
interesting insights:
            (*) Core number 4 identifies the class of the social profile
            (*) Maximum merit random walk of the lambda composition tree
approximately guesses the purport
of the text in the social profile
            (*) Betweenness and Closeness Centralities estimate the top-ranked
classes accurately compared to Degree and PageRank Centralities
      (*) Log Normal Fitness of the social profile after parsing the total work
experience and academic
stints of the profile - for linkedin this inverse log normal least energy
fitness is 1/(logE + logV) with no Wealth estimates - Wealth is caused by
E(Education) and V(Work/Valour) as far as professional profiles are concerned.
Lowest Value of Least Energy Fitness implies Highest Fitness. Parsing is done by
parse_profile() function which implements a switch design pattern - stint
polymorphs from work to academic tenures.
      (*) Log Normal Experiential Intrinsic Fitness of the social profile (by
applying mistake bound learning differential equation for recursive mistake
correction tree described earlier) - log normalized because of huge exponents
(*) Because of limitations due to Updated LinkedIn Scraping Policy
(https://www.forbes.com/sites/forbestechcouncil/2017/09/20/linkedin-vs-hiq-
ruling-casts-a-long-shadow-over-the-tech-industry/#19f535f35e6c), this is only
implemented as a non-crawled example for LinkedIn profile of self:
https://www.linkedin.com/comm/in/srinivasan-kannan-608a671 (pdf and text version
of pdf by pdf2txt)
(*) datetime python package is used to compute timedelta(s) and a regex matcher
isdaterange() function has been implemented for parsing date ranges in the
profile:
      (*) Work Experience Date Ranges have the Regex: <Month> <Year> - <Month>
<Year>
      (*) Academic Date Ranges have the Regex: <Year> - <Year> and months are
implicit
(*) This implementation does not assume a Web Crawler and Social Profile
Datasource and only requires a file input (datasource, file type and file name
are passed in as arguments to parse_profile()) - optionally, crawled social
profiles can be invoked as file arguments to parse_profile(). Presently only
linkedin datasource, text file types are supported. PDF parsing has been
implemented by PyPDF2 but there are IndirectObject related errors.

-------
Theory:
-------
An obvious application of HR analytics is feasibility of automatic recruitment
of people by profile analytics. Usual Interview and Written Test based process
of recruitment measures Intrinsic Merit (approximately subject to error) of
people. Analytics of Professional Networks is Fame (Degree etc.,) based. From
572.8 ("...Boolean majority function is equivalent to PageRank computed for 2
candidate websites on (n+2)-vertex graph (n user websites choose between 2
candidate websites - Good(1) and Bad(0), choice is fraction of outdegree from a
user website) normalized to 0 (smaller pagerank) and 1 (bigger pagerank).
Margulis-Russo threshold d(Pr(Choice=Good=1))/dp on this PageRank version of
Majority function is ratio of sum of influences of variables (user websites) to
standard deviation which has a phase transition to Good(1) at per-voter website
```

bias p >= 0.5..."). Majority Voting/PageRank or any other Fame/Degree measure is known to have phase transition from Low accuracy to High accuracy for per voter p-bias > 0.5 implying automatic recruiting is feasible if Fame lowerbounds Intrinsic Merit (Intrinsic Merit >= Fame). Stability or Resilience of Interview as LTF vis-a-vis Majority Function has been analyzed earlier in 365. Problematic condition for rendering automatic crowdsourced recruitment infeasible is: Intrinsic Merit << Fame. Comparison of Intrinsic Merit to Fame by correlation coefficients requires normalization of fame and merit rankings e.g fame and merit probability distributions and distance between these two distributions for same dataset. From references in 572, most experimental results in sports(e.g Elo rating), science (e.g research published) etc., show fame grows exponentially to merit implying merit is quantitatively proportional to log(Fame). For example, if PageRank/Degree centrality is computed on Professional Network Vertices and dynamic experiential intrinsic merit E of a profile at time point t is computed by previous implementation:

E = M*e^(kMt) = log(dv(t)) * e^(klog(dv(t))*t/clogt) / clogt for evolving degree dv(t)

crowdsourcing infeasibility condition reduces to:

log(dv(t)) * e^(klog(dv(t))*t/clogt) / clogt << dv(t)

(klog(dv(t))*t/clogt) << log(clogt * dv(t) / log(dv(t)))

---

614. (FEATURE) Social Network Analysis - People Analytics - Another example and Unicode corrections -
12 November 2018 - related to 613

---

1. RecursiveGlossOverlap_Classifier.py has been changed to ignore unicode errors in text strings
2. New example Curriculum Vitae TeX (self) has been analyzed by Recursive Lambda Function Growth Algorithm
for various intrinsic merit measure it prints.
3. Core classifier shows high relevance to human judgement and again Closeness and Betweenness centralities are
better than Degree and PageRank centralities (classes like "Silver" are probably produced because of relevance of "Silver" to "Hyderabad")
4. RecursiveLambdaFunctionGrowth.dot file has been recreated.
5. logs for this have been committed to testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.12November2018
6. Textual analysis of a CV is quite open-ended and unstructured - Nonetheless the classification by TextGraph core numbers gives reasonable estimate of crucial aspects in the resume

---

615. (THEORY) Hash Table Chaining, Locality Sensitive Hashing and Frequently occurring patterns - 13 November
2018 - related to all sections on LSH partitions, Majority Voting, Theoretical EVMs, Streaming,
Audio-Visual Pattern Mining, Bigdata Analytics

Locality Sensitive Hashing is a collision-supportive hash chaining which as described earlier has multitude
of applications in clustering similar elements of a set which can be votes for candidates(EVMs which colocate votes for a candidate in same bucket), bigdata or audio-visual spectrogram-histogram representation. Essentially, every chained hash table or LSH partition of a set is a histogram of probability distribution when viewed pictorially - e.g audio signals represented as series of animated histograms. Assuming dataset is a stream of these LSH partitions, problem is to decipher patterns underlying these histograms. There are LSH algorithms like Nilsimsa hash, Minhash etc., which find similarities of two sets (texts for example in Nilsimsa) by Jaccard, Hamming and other distance coefficients. If

each element in the stream of histograms/LSH partition/Chained Hashtable are considered as set of buckets, comparing two such histograms is tantamount to computing similarity of 2 consecutive sets of buckets by some hash metric e.g Minhash. It is worthwhile to mention that this histogram (set of buckets) pattern is ubiquitous across any algorithm doing an LSH partition or hash table bucket chaining. In the People Analytics example previously, LSH partition of People and Professional Profiles splits them into buckets of similar traits.

Distributed Hash Tables which geographically decentralize chunks of a huge table realise theoretical Electronic Voting Machines which have been defined earlier in terms of LSH partitions or Separate Chaining Hash tables in distributed/cloud setting - buckets of votes for candidates are distributed on cloud nodes. CAP Theorem applies to Theoretical EVMs implying only 2 of 3 (Consistency-serialization, Availability-readwrites are 100% reflected and Partition Tolerance-resilience to node/network failures) are feasible on an internet EVM based on LSH Partition/Separate Chaining. Real world EVMs are most vulnerable to network, sabotage or power failure (partition) making it a CA system.This limitation is less visible if number of candidates is small and table is local but serious if number of candidates (buckets) are huge and requires distribution. Even if partitions are assumed to be nil, there is always a tradeoff between consistency and latency i.e a distributed system has to compromise on speed for greater consistency. This has been formalized as extension of CAP Theorem - PACELC theorem.

Example:
If profiles are represented as TextGraphs as in previous implementation of HR Analytics, edges between vertices of the textgraph encodes the temporal information because most social profiles contain chronologically ordered timeline history of events - similarity measure for any two profiles could be the temporal cause-effect similarity between two profiles and thus Social Profile TextGraph is an EventNet too.

References:

615.1 PACELC Theorem extension of CAP - [Daniel Abadi] - http://cs-www.cs.yale.edu/homes/dna/papers/abadi-pacelc.pdf

---
616. (FEATURE) Software Analytics - Cyclomatic Complexity - Rewrite for merging two clauses - 14 November 2018
---
1. CyclomaticComplexitySparkMapReducer.py has been changed to merge True and False clauses for GraphFrames
2. Cyclomatic Complexity (Zeroth Betti Number) has been computed from GraphFrames stronglyConnectedComponents()
as per definition of Cyclomatic Number in Topological Graph Theory
3. Cyclomatic Complexity (First Betti Number) has been computed as E-V+[Zeroth Betti Number]
4. New callgrind and kcachegrind DOT files for execution of command "ls" have been committed and included for
Cyclomatic Complexity and Call graph GSpan mining:
        callgrind.ls.out.3104
        kcachegrind_callgraph_ls.dot
5. logs for this have been committed to CyclomaticComplexitySparkMapReducer.log.GraphXFrames.14November2018

---
617. (THEORY) Beatty Sequences and Complement Diophantines - related to all sections on Complementary Sets
,Ramsey 2-coloring and their Diophantine/Function representations - 16 November 2018, 18 November 2018 - draft updates to

https://arxiv.org/pdf/1106.4102v1

Complement Functions have been defined for arbitrary complementary sets (Integers/Rationals/Reals) in https://arxiv.org/pdf/1106.4102v1 and their relevance to Ramsey Coloring of Sequences and decidability of Diophantine representation of complementary sets (MRDP Theorem) have been expounded in detail earlier. Special Case of Complementary Sets of Integers is a well-studied problem and a classic result due to Beatty defines 2 complementary sets of integers based on solutions to equation 1/a + 1/b = 1 for irrationals a and b. Thus terminologies - Complement Functions, Complement Diophantines, Ramsey 2-Coloring of Sequences and Beatty Complementary Sequences - are synonymous with respect to set of Integers Z. Notion of Complementation generalizes it to any sets Boolean, Integer, Real or Rational. Integer and Real Complementation have solvable algorithms in the form of Diophantine equations subject to MRDP theorem and Tarski/Sturm, but complementation over rationals is open problem for the direction N to Q - mapping Q to N is straightforward.

ABC Conjecture and Complement Diophantines:

ABC conjecture states there are finite coprime triples a,b,c such that a+b=c and quality(q)=log(c)/log(rad(abc)) > 1 + epsilon for every epsilon > 0. Though there are infinitely many coprime triples a,b,c (a+b=c) having quality > 1, this conjecture predicts there are only finite number of coprime triples cluttered between every real value of epsilon. Complementary version of ABC conjecture is defined as:

(*) T = set of all coprime triples (a,b,a+b=c)

(*) Q1 = set of all coprime triples of quality > 1

(*) Q0 = set of all coprime triples of quality < 1

(*) Q0 U Q1 = T and Q0 /\ Q1 = Empty i.e sets Q0 and Q1 are complementary infinite sets (creates Beatty sequences equivalent for triples).

(*) Complementary Sets Version of ABC Conjecture: Infinite set of triples Q1 is the disjoint set cover or exact cover of finite sets of triples of quality 1+epsilon for every real epsilon.

(*) Previous complementary sets can be picturised as:

(*) T is a Rubik's cube in which each ordinate is a coprime triple (a,b,c)

(*) Q0 and Q1 partition T into two disjoint sets of qualities less than 1 and greater than 1.

(*) Q1 is the 3 dimensional half-space of Cube tiled by finite sets of triples of quality 1+epsilon for every real epsilon. As mentioned earlier, Pentominoes tiling is an exact cover problem for 2 dimensions.

References:

617.1 Beatty functions and Complementary Sets - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.514.2119&rep=rep1&type=pdf - "...Let U be a subset of Z. We call two non-empty sets A, B complementary with respect to U if A ∩ B = ∅ and A ∪ B = U. A classical result of 1926 due to S. Beatty [1] states that if α and β are positive irrational numbers with 1/α + 1/β = 1, then {floor(nα)} and {floor(nβ)} (n ∈ N) are complementary with respect to N, where floor(x) denotes the greatest integer function of real x. This result has since then been generalized by a number of authors, see e.g., [2], [4], [5], [6] and [7]. ..."

617.2 Diophantine Equations over Rationals and Reals - Section 3.1 - http://wwwmayr.in.tum.de/konferenzen/Jass07/courses/1/Sadovnikov/Sadovnikov_Paper.pdf - Solving in Q(multiplication by integers), Solving in R (Tarski and Sturm methods)

617.3 Three Dimensional Visualization of ABC Conjecture - http://www.lactamme.polytechnique.fr/images/CABC.21.20.1.M.D/display.html - visual intuition of the finite sets of coprime triples for every real epsilon > 0 for quality > 1 + epsilon.

617.4 Fraenkel Conjecture for exact cover decomposition of Z - https://core.ac.uk/download/pdf/82382652.pdf

617.5 Diophantines for 2D Pentominoes Tiling - https://did.mat.uni-bayreuth.de/~alfred/home/pentominoes.ps.gz - Locations of Tiles in 9*10 rectangle are specified by linear diophantines.
617.6 Kummer Theorem for Universal Diophantine Equation representation of arbitrary recursively enumerable set - Hilbert Tenth Problem and Two-way bridge between Number Theory and Computer Science - [Yuri Matiyasevich] - http://www.cs.auckland.ac.nz/~cristian/tcspi/ch9.ps - Diophantine Hierarchy - 9 Unknowns - Best known bound of number of unknowns (u) and degree (d) for Diophantine representation of an RE Set (in other words complement diophantine in the context of complementary sets) is (d,u) = (1.6 * 10^45, 9). Kummer theorem equates a binomial coefficient (a+b, b) to its exponential prime factorization 2^a2*3^a3*... where a2,a3,... are defined by Kummer Theorem: If integers a and b are written in p-adic (prime radix p) notation and are added the number of carries during this addition is given by ap. Because set of integers in binary notations can be expressed by binomial coefficients (a is number of 1s and b is number of 0s, a+b is the length of the binary integer string, (a+b,b) is the number of all possible integers having b 0s in binary representation), Kummer's theorem is widely used for exponential diophantine representation of arbitrary recursively enumerable sets. Word concatenations also have diophantine representation (related to 2.11 and 2.12 - Ramsey 255-coloring texts by alphabets) implying natural language texts can be represented by a diophantine equation which is helpful in text compression.
617.7 Further results on Hilbert Tenth Problem - [Zhi Wei Sun] - https://arxiv.org/pdf/1704.03504.pdf - Solving Diophantines in 11 unknowns over Z is undecidable.
617.8 Hilbert Tenth Problem for Rational Functions over Finite Fields is Undecidable - [Pheidas] - https://link.springer.com/article/10.1007/BF01239506 - related to previous problem of finite sets tile cover but finite fields require size of tiles to be power of prime (characteristic)
617.9 Decision method for elementary algebra and geometry - [Alfred Tarski] - Rand Corporation - https://www.rand.org/content/dam/rand/pubs/reports/2008/R109.pdf - decision procedure for real diophantines
617.10 Chaos Theory, Diophantines, Chaitin Theorem, Minimum Descriptive Complexity(Kolmogorov) - [Matiyasevich] - ftp://ftp.pdmi.ras.ru/pub/publicat/znsl/v377/p078.pdf - Computational Chaos Theory implies disorder in large sets arising from order (e.g.Logistic Map x(n+1) = kxn(xn-1)) - Chaitin theorem constructs a Diophantine set S which is chaotic and an exponential diophantine for it and its kolmogorov complexity for an initial finite fragment of S is O(log|S|).
617.11 Rayleigh Theorem or Beatty Theorem - https://en.wikipedia.org/wiki/Beatty_sequence - "...The Rayleigh theorem (also known as Beatty's theorem) states that given an irrational number r > 1, there exists s > 1 so that the Beatty sequences {B}_{r} and {B}_{s} partition the set of positive integers: each positive integer belongs to exactly one of the two sequences...." - Examples - Upper and Lower Wythoff Sequences [nr] and [ns] generated by Golden ratio r and s=r+1
617.12 Generalization of Beatty Theorem for Complementary Continuous functions over reals - https://web.archive.org/web/20140419091400/http://math.uncc.edu/sites/math.uncc.edu/files/2002_17_0.pdf - "...Main Theorem: Let F and G be real, continuous, strictly increasing, functions with domains [0,∞) satisfying F(0) = G(0) = 0 and $\lim_{x\to\infty}(F(x) + G(x)) = \infty$.For all t > 0, let Pt = {0,(F + G)−1(t),(F + G)−1(2t),(F + G)−1(3t), . . .}, and P+t = Pt\{0}. Then the two sequences At and Bt defined by At = {c(F−1)−1(n)b Pt:n ∈ N+t } and Bt = {b(G−1)−1(n)c Pt: n ∈ N+t } partition P+tfor all t > 0. Also, the elements of At are distinct and the elements of Bt are distinct ..."

---

618. (FEATURE) People Analytics - Experiential Intrinsic Merit - LinkedIn Profile Connections - 20 November 2018
- related to 443

1.SocialNetworkAnalysis_PeopleAnalytics.py has been changed to parse connections of a linkedin profile
2.Number of connections of a profile is then used to compute experiential intrinsic merit based on degree of
a vertex as:
    E = M*e^(kMt) = log(dv(t)) * e^(klog(dv(t))*t/clogt) / clogt for evolving degree dv(t)
3.This identity is derived from exponential relation between Fame (Degree) and Merit (Intrinsic fitness) of a
social profile - in https://www.microsoft.com/en-us/research/wp-content/uploads/2016/11/First-to-Market-is-not-Everything-an-Analysis-of-Preferential-Attachment-with-Fitness.pdf (described earlier)
4.Previous relation is anachronistic in the sense merit is defined in terms of fame when it has to be other way around. Function for experiential_intrinsic_merit() takes number of connections as a parameter and branches off into two clauses for least enery log normal merit if degree is 0 and degree based merit for degree > 0.
5.Example LinkedIn Profile Connections (committed in pdf and txt) have been analyzed for previous degree based
experiential merit and Recursive Lambda Function Growth merit. Example linkedin Connection pdf and text file have been committed to:
      ConnectionsLinkedIn_KSrinivasan.pdf
      ConnectionsLinkedIn_KSrinivasan.txt
6.Definition TextGraph of connections (.txt) extracts essence of the aura surrounding a social profile.
7.Dense subgraph (core) of this connections textgraph is a clustering coefficient measure i.e how well the neighbours of a vertex have semantic connections among themselves.
8.Though social profiles are prone to Sybil links for artificially bumping up popularity, previous analysis is
based on the assumption that humans do not link to another human unless they are trusted/befriended.
9.logs for this have been committed to:
      SocialNetworkAnalysis_PeopleAnalytics.log.20November2018 (degree experiential merit)
      SocialNetworkAnalysis_PeopleAnalytics.log2.20November2018 (recursive lambda function growth merit)
10.Recursive Lambda Function Growth merit logs have been truncated because of following recursion depth error in bintrees AVL Tree:
      RuntimeError: maximum recursion depth exceeded
but the logs show few erratic cycles having "missiles" which are inferred from random walk paths printed in the logs. Also Names of the connections have not been filtered which misleads the WordNet. Previous error impedes full analysis based on core number. But there are some relevant cycles related to "head hunter","Ph.D","dissertation","entrepreneur","engineer","managers","information_technology" etc., giving a glimpse of the social circle's nature.

---
619. (THEORY and FEATURE) Software Analytics - Cyclomatic Complexity - FTrace call graphs - 28 November 2018,12 April 2020
- related to 581,770

---
1. FTrace kernel function call graph shell script (asfer_ftrace.sh) already in USBmd and USBmd64 have been included as part of AsFer Cyclomatic Software Analytics for analyzing what happens in kernel within while executing a userspace code.
2. Example ftrace.log for Computational Geometric Factorization has been recorded in ftrace.DiscreteHyperbolicFactorization_TileSearch_Optimized.log
3. DOT graph creation function for parsing ftrace call graph log has been added in CyclomaticComplexitySparkMapReducer.py
4. FTrace call graph DOT file is written to

CyclomaticComplexitySparkMapReducer.ftrace_callgraph.dot
5. logs for Cyclomatic Analysis of kernel callgraph for previous factorization example have been committed to:
testlogs/CyclomaticComplexitySparkMapReducer.log.FTrace.28November2018

testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.factors

testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.ftrace.log.28November2018
6. PageRank, Degree Centrality, Cycles and Longest Path of Call graph are printed.
7. Least pageranked kernel functions are:
... ('unix_release\n', 0.0006824636787772093), ('nf_ct_get_tuple\n', 0.0006824636787772093), ('insert_work\n', 0.0006824636787772093), ('wake_up_worker\n', 0.0006824636787772093), ('__tcp_push_pending_frames\n', 0.0006824636787772093), ('bictcp_cong_avoid\n', 0.0006824636787772093), ('fsnotify_clear_marks_by_inode\n', 0.0006824636787772093), ('tcp_timewait_state_process\n', 0.0006824636787772093), ('ktime_get\n', 0.0006824636787772093), ('free_page_and_swap_cache\n', 0.0006824636787772093), ('update_page_reclaim_stat\n', 0.0006824636787772093), ('__cond_resched\n', 0.0006824636787772093), ('unlink_anon_vmas\n', 0.0006824636787772093), ('timer_interrupt\n', 0.0006824636787772093), ('inet_put_port\n', 0.0006824636787772093), ('update_process_times\n', 0.0006824636787772093), ('tcp_v4_send_ack\n', 0.0006824636787772093), ('ext4_get_inode_loc\n', 0.0006824636787772093), ('enable_8259A_irq\n', 0.0006824636787772093), ('__pagevec_free\n', 0.0006824636787772093), ('getnstimeofday\n', 0.0006824636787772093), ('dequeue_task\n', 0.0006824636787772093), ('__jbd2_journal_file_buffer\n', 0.0006824636787772093), ('fuse_prepare_release\n', 0.0006824636787772093), ('tcp_established_options\n', 0.0006824636787772093), ('fuse_release_common\n', 0.0006824636787772093)
...
which are top level invocations closest to userspace arising out of TCP traffic in Spark Executor Driver (TCP in kernel: http://vger.kernel.org/~davem/tcp_output.html)
8. It is worthy to note lot of ----exit() kernel functions related to VMA paging (e.g.do_exit()) which release virtual memory pages (possibly by GC) and TCP (tcp_write_xmit(), tcp_transmit_skb(), etc.,) having high degree centrality showing heavy TCP traffic.

Comparing call graphs by Graph Isomorphism or Graph Non-Isomorphism deduces code similarities. This is somewhat counterintuitive surprise to Program Equivalence Problem which is undecidable implying isomorphism detection is only an approximation. Program Equivalence essentially asks if two Turing machines accept same language (equivalent).

References:

619.1 Program Equivalence Problem is Undecidable - [Nancy Lynch] - Mapping Reduction to/from Halting Problem - https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-045j-automata-computability-and-complexity-spring-2011/lecture-notes/MIT6_045JS11_lec09.pdf - two Turing machines M1 and M2 are contrived and hypothetical Turing Machine M compares the M1 and M2. Reduction: M halts on x <=> <M1,M2> are not equivalent (EQ'). M1 always rejects while M2 accepts if M accepts x else rejects. There are two possibilities:
(*) M1 rejects and M2 accepts (because M accepts x => M1 != M2)
(*) M1 rejects and M2 rejects (because M rejects x => M1 == M2)
(*) But LHS is undecidable Mortality problem => M1 and M2 are not equivalent is undecidable
(*) Complement of the previous is hence undecidable
619.2 Call Graphs for Multilingual Software Analysis - [Anne Marie Bogar, Damian Lyons, David Baird] - https://arxiv.org/pdf/1808.01213.pdf - NeuronRain Call Graphs are dynamic and multilingual - KCachegrind for userspace C/C++/Java and SATURN/FTrace for kernelspace C code.
619.3 Game theoretic epidemic model of cybercrimes -

https://link.springer.com/chapter/10.1007/978-3-642-17197-0_9 - [Bensoussan A., Kantarcioglu M., Hoe S. (2010) A Game-Theoretical Approach for Finding Optimal Strategies in a Botnet Defense Model. In: Alpcan T., Butty=E1n L., Baras J.S. (eds) Decision and Game Theory for Security. GameSec 2010. Lecture Notes in Computer Science, vol 6442. Springer, Berlin, Heidelberg] - A Game-Theoretical Approach for Finding Optimal Strategies in a Botnet Defense Model - "...Botnets are networks of computers infected with malicious programs that allow cybercriminals/botnet herders to control the infected machines remotely without the user's knowledge. In many cases, botnet herders are motivated by economic incentives and try to significantly profit from illegal botnet activity while causing significant economic damage to society. To analyze the economic aspects of botnet activity and suggest feasible defensive strategies, we provide a comprehensive game theoretical framework that models the interaction between the botnet herder and the defender group (network/computer users). In our framework, a botnet herder's goal is to intensify his intrusion in a network of computers for pursuing economic profits whereas the defender group's goal is to defend botnet herder's intrusion. The percentage of infected computers in the network evolves according to a modified SIS (susceptible-infectious-susceptible) epidemic model. For a given level of network defense, we define the strategy of the botnet herder as the solution of a control problem and obtain the optimal strategy as a feedback on the rate of infection. In addition, using a differential game model, we obtain two possible closed-loop Nash equilibrium solutions. They depend on the effectiveness of available defense strategies and control/strategy switching thresholds, specified as rates of infection. The two equilibria are either (1) the defender group defends at maximum level while the botnet herder exerts an intermediate constant intensity attack effort or (2) the defender group applies an intermediate constant intensity defense effort while the botnet herder attacks at full power...." - FTrace analyzer in NeuronRain for kernel callgraphs provides a streaming delta-debugging feedback on rate of infection e.g delta changes (frequent call subgraphs in evolving SIS graph) in malicious patterns of infected kernels thus aiding a strategy.

---
620. (THEORY) Space Filling, Separate Chaining Hash Tables, Theoretical EVMs, Timeout, Balls-and-Bins, Linear
 Programming, Program Analysis, Cellular Automata, Majority Hardness Amplification Lemma - 30 November 2018, 1 December 2018, 2 December 2018, 28 January 2019 - related to 135, 517, 615

Space Filling has been defined in terms of Linear Program in https://sites.google.com/site/kuja27/Analysis%20of%20a%20Randomized%20Space%20Filling%20Algorithm%20and%20its%20Linear%20Program%20Formulation.pdf and its cellular automaton algorithm version has been described earlier. This linear program with constraint that every variable is non-zero:

x1 + x2 + ... + xn = N, for all xi > 0

is isomorphic to an LSH set partition, Integer Partition or Separate Chaining hash table partition of n parts of a set of N items. In most scenarios this is a restricted partition. Separate Chaining and Balls-Bins problems are equivalent if buckets are filled at random. Following are some of the probability estimates for Balls-and-Bins problem:

620.1 Average number of balls in a bin = n/b (n balls and b bins)

620.2 Number of balls to toss till a bin contains a ball:

probability of ball/item landing in a bin/bucket q=1/b, p=1-q

Number of tosses till success is Bernoulli trial geometric distribution

For success after kth ball toss, bernoulli probability: p^(k-1)q

Since k is a random variable, E(k) = Sum over k*p^(k-1)q = b

620.3 Number of balls tossed till every bin has a ball:

Total number of bins = b

Number of balls n is partitioned as n1 + n2 + n3 + ... + nk = N for each stage of tossing.

After ith stage, number of bins having atleast one ball = i-1

Number of remaining buckets = b-i+1

Probability of a toss landing in remaining buckets in ith stage = (b-i+1)/b

Each ni is a random variable of probability = b/(b-i+1)

Sum of E(ni) < O(blnb) before every bin has a ball.

Balls and Bins have histogram representation and previous bounds apply to histogram patterns too. In the context of theoretical EVMs if every voter votes perfectly at random with no prejudice to a candidate, number of voters required for every candidate to have atleast 1 vote = O(blnb) for b candidates. Course material in (https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt) describe a separate chaining hashtable timeout based OS scheduler pattern and similar bounds apply - Number of processes required for atleast one process per timeout value = O(blnb). Previous linear program version of separate chaining if solved by Balls-and-Bins has similar bounds - O(n*ln(n)) for solving LP of n variables.

To ensure secrecy of balloting, every vote appended to a candidate bucket in Theoretical Separate Chaining EVM has to be mapped to a unique one time randomly generated numeric id obfuscating the voter identity (and each bucket is encrypted). Mining patterns in this anonymous dictionary elicits only voting patterns and not voter patterns. Majority version of hardness amplification lemma described earlier envisages a MajorityInverse() function which inverts Majority+VoterSAT Boolean function composition for 2 candidates and obtains voters voting for a candidate and per voter SAT assignments top-down. Hardness of this MajorityInverse() is crucial to prove amplification. Theoretical EVM for 2 candidates is exactly separate chaining/LSH version of Majority gate of binary inputs 0 and 1 and Majority() and MajorityInverse() are defined as:

- Majority(): candidate of maximum sized voters bucket
- MajorityInverse(): voters bucket for a candidate

Intriguingly, MajorityInverse() is palpably easier in this separate chaining/LSH version of majority gate but previous anonymizer rules out easy inversion. Voter is issued a receipt containing the encrypted ciphertext of the candidate bucket index his/her vote is appended to. Encrypted receipt of vote issued to voter is more reliable than VVPAT which is in plaintext and not voter-received. Ballot secrecy is not compromised because this encryption is as secure as any other ciphertext transmitted over the network by public key infrastructure. Only authority accountable for conduct of voting can decrypt the receipt in case of disputes.

References:

620.4 Introduction to Algorithms - [Cormen-Leiserson-Rivest-Stein] - Coupon Collector Problem/Balls and Bins Problem - Page 134 (5.4.2), Page 1201 (C.4)

620.5 Analysis of Electronic Voting System - John Hopkins Paper - [TADAYOSHI KOHNO, ADAM STUBBLEFIELD, AVIEL D. RUBIN,DAN S. WALLACH] - https://homes.cs.washington.edu/~yoshi/papers/eVoting/vote.pdf - Analyzes C++ Source Code of Diebold AccuVote EVM and stresses VVPAT. Vulnerabilities include unencrypted ballot definition file tamper, Safety of Protective Counter storing number of votes polled, Party Affiliation tamper etc., - "...On a potentially much larger scale, if the voting terminals download the ballot definition over a network connection, then an adversary could tamper with the ballot definition file en-route from the back-end server to the voting terminal; of course, additional poll-worker procedures could be put in place to check the contents of the file after downloading, but we prefer a technological solution. With respect to modifying the file as it is sent over a network, we point out that the adversary need not be an election insider; the adversary could, for example, be someone working at the local ISP. If the adversary knows the structure of the ballot definition, then the adversary can intercept and modify the ballot definition while it is being transmitted. Even if the adversary does not know the precise structure of the ballot definition, many of the fields inside are easy to identify and change, including the candidates' names, which appear as plain ASCII text...". In the context of theoretical EVMs on cloud defined previously based on LSH/Separate chaining, these encryption prerequisites are

relevant in addition to CAP theorem limitations and PACELC theorem consistency versus speed trade-off. Per-candidate vote counters are replaced by hash table buckets which have to be secured. History preserving VVPAT audit trails require linking voters to votes which is obviously evident from per-candidate vote buckets. Frequent item set mining (e.g FPGrowth) of EVM buckets could point to malicious and suspicious voting patterns which is not feasible in counter increment.

---
621. (FEATURE) Social Network Analysis - People Analytics - Dictionary filter for names - 4 December 2018

1.RecursiveGlossOverlap_Classifier.py has been changed to define a new function nondictionaryword() for filtering nondictionary words. This is necessary for parsing text of Social Profiles which contain names in SocialNetworkAnalysis_PeopleAnalytics.py. Finding names in text is an open AI problem which comes under the purview of Named Entity Recognition (NER) which often requires CRF or Neural Network Training models.
2.As an alternative to statistical learning, if a word in text is not in a language Dictionary it is assumed to have exited any semantic network (WordNet for English etc.,) which could be names of people, places etc., But there are some exceptions to this and countably few names are in language dictionaries and encyclopedia.
3.Function nondictionaryword() looksup a word in PyDictionary which is based on WordNet, Google Translation, Thesaurus.com and returns true or false based on language dictionary. If for a word lookup null meaning is returned, it is outside Semantic Network and nondictionaryword() returns true.
4.Depth of the Recursive Gloss Overlap recursion has been parametrized in RecursiveGlossOverlap_Classifier.py and RecursiveLambdaFunctionGrowth.py and invoked in SocialNetworkAnalysis_PeopleAnalytics.py for depth 2.
5.WordNet and ConceptNet consist of network of concepts than textual words. PyDictionary has additional support for other datasources - Google Translation and Thesaurus - specific to language words.

---
622. (THEORY and FEATURE) Text (De)Compression by HMM on Vowelless texts - Prefix and Suffix Probabilities from English Dictionary - 6 December 2018

1.Python implementation for Decompressing vowelless compressed text TextCompression.py has been updated to invoke Prefixes and Suffixes probability priors computed from English Dictionary instead of hardcoded priors.
2.New source file WordSubstringProbabilities.py for computing word prefixes and suffixes probabilities by FreqDist has been committed which parses English Wordlist from Dictionary.txt into two probability distribution dictionaries for prefixes and suffixes.
3.Function wordprefixsuffixprobdist() in WordSubstringProbabilities.py called in TextCompression.py has been nominal because of computationally intensive Maximum Likelyhood Estimator in Hidden Markov Model (likelydict) and hardcoded priors are used.
4.Logs for this commit have been committed to WordSubstringProbabilities.log.6December2018 and TextCompression.log.6December2018.gz
5.compressedtext.txt and decompressedtext.txt have been updated
6.MLE likelydict is O(prefixes*suffixes) = O(number_of_words^2*length_of_longest_word^2) is costly one time disk read and might require some memcache-ing during initialization.

623.(THEORY) Arithmetic Circuit Complexity, Diophantine Complexity, Universal

Diophantine Equation, ABC
Conjecture, Complementary Sets, Diophantine Representation of Complementary Sets, Proof Complexity - 7 December 2018, 8 December 2018 - related to 617

Arithmetic Circuits are directed acyclic graphs of operands +(sum),*(product) as internal circuit elements (gates) and variable or numeric constant inputs as leaves. There is one output gate at root. Arithmetic circuits formalize computation of polynomials over a coefficient field. Some prominent upper bound (Big-O) results in arithmetic complexity include Matrix Multiplication and lower bound (Omega) results include size of arithmetic cicruit computing exponential polynomials. It has to be noted that solving Diophantine equations and computing polynomials are two opposite inverse directions - solving diophantines involves searching for a solution space while computing polynomials is circuit value problem and involves substituting solution values in arithmetic circuit input variables and evaluating the root gate. Algebraic Complexity classes VP and VNP define the arithmetic hierarchy of polynomials - VP contains polynomials which have polynomial size arithmetic circuits while VNP has polynomials such that coefficient of every monomial in it can be found efficiently. Permanent of a matrix has been shown to be complete for the class VNP by [Valiant]. If Permanent has polynomial size circuits, VP=VNP. From MRDP theorem, every recursively enumerable set is known to have a diophantine equation representation. There exists a universal diophantine equation which represents every recursively enumerable set ([Jones] theorem). Present state-of-the-art is universal diophantine of 9 unknowns of huge degree 1.6 * 10^45. It is open if Diophantine complexity class D ([Adleman-Manders]) is in NP (or D in VNP in algebraic complexity because P in NP implies VP in VNP?). Complementary Sets (Beatty Sequences) and their relevance to ABC Conjecture have been described earlier. By universal diophantine equation of 9 unknowns but huge degree, every subset of quality 1+epsilon for every real epsilon > 0 predicted by ABC conjecture can be represented. Complexity of a Diophantine Equation is also expressed by proof complexity notion of number of additions, multiplications required for verifying a solution. For universal diophantine equation, current known proof complexity bound is 100.

References:

623.1 Arithmetic Circuit Complexity - https://en.wikipedia.org/wiki/Arithmetic_circuit_complexity
623.2 Circuit Complexity - [Heribert Vollmer] - Chapter 5 - Arithmetic Circuits and Straightline Programs - https://books.google.co.in/books?id=qOepCAAAQBAJ&pg=PA172&lpg=PA172&dq=arithmetic+circuits+heribert+vollmer&source=bl&ots=KiFHOVH3kh&sig=Q5PaW5rJEHI8LBU0iYfNSrRXrwk&hl=en&sa=X&ved=2ahUKEwivu8GZo43fAhXLTn0KHek8BWI4ChDoATAJegQIABAB#v=onepage&q=arithmetic%20circuits%20heribert%20vollmer&f=false
623.3 Undecidable Diophantine Equations - [James P Jones] - https://pdfs.semanticscholar.org/7f96/11e582cf4a2efb1155c2fde9c891c6fc7be3.pdf - Universal Diophantine Equation for representing every recursively enumerable set which applies Kummer's Theorem for factoring Binomial Coefficients.
623.4 Arithmetization of 3SAT - Randomized Algorithms - [Rajeev Motwani - Prabhakar Raghavan] - Page 177 (Section 7.7) - 3SAT can be arithmetized to a multilinear diophantine polynomial by replacing each clause of the form (x1 V x2 V x3) to (1-(1-x1)(1-x2)(1-x3)) which can be solved for integer or real solutions. Number of unknowns in diophantine equals number of variables of 3SAT. This reduction implies diophantine class D is in NP. Real solutions to this SAT diophantine (e.g by Tarski method) is another kind of relaxation similar to least squares approximate MAXSAT Solver. But an added advantage is there exists a universal diophantine equation in 9 unknowns for any 3SAT diophantine though number of variables in 3SAT could be unlimited.

---

624. (THEORY) Shell Turing Machines, MRDP Theorem, Category Theory, Diophantine Equations, Turing Degrees, Embedding Formal Languages in Vector Space - 10

December 2018, 21 August 2019 - related to 538

Intuition for Shell Turing Machines defined in https://5d99cf42-a-62cb3a1a-s-sites.googlegroups.com/site/kuja27/UndecidabilityOfFewNonTrivialQuestions.pdf has been explained in NeuronRain FAQ https://neuronrain-documentation.readthedocs.io/en/latest/ . Dimension of a Shell Turing Machine parallels Turing Degree of a Turing Machine which is a measure of unsolvability of a set. Every Shell Turing Machine is defined on a vector space and all Shell Turing machines of similar dimension create an equivalence class. Partial Order Semi-Lattice is induced by this vector space dimensional hierarchy of Shell Turing Machines. Following example Shell Turing Machines defined over two different vector spaces of dimensions d1 and d2 motivate solvability hierarchy:

Turing Machines Td1 and Td2 find the lexicographically longest tuple (word) in languages L(Td1) and L(Td2). Tuples in L(Td1) and L(Td2) are of the form [x1,x2,x3,...,xd1] and [x1,x2,x3,...,xd2] respectively. Though both Td1 and Td2 are computationally similar, words in L(Td1) and L(Td2) belong to two different vector spaces of dimensions d1 and d2 therefore returning different results of lengths d1 and d2. This implies Turing degrees of Td1 and Td2 are different and the two machines solve varied difficulties.

In this sense, Shell Turing Machines generalize the notion of word embeddings of natural languages (e.g Word2Vec) in a vector space to any formal language - Regular, Context-Free, Context-Sensitive or Natural, Recursive, Recursively Enumerable - and formal languages become algebraic sets (similarity to VP and VNP is intriguing). MRDP Theorem implies Goedel's First Incompleteness Theorem. MRDP Theorem implies there are recursively enumerable sets having Diophantine representation but not recursive (there are no halting accept/reject Turing machines and therefore H10 is undecidable for set of integers). For Shell Turing Machines, MRDP theorem applies too but there is an additional fine structure (Dimension hiearchy partial order) introduced by vector space dimensions similar to Turing Degrees. Another relevant question arises: How Shell Turing Machines map to Diophantine representations i.e What is the relation between (number of unknowns, degree) of a diophantine equation of an RE set to the vector space dimension of Shell Turing Machine accepting it.

Example embedding of natural language text sentence in a Shell Turing Machine defined over vector space:

Sentence: "This is bigger than the biggest" is an oxymoron if read flat with no dimensional depth. Assuming
there exists set of Shell Turing Machines defined on a vector space of dimension d+x having oracle access to
Shell Turing Machine defined on a vector space of dimension d (Turing Jump), following dimension subscripted sentence is an example sentence in a Shell Turing Machine of dimension d+x having oracle access to lower dimensional (d) Shell Turing Machine and transcends two vector spaces of varied dimensions:

This (d+x) is bigger (d+x) than the biggest(d)

Vector Space for a Shell Turing Machine can be any arbitrary topological space in real analysis e.g Baire Space, Cantor Space (sets homeomorphic to Cantor Set Fractal obtained by removing midsegment of a line indefinitely)

Set of Recursively Enumerable Languages can be written as disjoint union/exact cover of recursively enumerable languages of Shell Turing Machines embedded in vector spaces of all possible dimensions:

RE = RE(d1) U RE(d2) U RE(d3) ... U RE(dn)

Since each dimension corresponds to a Turing Degree (measure of unsolvability), Partial Order Semi Lattice of all Shell Turing Machines has a Lowest Upper Bound dimension - There exists a maximal dimension Turing Machine dm accepting some RE(dm) language. Shell Turing Machines are most befitting in formalizing Computational Physics problems e.g Quantum and Relativistic Physics, String Theory/M-Theory, Branes, Multiverses.

An adaptation of Halting Problem to Shell Turing Machines

Undecidability of Halting Problem is proved by following gadget (oversimplified):
Turing Machine M' accepts an encoding of another Turing Machine M and an input x - M'(M,x).

M' rejects and halts if M accepts and halts on x
M' accepts and halts if M rejects and halts on x

Turing Machine M'' accepts if M' rejects and rejects if M' accepts which detects M has halted either way - M''(M'(M,x))

Contradiction is established by changing the Turing machine encoding to M' from M as M'(M',x):

M' rejects and halts if M' accepts and halts on x
M' accepts and halts if M' rejects and halts on x

which is crucial to undecidability of halting. Previous encoding has no dimensional information. If M' and M are
Shell Turing Machines of varied dimensions d1 and d2, Halting Problem is rephrased as - M''(M'[d1](M[d2],x[d2])) - Turing Machine M' defined in space of dimension d1 accepts encoding of Turing Machine M and an input x defined in space of dimension d2 (d1 > d2)

Following diagonalization proof of halting problem undecidability extends to Shell Turing Machines:

Let f(x,i) be a function defined as:
f(x,i) = 1 if Turing Machine x halts on input i, 0 if Turing Machine x does not halt on input i.
Let g(y) be another function defined as:
g(y) = 0 if f(y,y)=0, else undefined (if f(y,y)=1)

Diagonalization is an infinite two dimensional matrix D defined as:
D(x,y) = 1 if Turing Machine x halts on input y, 0 if Turing Machine x does not halt on y
Proof is by showing function f(x,i) is partial and not total (not defined for all inputs) and thus not recursive but recursively enumerable.

Nothing prevents asking what is g(g):
g(g) = if f(g,g)=0 then 0, else undefined (if f(g,g) = 1)

=> f(g,g) = 1 if Machine g halts on encoding of g, 0 if Machine g does not halt on encoding of g (Diagonal of the Diagonalization Matrix D)

But from previous definition of g(g):
g(g) = if f(g,g)=0 (if Machine g does not halt on g) then g(g) halts returning 0 and
if f(g,g)=1 (if Machine g halts on g) then g(g) is undefined
which is a contradiction.

The fact that there exists atleast one Turing Machine g whose halting is not decidable makes f(x,i) partial and not a total recursive function and therefore Halting problem is undecidable. It has to be noted that function g has oracle access to Halting problem function f(x,i) thus defining a Turing jump (0'). But Recursively Enumerable set has been previously expressed as disjoint union of languages of Shell Turing Machines defined on all possible vector spaces. This implies there are Shell Turing Machines which are not recursive (but represented by a Diophantine equation). Implications of this for problems in Quantum computation and Relativistic Physics defined for example on Hilbert Spaces are huge - There are Shell Turing Machines defined on certain vector spaces which are not recursive (or) there are no physics experiments for verifying truths of physical realities represented by these Hilbert Machines.

Problem of relevance to Shell Turing Machines is: What happens to halting problem if functions f and g are defined on two different vector spaces of unequal dimensions. If f(x,i) is embedded on a Hilbert space S1 which contains all other vector spaces and g(y) on another Hilbert space S2 (S1 contains S2), truth value of a logical statement in two quantifiers:

For all Shell Turing Machines x defined in Hilbert space S(d) of dimension d, there exists input i defined in S(d) such that x halts on i (True or False)

has to be exported to f(x,i) defined on Hilbert space S1 containing S(d)

This lifting/export of logical statements between Hilbert Spaces of 2 different dimensions is accomplished by Linear Transformations between Hilbert Spaces. Section 4.2 of Reference 624.5 describes some example definitions of logical formulas (conjunctive normal form) in Hilbert Spaces. Conjunctive Clauses of the formulas are represented as vectors of a Hilbert Space. It is not necessary that all transformations are possible. Computational Physics example: Assuming logical statements inside a blackhole singularity are defined in some Hilbert Space, existence of all possible linear transformations implies information (truths of logical formulas) oozes out of the blackhole space into observable space (e.g Hawking Radiation, Blackholes are not Black, Blackhole information paradox)

Traditional results on homomorphisms between Vector Spaces apply to Shell Turing Machines defined on any two vector spaces one of which is a subspace of the other:

(*) If T is a homomorphism defined from vectorspace U to vectorspace V of kernel W, then V is isomorphic to U/W. Conversely, if U is a vectorspace and W is a subspace of U, then there is a homomorphism of U onto U/W.

(*) If U is a vectorspace defined over field F, and W is a subspace of V then V/W is a quotient vectorspace over F if for v1, v2 in V and v1 + W in V/W, v2 + W in V/W:

(v1 + W) + (v2 + W) = (v1+v2) + W

(and)

For a in F, a(v1 + W) = av1 + W

If Shell Turing Machines S1(U) and S2(W) are defined over vector spaces V and W and W is subspace of V and there is a homomorphism T(defined by relation "bigger than") from V to V/W, previous example oxymoron sentence - "This is bigger than the biggest" - is logically defined in quotient space V/W in first order logic quantifiers as (annotated by square parentheses]:

This is [There exists v1 in V] bigger than [T(v1,identity + w1):v1 > identity + w1] the biggest [there exists w1 in W such that for all w in W, w1 > w]

Shell Turing Machines can be defined in terms of Category Theory too:

Category Theory of [Eilenberg-MacLane] abstracts whole of mathematics into following:

(*) Category C is a Set of Objects associated with a topological space

(*) Every Category has set of morphisms f:a-b defined on objects a,b in Category C - morphisms are maps between objects on topological spaces - every morphism f:a-b on objects a,b in topological spaces can be a Shell Turing Machine defined on topological spaces of a and b

(*) Functors F are maps between two Categories C1 and C2 - for each object x in C1, F(x) is in C2 and for each morphism f:a-b in C1 morphism, F(f):F(a)-F(b) is in C2 - every functor F can be a Shell Turing Machine defined on topological spaces of F(a) and F(b)

References:

624.1 Turing Jump, Priority Method of Emil Post, Injury - https://en.wikipedia.org/wiki/Turing_degree#Post's_problem_and_the_priority_method - Turing jump of a Turing Machine X is the set X' of Turing machines which

halt by having oracle access to X or in other words, X' is set of harder problems than X. Priority method is applied to prove existence or otherwise of intermediate degrees between two turing machines of different jumps e.g Turing Degree 0 (halting problem) and Turing Degree 0' (problems having oracle access to halting problem). Priority method tabulates requirements Ae and Be for an oracle machine e where Ae stipulates e does not compute 0' from X and Be implies non-oracle machine e does not compute X. Dimensional Hierarchy of unsolvability follows from Turing Jump.

624.2 Embedding Turing Machines on Hilbert Space - https://mathoverflow.net/questions/313343/embedding-turing-machine - Counts Functions (for languages of Turing Machines) by number of states and searches for the Turing Machines for these languages - This is the opposite direction of Shell Turing Machine definition which has an intrinsic dimensional parameter in addition to state.

624.3 Definability of Turing Jump - [Shore-Slaman] - https://math.berkeley.edu/~slaman/papers/jump.pdf - relation between Turing Machines A and B, A <T B implies A is recursive in B or A is Turing-computable by oracle access to B. A =T B (A <T B and B <T A) is the Turing Degree, an equivalence class of machines solving sets of similar difficulties. Turing Jump A'={e| Machine e halts on oracle access to A}

624.4 Aspects of Turing Jump - [Slaman] - https://math.berkeley.edu/~slaman/papers/lc2000.pdf

624.5 Hilbert Machines - Turing Machines in Complex Inner Product Spaces and Applications to Quantum Computation - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.73&rep=rep1&type=pdf - has close resemblance to definition of Shell Turing Machines. Linear Machines are conventional Turing Machines defined over input vector space, output vector space and a state vector space. Concrete Hilbert Machines are Linear Machines defined over a Hilbert Complex Vector Space and a linear Complex Operator - Theorem 8 - Any finite automata can be represented in a finite dimensional Hilbert Space by a Hilbert Machine - Examples of Hilbert Machines - Quantum Computation: Input and Output to a Quantum Computer are represented by vectors on Hilbert Space and computation (state transition) is performed by Unitary Operator H on Hilbert Spaces (Operator HH*=H*H=I, H* is infinite dimensional conjugate transpose of H) - Expected Value of State(at time t) = e^(-iHt). Shell Turing Machines go beyond just embedding a Turing Machine in vector space - As the name "Shell" suggests inspired by Unix shells, truths of logical statements in lower dimensional vector spaces are allowed to be exported to higher dimensions.

624.6 The Elegant Universe: Superstrings, Hidden Dimensions, and the Quest for the Ultimate Theory - [Brian Greene]

624.7 Turing Degrees of Diophantine Sets - Computably Enumerable Turing Degrees - Hilbert Tenth Problem - Sixth Asian Logic Conference - https://books.google.co.in/books?id=7IbVCgAAQBAJ&pg=PA152&lpg=PA152&dq=diophantine+equations+turing+degree&source=bl&ots=q9HomH3o-9&sig=rHKb81u6mifJvOX_-kjKv4YRdUY&hl=en&sa=X&ved=2ahUKEwjMpLvsur3fAhVNeisKHd_MA-U4ChDoATABegQIBRAB#v=onepage&q=diophantine%20equations%20turing%20degree&f=false

624.8 Degrees of Unsolvability: Tutorial - https://pdfs.semanticscholar.org/acae/fe1907be68d357700f0690519251fc948fdb.pdf - Mass Problems, Muchnik Degrees, Turing Degrees

624.9 Linear Transformations in Hilbert Space - [Stone] - https://www.ams.org/journals/bull/1934-40-11/S0002-9904-1934-05973-1/S0002-9904-1934-05973-1.pdf - Transform points on one Hilbert Space to points on another Hilbert Space

624.10 Geometry of Interaction and Linear Logic - [Girard] - https://en.wikipedia.org/wiki/Geometry_of_interaction - Proofs are represented as Networks of Logical Statement implications than trees in Sequent Calculus. This has some resemblance to Implication Graph Convex Hulls defined in http://sourceforge.net/projects/acadpdrafts/files/ImplicationGraphsPGoodEquationAndPNotEqualToNPQuestion_excerpts.pdf/download, https://sites.google.com/site/kuja27/NotesOnConceptHypergraphHMM_and_ImplicationGraphConvexHulls_2013-12-30.pdf?attredirects=0&d=1, https://sites.google.com/site/kuja27/ImplicationRandomGraphConvexHullsAndPerfect

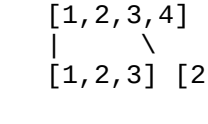
VoterProblem_2014-01-11.pdf?attredirects=0&d=1
624.11 Searle's Chinese Room Argument, Turing Test and AI - https://en.wikipedia.org/wiki/Chinese_room - Turing test differentiates a human from a machine by question-answering (e.g reCAPTCHA). If there is a machine translation turing machine translating English to Chinese, it does not necessarily imply machine understands English and Chinese(strong AI), but only simulates it (weak AI) - Human equivalent of a Turing machine in a room can exactly mimick Turing Machine by executing the translation algorithm churning out Chinese from English with no knowledge of English or Chinese whatsoever. Chinese room example demarcates measure of difficulties of two sets - understandability/consciousness and simulatability - and thus a Turing jump - human has oracle access to a translation turing machine.
624.12 Definability of Logical Formulas - http://www.math.wisc.edu/~msoskova/talks/MadisonColloquium.pdf
624.13 Topics in Algebra - [Israel N.Herstein] - Chapter 4 - Vector Spaces and Modules - Page 174 - Theorem 4.1.1 and Lemma 4.1.2 - Quotient Spaces
624.14 Reproducing Kernel Hilbert Space - https://en.wikipedia.org/wiki/Reproducing_kernel_Hilbert_space - Mathematical Functions are embedded on Hilbert Space and have distance metric defined on them. Every Function can be defined by a Turing Machine from Church-Turing Thesis.

---
625. (THEORY) Recursive Lambda Function Growth - TextGraph Machine Translation - 10 December 2018 - related to 178
---
1. RecursiveLambdaFunctionGrowth python implementation has a new function to do machine translation of Text Graph english word vertices to another natural language. It uses PyDictionary Google Translate REST API service internally.
2. An example Text Graph from Recursive Lambda Function Growth has been translated from English to Kannada (an Indian language).
3. This is just a primitive implementation without sentences creation which can be done through usual graph traversal or random walks.
4. Logs have been committed to testlogs/RecursiveLambdaFunctionGrowth.log.MachineTranslation.10December2018

---
626. (THEORY) Partial Ordered Rankings, Intrinsic Merit Vectors, Galois Connections and ThoughtNet - related to all sections expanding https://tac.nist.gov//publications/2010/participant.papers/CMI_IIT.proceedings.pdf and https://arxiv.org/abs/1006.4458 and ThoughtNet - 21 December 2018

Search Engines usually rank results as linear total order. Alternatively for each query ranked set of results can be a partial order and not necessarily total. This has been mentioned as a passing reference in ArXiv and NIST TAC articles previously. Traditionally Intrinsic Merit/Fitness is a scalar quantity which can be vectorized based on feature dimensions. Consequently, two URLs may not be comparable by a total ordering < relation. Following example intrinsic merit vector partial ordered rankings of websites intuit this:

(1) Subset partial ordered ranking - set of subsets ordered by containment:



```
      |      |
      [1,2] [2,3]
```

(2) Less Than or Equal To Ranking - Each tuple has elements of value less than or equal to elements of predecessor tuple:

```
      [1,2,3,4]   [2,3,2,5]
      |                 |
      [1,1,2,3] ----- [2,2,3,4]
      |
      [1,1,1,2]
```

Search Results for any two different queries thus create two partial ordered tuple sets of rankings. Galois
connections define maps F and G between elements of two posets A and B subject to conditions:

```
      a in A, b in B
      F(a)=c in B, G(b)=d in A
      F(a) <= b if and only if a <= G(b)
```

Previous Galois Connections provide an alternative spectacle to perceive ThoughtNet - ThoughtNet as defined
elsewhere earlier is a hypergraph of sentence hyperedges on evocative hypervertices motivated by Psycho-and-Neurolinguistics. Search engine queries can be construed as evocatives returning results evoked by the query
and thus search engine results are hyperedges of a ThoughtNet (Internet itself as a ThoughtNet) and queries are evocative hypervertices - same search result URL can appear in more than two queries creating the hyperedge across these query hypervertices.Previous partial ordered rankings of search results bring an additional application of Galois connections amongst search result posets - if URLs a,b,c,d are such that a=c and b=d aforementioned Galois connections are equivalent to hyperedges between ranked search query result posets in ThoughtNet and evocative hypervertices are search queries => ThoughtNet is a giant transitive Galois Connection which is different from Contextual Multi Armed Bandit formulation of ThoughtNet earlier. Various results and theorems applicable to ThoughtNet Hypergraphs (and its stripped down version of Survival Index Timeout pattern) have been mentioned in:
https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/
AdvancedComputerScienceAndMachineLearning.txt

References:

626.1 Galois Connections - https://en.wikipedia.org/wiki/Galois_connection

---

627. (FEATURE) Vowelless Text (De)Compression - simplified likelydict MLE for HMM - 3 January 2019

---

1. TextCompression.py has been classified to new class VowellessText and all functions have been made its members.
2. One time Likelyhood Dictionary computation of prefixes,suffixes and substrings has been shifted to WordSubstringProbabilities.py - wordlikelydict() and wordprefixsuffixsubstringsprobdist()
3. All precomputed prefix and suffix priors are initialized in TextCompression.py __init__()
4. Costly loop in HiddenMarkovModel_MLE() of TextCompression.py has been done away with and instead wordlikelydict() has been changed to choose between costly exhaustive search and substringsdict list comprehension.
5. Hardcoded priors for HMM have been removed and replaced by previous likelihood computation.
6. Logs for this have been committed to testlogs/TextCompression.log.3January2019
7. likelydict is created on demand once per compressed word than earlier heavy

initialization for all possible likelihoods.
8. Regular Expressions are used in list comprehension for matching strings - "_" are treated as alphabets in lower or upper case.
9. For Exhaustive Search. following line is crucial which computes the conditional probability of a prefix for a suffix:
            likelydict[k3[:-1]+k3[len(k3)-1]+k4[1:]] = v3 * v4
10. The else clause, generalizes it and applies precomputed frequency probabilities of concatenated substrings circumventing the following conditional probability:
            likelydict[substring1 + substring2 + ... + substringN] = Pr(substring1) * Pr(substring2) * ... * Pr(substringN)
by:
            likelydict[substring1 + substring2 + ... + substringN] = Pr(substring1 + substring2 + ... + substringN)
11. Both should be almost equal if substring random variables are independent and identically distributed.
12. An example English text from https://www.valmikiramayan.net/utf8/yuddha/sarga128/yuddha_128_frame.htm - Verse 120 has been compressed and decompressed.
13. Decompressed text has almost more than 50% accuracy and most priors are equal misleading the decipherment.
14. Previous HMM based probabilistic decompression can be compared to List decoding Berlekamp-Welch implementation which maps text to a polynomial

---
628. (FEATURE) Software Analytics - Hardcoded inputlayer replaced by Psutils - 8 January 2019 - Usecase
in 399 implemented

1.DeepLearning_SoftwareAnalytics.py in software_analytics/ has been updated and hardcoded process statistics
for input layer of BackPropagation have been replaced by Psutil systemwide load percentages
2.This implements the usecase for software analytics mentioned earlier and finds the weights for
input layer, hidden layer and output layer of BackPropagation multilayered perceptron.
3.Psutil provides cpu_percentage(), memory_percentage() and disk_usage() functions fields from which have been used as input layer of the perceptron.
4.Sampling loop has been limited to 2 but it can be any number. For each value of instantaneous system load (cpu, memory, disk IO) backpropagation iterates for sometime (limited to 10000) and finds the weights.
5.For any two successive load values of input layer at two time points t and t+delta, the weights of the perceptron fluctuate and should theoretically stabilize for significant number of load samples (which is presently 2)
6.Output layer has been set to a high value of 99% implying heavily thrashed system and input percentage values are divided by 10000 instead of 100 for probabilities to prevent overflow errors (NaN and Inf).
7.Logs have been committed to testlogs/DeepLearning_SoftwareAnalytics.log.8January2019

---
629. (THEORY) Complement Diophantine Equations over Complex Numbers and Algebraic/Arithmetic Circuits for them, Blum-Shub-Smale Turing Machines, Thue and Siegel Equations, Elliptic Curves, Mordell Conjecture and Theorem, ABC Conjecture, Riemann Zeta Function,Chaos - 9 January 2019,12 April 2020 - related to 24,490,617

As mentioned in previous sections, finding Diophantine representation of a complementary set reduces to creating a map f from diophantine set

a={a1,a2,...,an} such that f(x,b)=0 for unknown x in {Integers/Reals/Rationals/Complex} and parameter b in a. Discussions previously have been mostly restricted to Integers, Reals and Rationals and terminology "complement function" is applied to a diophantine of integer solutions to x and is totally defined for all x(recursive) throughout all sections of this draft. Extension to Complex Diophantine Sets involves applying a converse of Thue's Equation which exactly solves a known homogenous bivariate diophantine f:

f(x,y)=m, m is non-zero integer.

f(x,y) is factorized as:

f(x,y)=a(d)*product_r=1_to_d(x-alpha(r)*y)

where alpha(r) are complex roots of f(t,1)=a0+a1*t+a2*t^2+...a(d)*t^d=0.

But f is assumed to be unknown for a complementary diophantine set and has to be represented by reversing the factorization to an interpolation:

f(x,y)=a(d)*product_r=1_to_d(x-alpha(r)*y)

for all alpha(r) in a complex complementary diophantine set CDS={alpha1,alpha2,...,alphad} which yields f(x,y) for integer values of x,y. Assumption here is CDS is the set of complex roots of f(t,1)=a0+a1*t+a2*t^2+...a(d)*t^d=0.

Siegel Equation generalizes Thue's Theorem for arbitrary diphantine equations in integer coeffients. f(x,y) is decomposed as:

f(x,y) = h(x,y) + k(x,y)

where h(x,y) is homogenous and Thue's equation applies. Set of complex solutions of f(x,y) adjoined by set of distinct {alpha(r)} form a Riemann compact surface of singularities or an algebraic curve which is the required Complement Diophantine Surface over Complex Numbers. Mordell-Weil Theorem is a classic result which states diophantine elliptic curves of the form:

y^2 = x^3 + ax^2 + bx + c

have finitely many rational solutions which can be obtained by a chord-tangent algorithm and create an abelian group. Elliptic curve diophantines of finite rational solutions are suitable candidates for formalizing finite sets of triples in complement diophantine version of ABC conjecture mentioned in 617 - "...Q1 is the 3 dimensional half-space of Cube tiled by finite sets of triples of quality 1+epsilon for every real epsilon. As mentioned earlier, Pentominoes tiling is an exact cover problem for 2 dimensions...".

Algebraic or Arithmetic circuits for Diophantines over Complex Numbers can be defined by complexity classes over complex numbers (classes Pc, Npc,...). Blum-Shub-Smale Turing Machine model generalizes Turing Machines over Z[2] to any Field Real,Complex,Rationals etc.,For example, Hilbert Nullstellensatz problem decides if set of polynomials have common root and is in algebraic complexity class HNc over complex numbers. Riemann Zeta Function can be represented as a disjunctive clauses or non-uniform Hilbert Nullstellensatz algebraic circuit in 0-1-HNc (0 if s is a non-trivial zero else 1) of product polynomials having complex roots (non-trivial zeros):

RZF=product_of(1/[1-prime(k)^(-s)]) for all primes 2,3,5,7,...

Algebraic computation is defined by a binary tree - Algebraic Computation Tree over +,-.%,*,sqrt().From Koiran's Theorem 0-1-NPc (variables are binary but polynomial size certificates are complex) is in AM[2]=BP.NP if Riemann Hypothesis is True which equates conventional complexity classes over Z[2] and algebraic classes. 0-1-HNc is 0-1-NPc-complete and 0-1-HNc is in PSPACE. This is an alternative circuit formulation to Euler-Fourier polynomial mentioned in 24 for Riemann Zeta Function. Number of solutions (non-trivial zeros) to previous RZF circuit can be characterized by counting class equivalent Sharp-0-1-HNc (not in literature). From Riemann Hypothesis, all these solutions must have Re(s) = 0.5.

References:

629.1 Diophantine Equations - Survey - [V Srinivas, Tata Institute of Fundamental Research] - https://www.currentscience.ac.in/Downloads/article_id_059_12_0589_0594_0.pdf

629.2 Exploring Number Theory via Diophantine Equations -

https://www.csbsju.edu/Documents/Math/Sunil-CC-2009-B1.pdf - Gaussian Intgers, Pythagorean Triples
629.3 Fermat's Last Theorem - [Amir D.Aczel] - Shimura-Taniyama Conjecture: Elliptic Curves are Modular Forms and proof of Fermat's Last Theorem by [Andrew Wiles]
629.4 Algebraic Complexity - [Arora-Barak] - Blum-Shub-Smale Model - http://theory.cs.princeton.edu/complexity/algebraicchap.pdf
629.5 Extended Riemann Zeta Function - Theory and Problems of Complex Variables - [Murray Speigel] - Schaum Series - Page 273 - RZF(1-z) = 2^(1-z)*Pi^(-z)*Gamma(z)*cos(Pi*z/2)*RZF(z) which symmetrically defines RZF with zero knowledge of primes and thereby a composition of algebraic computation tree for RZF in Blum-Shub-Smale model over complex field.
629.6 Undecidability of Mandelbrot Set (Fractal Logistic map) and Decidability of its complement - https://users.cs.duke.edu/~reif/courses/complectures/books/AB/ABbook.pdf - [Arora-Barak] - 14.3.3 - Page 290 - "...Definition 14.23 (Mandelbrot set decision problem) Let PC(Z) = Z^2 + C. Then, the Mandelbrot set is defined as M = {C | the sequence PC(0), PC(PC(0)), PC(PC(PC(0))). . . is bounded }. Note that the complement of M is recognizable if we allow inequality constraints. ..." - has far reaching ramifications for problems in chaotic non-linear dynamics.

---

630. (THEORY and FEATURE) Music Pattern Mining - Audio to Notes - Intrinsic Merit of Music - 12 January 2019, 21 January 2019 - related to all sections on Intrinsic Merit and Fame - related to 67,68,69,593

---

1. Implementations in NeuronRain on Intrinsic Merit Versus Fame previously have so far concentrated only on Merit of Text (TextGraphs), Large Scale Visuals (Videos and Images - LSVR) and People (Interview/Examination/Contest based merit, Analytics of Social Network Profile Vertices). Merit of Audio or Complexity of Music has been defined theoretically earlier as a minimum size DFA or Turing Machine accepting musical notes as language which is a minimum description length measure similar to Kolmogorov complexity.
2.To this effect music_pattern_mining/AudioToBitMatrix.py has been changed to include a new function which converts an audio waveform to frequency domain by FFT. These frequencies are in turn mapped to a string of musical notes over octave alphabet - C,D,E,F,G,A,B,C (12 including fine divisions of notes denoted by #) - librosa API are invoked.
3.Essentially by mapping music to huge strings over octave alphabets the problem of music mining is made easier and all algorithms implemented for string mining in NeuronRain apply to music as well.
4.Since music is often related to creative genius, complexity of string of notes is an approximation of merit.
5.String of musical notes differ from usual text strings because of an inherent periodicity in music (self-similar fractal-like ascending and descending sequence of notes)
6.logs for an example audio to notes string has been committed to python-src/music_pattern_mining/testlogs/AudioToBitMatrix.log.AudioToNotes.12January2019
7.State machine diagrams for Minimum Turing Machine or DFA accepting languages of strings of octave notes
are directed graphs possibly having lots of cycles implying inherent periodicity in music. All directed graph complexity measures and algorithms therefore could quantify merit of audio or music.
8.Emphasis is laid more on music patterns than audio patterns because audio is simply some textual content extractable by speech recognition and spelt out in flat notes of similar frequency mostly without modulation excluding emotives.

---

631. (THEORY) Computational Geometric Factorization, Alternative Ray Shooting Algorithm, BQP and QNC, Pythagorean Triples, Possibility of decoherence from

Quantum NC to Classical PRAMs, No cloning theorem - related to 562 and all other sections on factorization - 21 January 2019

Pythagorean triples are integers of the form a^2 + b^2 = c^2 which can be factorized into product of
Gaussian Integers (Z[i]) as c = (a+ib)(a-ib) which is a hyperbolic curve defined on complex number field (C^2). This extends integer factorization to complex Gaussian Integers factorization of Pythagorean triples. Blum-Shub-Smale Turing Machines can factorize Pythagoren triples to Gaussian Integers. Ray Shooting Query Algorithms so far described previously depend on rays from origin (0,0) and piercing hyperbolic arc bow at different spacings. Instead of ray shooting from origin (0,0), end points of the hyperbolic arc can be connected by a chord (line between (0,N) and (N,0)) and midpoint of this chord is an alternative origin for ray shooting queries. This inverts the sweepline direction of rays from (0,0) to (N,N) to (N,N) to (0,0).

From QNC Computational Geometric Factorization described earlier, hyperbolic pixelated arc bow can be specified as a Hilbert Space vector superposition of factor/product states (ket notation) over complex amplitudes Bi:

$$|(p,q)> = B1|(x1,y1)> + B2|(x2,y2)> + ... + Bn|(xO(N),yO(N))>$$

and Bi^2 is the classical probability of a state |(xi,yi)>. QNC contains polynomial size quantum multiplication logdepth toffoli gates while NC has polynomial size logdepth PRAM processors and containment NC in QNC in BQP is known. Quoting reference 562.10 - "...If the HHL algorithm could be adapted to the case where (say) a single bi is 1 and all the others are 0, then it would contradict the impossibility of an exponential speedup for black-box quantum search, which was proved by Bennett, Bernstein, Brassard, and Vazirani...", if classical probability Bi^2 of some state |(xi,yi)> is 1:

$$Bi^2 = (a+ib)^2 = (a^2 - b^2) + i(2ab) = 1$$

implying b=0 and |a| = 1

Zero imaginary part in previous sceanario implies a possibility of decoherence from quantum toffoli gates in QNC to PRAMs in NC. No cloning theorem implies quantum state cannot be replicated. But since NC is contained in QNC, every PRAM algorithm in NC can be translated to a quantum algorithm in QNC. For example any two toffoli multiplication gates in QNC algorithm for two PRAMs in NC algorithm (e.g for previous factorization problem allowing decoherence), must have different state superposition vectors v1 and v2 such that v1 has some factor state amplitude Bi(v1) and v2 has other factor state amplitude Bk(v2) of classical probabilities 1: Bi(v1)^2 = Bk(v2)^2 = 1 which does not violate No cloning theorem.

Even if classical probability Bi^2 of some factor state Bi=|(xi,yi)> is not 1 but close to 1 in previous superposition of factor/product states, implying amplitudes of other factor states are non-zero, QNC factorization collapses to a classical BPNC factorization (e.g. a randomized PRAM factorization algorithm) despite no derandomization to NC. Derandomizing BPNC to NC has been mentioned previously in references 53.12.1 and 53.12.4 (Hitting Set Generators) which does not rule out possibility of a complete collapse of QNC all the way upto NC via BPNC.

References:

631.1 Wave Function Collapse and Various Interpretations of QM - https://en.wikipedia.org/wiki/Wave_function_collapse - Quantum Decoherence is one of the ways to interpret collapse of wave function to a classical state
631.2 Computational Meaning of Quantum Gates and Measurement Gates - https://www.fi.muni.cz/usr/gruska/quantum10f/qc1004.pdf - [Bernstein-Vazirani] Theorem implies for universal quantum gates it is sufficient to have real amplitudes
631.3 Measurement Principles of Quantum Gates - http://www-inst.eecs.berkeley.edu/~cs191/sp12/notes/chap1&2.pdf - Measurement by Phases
631.4 BQP is contained in PP - https://www.cs.cmu.edu/~odonnell/quantum15/lecture23.pdf - Bounded Error Quantum

Polynomial Time computation is contained in Classical Probabilistic Polynomial Time computation - Theorem 4.1 - [Adleman-DeMarrais-Huang]. But derandomizing PP to P or PP success amplification does not seem to have a parallel to that of BPP success amplification. As a special case, hypothetical PP factoring algorithm can be made bounded error by invoking ray shooting query oracles to find approximate factors which bound the failure probability and thus bringing factorization to BPP or BPNC (by parallel ray shooting) which can then be derandomized to NC.
631.5 BPP is contained in BQP and Feynman Path Integral Proof of BQP in PP - https://www.scottaaronson.com/democritus/lec10.html - classical coin flip is a quantum Hadamard Gate mapping qubit |0> to 1/sqrt(2)(|0> + |1>) and qubit |1> to 1/sqrt(2)(|0> - |1>). Hadamard Gate is a Universal Quantum Computation Gate. Combined together Bounded Error Quantum Computation lies between bounded classical 2-sided error and unbounded classical 2-sided error complexity classes
631.6 Quantum Hadamard Gate - https://en.wikipedia.org/wiki/Quantum_logic_gate#Hadamard_(H)_gate
631.7 Is Factoring really in BQP? really? - https://rjlipton.wordpress.com/2011/01/23/is-factoring-really-in-bqp-really/ - "...The problem is that if we press the button again at time {i+1}, we may get a bit that pertains to some other value {y_{i+1}} that is not the same as {y_i}. If we could guarantee that repeated taps on {B_1} would stay focused on {y_i} then we could apply search-to-decision and recover the function value {y_i}..."

---

632. (FEATURE and THEORY) Intrinsic Merit of Music, Exact Learning of DFAs, Minimum Description Length
Merit - 29 January 2019 - related to all sections on Intrinsic Merit and Fame - related to 67,68,69,593,
630

---

1. New function audio_merit() has been defined in AudioToBitMatrix.py which invokes minimum_descriptive_complexity() to compute MDL for string of octave notes for a musical clip.
2.Source files MinimumDescLength.py and Entropy.py have been changed to include __main__ and rewritten
to define member function minimum_descriptive_complexity() and some other error checks for floating point overflow.
3.An example musical MP4 clip excerpt of JS Bach has been analyzed. This masterpiece is featured in Douglas Hofstadter's Godel-Escher-Bach:An Eternal Golden Braid for its self-similar strange loops of notes (a mandelbrot fractal set)
4.This implementation thus is a culmination of 4 classes of intrinsic merits on Internet and WWW - Text, Video, Audio and People - defined in this draft.
5. Following are intrinsic merit implementations in this draft which expand on publications http://arxiv.org/abs/1006.4458, http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf and http://arxiv.org/abs/1106.4102 (Complement Diophantines are intrinsic merit measures too because text obtained by word concatenations can be represented by word diophantine equations and can be Ramsey 255-colored by alphabet locations thereby creating arithmetic progressions):

5.1.Textual merit is analyzed by Recursive Gloss Overlap and Recursive Lambda Function Growth algorithms - Graph Tensor Neuron Network Merit.

5.2 People Merit is analyzed by a mixture model of Intrinsic(Academic/Work/Interview/Contest/Examinations) and Experiential Mistake Bound Merit by a recursive mistake correction tree differential equation.

5.3 Video Merit is analyzed by EventNet/ImageNet Tensor Products Algorithm.

5.4 Audio Merit is defined theoretically as complexity of exact learning of DFAs on strings of notes and their minimization in [Dana Angluin] L* Algorithm Model and implemented by a less intensive Minimum Description Length merit on string of notes in this commit. Implementation of L* could be exponential and there are PAC learning approximate DFA learning algorithms.

6.Previous algorithms for 4 classes of intrinsic merit are theoretical approximations based on algorithmic graph theory, causality and logical time in clouds, learning theory and connectomes with no invocation of statistical tool and by no means the only way to analyze merit but one of them.

References:

632.1 Hilbert Tenth Problem: What was done and What is to be done - [Yuri Matiyasevich] - https://books.google.co.in/books?id=qWYbCAAAQBAJ&pg=PA11&lpg=PA11&dq=word+concatenation+diophantine&source=bl&ots=zQDO7Jy-kK&sig=ACfU3U1CeZ4WvshXDu62TEIOmb0bofrcbg&hl=en&sa=X&ved=2ahUKEwjItsyO5JLgAhUSfSsKHZ-NAx8Q6AEwAnoECAQQAQ#v=onepage&q=word%20concatenation%20diophantine&f=false - "...Under this representation of matrices, concatenation words corresponds to matrix multiplication and thus can be easily expressed by a system of diophantine equations ..."
632.2 The Musical Offering - JS Bach - https://en.wikipedia.org/wiki/The_Musical_Offering
632.3 Godel Escher Bach: An Eternal Golden Braid - [Hofstadter] - Endlessly rising canon - "...In this canon, Bach has given us our first example of the notion of Strange Loops. The
"Strange Loop" phenomenon occurs whenever, by moving upwards (or downwards) through the levels of some hierarchical system, we unexpectedly find ourselves right
back where we started..."
632.3 Exact Learning of DFA - Angluin's L* Algorithm - teacher/learner model of membership and equivalence queries - http://www.cs.ox.ac.uk/james.worrell/DFA-learning.pdf

---
633. (THEORY and FEATURE) GIS and Urban Planning Analytics, Convex Hull - 31 January 2019
---
1. ImageGraph_KerasTheano.py has been changed to include a new function for finding convex hull of set of
points/pixels in an image
2. QHull ConvexHull() function from SciPy Spatial library - https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html - has been invoked in the new convex_hull() function defined for this purpose.
3. An example demographic GIS satellite image of Chennai Megapolis (Population density of Urban Agglomeration) from SEDAC website - http://sedac.ciesin.columbia.edu/mapping/popest/gpw-v4/ - has been analyzed to draw a convex hull around thickly populated regions and convex hull vertices are printed in terms of some area units (not sqkm or sqmiles).
4. OpenCV2 provides api for convex hull too - https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/hull/hull.html
5. Convex Hull analytics of GIS imagery is helpful for Urban planners and sustainable development.

---
634. (FEATURE) Software Analytics and Scheduler Analytics - Backpropagation correction - 9 February 2019

1.DeepLearning_SchedulerAnalytics.py and DeepLearning_SoftwareAnalytics.py have been updated for
correction to a serious bug in BackPropagation computation.
2.Earlier, BackPropagation Neural Network (BPNN) object for Software and Scheduler Analytics was getting created for every input training dataset element

erroneously.
3.This has been remedied by instantiating BackPropagation class only once outside the loop and updating only the input layer and/or expected output layer for each element in input dataset iteration.
4.Iterations have been curtailed for concise logs.
5.By this change, weights are updated for every psutil per process proc object (Scheduler Analytics) or systemwide process statistics info (Software Analytics) thus learning weights of the multilayer perceptron from system performance.
6.logs for this have been committed to testlogs/DeepLearning_SchedulerAnalytics.log.9February2019 and testlogs/DeepLearning_SoftwareAnalytics.log.9January2019 (Timestamped wrongly - must be read DeepLearning_SoftwareAnalytics.log.9February2019)
7.After sufficiently large number of input training data, hopefully this BPNN would be able to predict outputs from input per-process and systemwide performance statistics data e.g 99% load average.

---
635. (THEORY and FEATURE) Intrinsic Merit of Texts, Ramsey Theory, Complement Diophantines - Ramsey number of text graph - related to 2,617,630,632 and all sections on Fame and Merit - 11 February 2019

1.It has been mentioned earlier that texts which are concatenations of words can be represented by
*) Matrix Diophantine Equations *) and (255 or 511) coloring of texts by ASCII/Unicode alphabets (excluding space) which give rise to ramsey ordering by Van Der Waerden Theorem assuming texts of size N are integer sequences of locations 1-to-N colored by alphabets containing monochromatic arithmetic progressions of alphabet(color) locations.
2.Best known upper bound for Van Der Waerden Number for a least length sequence of N integers = W(r,k) for r-coloring of an integer sequence and arithmetic progression of least length k <= 2^2^(r^2^2^(k+9)) by [Timothy Gowers]
3.Previous upperbound for texts colored by alphabets of size 255 implies size N of a natural language text to have length k arithmetic progressions on alphabet locations <= 2^2^(255^2^2^(k+9)) which is huge even for small k.
4.Ramsey coloring R(r,s) of graphs is equivalent of Van Der Waerden sequence coloring which defines emergence of monochromatic cliques r or independent sets s (also known as Friends and Strangers theorem).
5.By definition of complementation in generic sense, k-coloring partitions a set into k disjoint subsets which can be represented by some diophantine (e.g Beatty functions) - Complementation is a size 2 partition.
5.In this commit, Ramsey R2 function is invoked from NetworkX API on textgraphs of natural language texts and maxclique-maxindependentset pairs are computed.
7.This is a less intensive intrinsic merit measure for a text and Van Der Waerden number computation has no mention in python libraries presently and is daunting too - SAT solvers are used to compute Van Der Waerden Numbers in some algorithms.
8.Following this reasoning, strings of octave alphabets in music can be construed as 12-coloring (and more including majors and minors) of music as sequence and indicates inevitable emergence of order from chaos.

References:

635.1 SAT Solver for Van Der Waerden number 1132 = V(2,6) - https://www.cs.umd.edu/~gasarch/TOPICS/vdw/1132.pdf

---
636. (THEORY) Computational Geometric Factorization - Parallel Planar Point Location of factors in Monotone Subdivision formed by Hyperbolic Arc Pixel Polygons with no necessity for Ray Shooting Queries, Van Der Waerden Ramsey Coloring of Sequences, Computational Algebraic Geometry - related to 506,601,605

and other sections on factorization - 12 February 2019, 18 February 2019, 20 February 2019, 22 September 2019, 20 October 2020

In previous sections on computational geometric factorization, various parallel algorithms based on k-mergesort,segment tree, wavelet tree,sorting networks,planar point location and ray shooting were described to locate factors on the hyperbolic arc tile segments (or) pixel array polygons. A continuous hyperbolic curve is "rectified" to a chain of non-intersecting polygons (juxtaposed) by considering each tile segment (x, N/x, x+1, N/x) of hyperbola obtained by differential calculus identity delta=N/[x(x+1)] as a rectangular polygon of dimension 1 * N/[x(x+1)] which is an array of pixels of dimension 1 * 1 containing the product of x and y ordinates in arithmetic progression and thus implicitly sorted. These adjoining rectangular polygons which meet at endpoints, create a plain simple line graph formed by their vertices and edges which is strongly connected. This set of polygons partition the 2 dimensional plane of area (1,N)*(1,N) into 3 planar subdivisions - space above hyperbolic polygons, space within the hyperbolic polygons and space below hyperbolic polygons. Earlier algorithms assumed arbitrary subdivisions and had ray shooting to find a polygon containing approximate factor as a prerequisite which is then binary searched. But this chain of hyperbolic tile polygons is monotone because y-axis is strictly decreasing or non-increasing for any path along the plain simple line graph (PSLG) formed by hyperbolic polygonal planar subdivision. There are efficient algorithms for PRAM geometric searching of monotone subdivisions mentioned in references below. Thus locating a hyperbolic pixel array polygon containing factor point can be done in O(logN) PRAM time and finding the factor in this polygon by sequential binary search in additional O(logN) sequential time. Thus factorization by planar point location in monotone subdivision of hyperbolic pixel array polygons is in O(logN) parallel + O(logN) sequential time - and thus in Nick's class - without requirement of approximate factoring by ray shooting queries.

Arithmetic Progression of points as product of ordinates within each Hyperbolic pixel array polygon brings Ramsey/Van Der Waerden coloring of sequences into the realm of factorization - by k-mergesort the tile segement polygons are unified into single totally ordered integer sequence of length N*N/(N+1). If each arithmetic progression polygon is colored uniquely, the sorted sequence is a union of set of k-colored arithmetic progressions of some minimum length (or) N*N/(N+1) = VanDerWaerden(k, minimum length of a segment)

In all sections related to factorization in this draft, factoring only one integer at a time has been the focus. Geometric intuition of hyperbolic tile/polygon segmentation allows storing segments or polygons for hyperbolic curves of multiple integers in same datastructure which could be segment tree, wavelet tree, or point location tree. For example two rectified hyperbolic polygons/segments for xy=N1 and xy=N2 for factoring N1 and N2 can be stored in single search tree which can be queried for factor points of N1 and N2 independently. This optimization saves lot of parallel tree construction time and space.

References:

636.1 Parallel Planar Point Location for Monotone Subdivisions - https://www.csun.edu/~ctoth/Handbook/chap46.pdf - Table 46.5.1 - Parallel Point location in monotone subdivision by EREW PRAM in O(log n) time, O(n) parallel work - Reference [TV91] in 636.2

636.2 Parallel transitive closure and point location in planar structures - Lemma 4.8 - [Tamassia-Vitter] - http://www.ittc.ku.edu/~jsv/Papers/TaV91.transclosure.ps.gz

636.3 Computational Geometry - [Preparata-Shamos] - Point Location - Section 2.2 - https://books.google.co.in/books?id=_p3eBwAAQBAJ&printsec=frontcover#v=onepage&q=point%20location&f=false

636.4 Point Location in Successive Convex Hulls - [Greenlaw-Hoover-Ruzzo] - List of P-Complete Parallel RAM Problems - A.9.5 -

https://homes.cs.washington.edu/~ruzzo/papers/limits.pdf - Given a point on 2D Plane, is the point in k-th remaining convex hull after finding and removing convex hulls repetitively? - Previous monotone chain of hyperbolic pixel array polygons are successive convex hulls containing respective tile segment points
636.5 CREW Parallel RAM Version of Kirkpatrick's Planar Point Location - Chapter 5 - Some Parallel Geometric Algorithms - Section 3 - https://books.google.co.in/books?id=ThBR22HOjMcC&pg=PA85&lpg=PA85&dq=parallel+planar+point+location&source=bl&ots=jFBpEv60Ab&sig=ACfU3U1fffe1j5o-w6GWnZHKxAsW0iaU_A&hl=en&sa=X&ved=2ahUKEwik4uHr9cTgAhUDBIgKHRDlAwc4FBDoATAFegQICRAB#v=onepage&q=parallel%20planar%20point%20location&f=false - Kirkpatrick's decomposition triangulates a Planar Straight Line Graph (plain simple line graph or PSLG) e.g PSLG of Hyperbolic arc pixel array polygons previously described for factorization. CREW PRAM Version of this triangulation algorithm is O((logN)^k) parallel time and O(logN) sequential query time but requires O(N) processors.
636.6 Parallel Planar Point Location by Parallel Construction of Subdivision Hierarchies - [N. DADOUN AND D. G. KIRKPATRICK] -https://ac.els-cdn.com/0022000089900421/1-s2.0-0022000089900421-main.pdf?_tid=c9b3fe06-f5de-4038-a511-1e201f91edc8&acdnat=1550480942_c4eea646a5bb64a5444edc195ca6743d - "...It is worth noting that Atallah and Goodrich [AG] use the parallel plane sweep technique to perform planar point location with O(log n log log n) parallel preprocessing, O(n log n) space and O(log n) sequential query time. The subdivision hierarchy technique uses 0((log n)^2) parallel preprocessing (O(log n log* n) for convex subdivisions), O(n) space and O(log n) sequential query time. ..."
636.7 Parallel Planar Point Location by Parallel Construction of Plane Sweep Tree - [Atallah and Goodrich] - https://docs.lib.purdue.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1481&context=cstech - Theorem 3.12 and Table 1 - Summary of Results in the end - "...Given a planar subdivision S consisting of n edges, we can construct in parallel
a data structure which, once constructed, enables one processor to determine for any query point
p the face in S containing p in O(logn) time. The construction takes O(lognloglogn) time and
O(nlogn) space using O(n) processors on a CREW PRAM...." - It has to be observed that PSLG from Hyperbolic arc pixel array polygonal rectification is the simplest form of its kind containing O(N) rectangular faces and <= 4*N edges, 4 per rectangle.
636.8 List of Problems in NC and Drawbacks of NC Theory - http://pages.cs.wisc.edu/~tvrdik/3/html/Section3.html - Parallel sorting and most of Computational Geometric Problems are in NC - "...The class NC may include some algorithms which are not efficiently parallelizable. The most infamous example is parallel binary search...NC theory assumes situations where a huge machine (polynomial number of processors, e.g., millions) solves very quickly (polylogarithmic time, e.g., seconds) moderately sized problems (e.g., hundred of thousands input items). In practice, however, moderately sized machines (hundreds, at most thousands of processors) are used to solve large problems (e.g., millions of input items). So that the number of processors tends to be subpolynomial, even sublinear...." - This practical limitation of NC applies to computational geometric factorization by planar point location especially when huge integers e.g semiprimes of the order of 1024 bits or more in public key infrastructure have to be factorized. This requires huge number of processors (PRAMs) but yet theoretically highly parallelly feasible. Precision ray shooting queries optimization like tile-summation ray shooting sieves out lot of factorless segments of hyperbolic arc bow finding approximate factors which are in proximity to exact factors and with high probability in same polygon/segment facilitating binary search of arithmetic progression.If a ray shooting query followed by binary search fails to find exact factor, angle of the ray can be altered in either direction and shot again covering unswept areas of the plane. Ray shooting partitions the hyperbolic arc into O(N/loglogN) segments for O(loglogN) prime factor ray queries.
636.9 NC1 and L - Theorem 3.5 -

https://cse.buffalo.edu/~regan/papers/pdf/ALRch27.pdf - [Eric Allender,Rutgers University,Michael C. Loui,University of Illinois at Urbana-Champaign,Kenneth W. Regan,State University of New York at Buffalo] - O(logN) Parallel Planar Point Location time implies complexity class NC^1 which is contained in deterministic logspace class L. This is quite counterintuitive because despite prohibitively high number of PRAMs even if simulated by BSP, space required for computational geometric factorization is logarithmic.
636.10 Generalized Planar Point Location - [Chazelle-Sharir] - Collins Decidability Theorem (refinement of Tarski Decidability) based point location on arbitrary set of polynomials defined over euclidean plane - https://www.sciencedirect.com/science/article/pii/S074771710880065X/pdf?md5=1abc953f7893218fefbad2e9bb83d68c&pid=1-s2.0-S074771710880065X-main.pdf&_valck=1 - Factorization is limited to locating integer points on hyperbola xy-N=0
636.11 Union of Arithmetic Progressions - [Newman] - http://inis.jinr.ru/sl/M_Mathematics/MT_Number%20theory/Newman%20D.J.%20Analytic%20Number%20Theory%20(GTM%20177,%20Springer,1998)(ISBN%200387983082)(81s)_MT_.pdf - Page 14 - Every tile segment in rectified hyperbola (which is a monotone subdivision formed by juxtaposed line segments decreasing or non-increasing in one coordinate and thus is a plain simple straightline graph) is an arithmetic progression. By analytic number theory every tile segment arithmetic progression an + b can be written as a generating function z^b/(1 - z^a). It is natural to ask if union of tile segment arithmetic progressions can be written as single generating function. Answer is no by contradiction. Generating functions based arithmetic progression exact cover of set of natural numbers is a special case of complement diophantines and complementary equations - "...Thus the dissection into evens and odds corresponds to the identity #∞ n#0 z^n = #∞ n#0 z^2n + #∞ n#0 z*2n+1, and the dissection into 2n, 4n + 1, 4n + 3 corresponds to #∞ n#0 z^n = ∞ n#0 z^2n + #∞ n#0 z*4n+1 +#∞ n#0 z*4n+3, ..."
636.12 Van Heuraets Rectification of Curves - https://www.maa.org/press/periodicals/convergence/mathematical-treasures-van-heuraets-rectification-of-curves
636.13 Numerical Rectification of Curves - [B. P. Acharya, M. Acharya and S. B. Sahoo - ITER,S’O’A University, Bhubaneswar, India] - http://www.m-hikari.com/ams/ams-2014/ams-17-20-2014/acharyaAMS17-20-2014.pdf
636.14 Exact Rasterization of Algebraic Curves - [Stussak] - https://opendata.uni-halle.de/bitstream/1981185920/7892/1/Dissertation%20-%20Christian%20Stussak%20-%20On%20reliable%20visualization%20algorithms%20for%20real%20algebraic%20curves%20and%20surfaces.pdf - Figure 3.2 - Correct and wrong rasterizations of the curve y^2 + x^2*(x + 1) = 0 - Rasterization of an algebraic curve in Computer Graphics is the Computational Geometry equivalent of rectification - curve is approximated by pixels. Bresenham Line Rasterization is limited to approximating straightlines by pixels - "...we solve the task of rasterizing a real algebraic plane curve VR(F) defined by a polynomial F ∈ Z[x, y]. Although many algorithms exist for rendering such a curve, only a few of them guarantee the correctness of the output. The term correctness is often interpreted as topological correctness, i.e. the graph induced by the rendering is isotopic to VR(F)...." - "...Real algebraic space curves are visualized by determining a line strip approximation to their segments using a projection and lifting approach..." - Computational Geometric Factorization rasterizes hyperbola by axis-parallel line segments which are juxtaposed pixel-array polygons. Finding a polygon containing factor point can be done in polylog time and searching the polygon is logarithmic because of (arithmetic progression) implicit sortedness of pixels.
636.15 Graphics Computation in PRAMs - Parallel Rasterization - XMT-GPU - http://users.umiacs.umd.edu/~vishkin/XMT/ICPP08version.pdf - [Thomas DuBois University of Maryland College Park, MD tdubois@cs.umd.edu,Bryant Lee Carnegie Mellon University Pittsburgh, PA bryantl@cs.umd.edu,Yi Wang Virginia Polytechnic Institute Blacksburg, VA samywang@vt.edu, Marc Olano University of Maryland, Baltimore County Baltimore, MD olano@umbc.edu,Uzi Vishkin University of Maryland College Park, MD vishkin@umiacs.umd.edu] - Draft sections on Computational Geometric Factorization in this design document assume that total number of O(N) hyperbolic arc tile segments of endpoints (x, N/x, x+1, N/x) are axis-parallel rectified by differential calculus identity delta=N/[x(x+1)] which is length of

each segment and N/(logN)^k processors are work-optimally allocated (logN)^k segments each (from section 477 on Local Tile Search). Each of N/(logN)^k PRAM processors binary searches (logN)^k segments allocated to it in (logN)^k * logN = (logN)^(k+1) parallel time. Instead, Rasterizing hyperbolic arc by Parallel RAMs approximates hyperbola on pixel grid which may not be axis-parallel.
636.16 Rasterization of Conic Sections - Efficient Integer algorithms for drawing Ellipse,Hyperbola,Parabola - refinement of Bresenham,Pitteway,Kappel conic sections and line drawing algorithms - https://alexanderagathos.com/publications/CAG98.pdf - Section 4.1
636.17 Topology of Plane Algebraic Curves - https://who.paris.inria.fr/Elias.Tsigaridas/files//clpprt-socg-2009.pdf - any algebraic curve including hyperbola drawn on R^2 could be mapped to a planar graph G which is ambient isotopic to the curve - "...Even more precisely we want the curve C and the graph G to be ambient isotopic in the real plane. In other words, we require the existence of a continuous map F : R^2 ×[0, 1] −→ R^2 , such that Ft is an homeomorphism for any t ∈ [0, 1], F0 is the identity of R^2 and F1(C) = G ..." - In other words hyperbola is piece-meal approximated by set of non-axis parallel edges of a graph by cylindrical algebraic decomposition (CAD) or other algebraic geometry algorithms. Each edge of this isotopic graph could be independently and parallely rasterized by Bresenham line rasterization to obtain set of pixel array polygons which could be located for factor points in polylog time.
636.18 On the Sorting of Points along an algebraic curve - [Johnstone-Bajaj] - Convex segment decomposition - https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1702&context=cstech - "... Given a segment AB of an algebraic curve, a set of points on the curve is sorted from A to B along AB by putting them into the order that they would be encountered in traveling continuously from A to B along AB. ...". Parallel planar point location factorization in its earlier version required global sorting of set of segments (arithmetic progressions of ordinate products) of hyperbolic arc within an interval to facilitate binary search of factor points. Local binary search obviates global sorting by assigning O((logN)^k) implicitly sorted segments each to O(N/(logN)^k) processors in parallel for sequential binary searching them which is of PRAM time O((logN)^(k+1)).

---
637. (FEATURE) NeuronRain Usecases - Analytics Piloted Drone Online Shopping Delivery - 25 February 2019

1.This commit creates a new NeuronRainApps/ directory in python-src/ for illustrating usecases defined in asfer-docs/NeuronRainUsecases.txt
2.An unmanned aerial vehicle usecase is implemented by a pseudocode in NeuronRainApps/Drones/OnlineShoppingDelivery.py
3.Following static mission plan example from DronecodeSDK has been changed to OnlineShoppingDelivery.py:
    3.1 Example Drone Mission - https://github.com/Dronecode/DronecodeSDK-Python/blob/master/examples/mission.py
    3.2 Drone MissionItem Proto - https://github.com/Dronecode/DronecodeSDK-Proto/blob/a54b353d73ff8d6e36c716b5278990e0f8cb770c/protos/mission/mission.proto
4.Dynamic Mission Plan: Autopilots the UAV drone based on GIS analytics navigation variables
- e.g. longitude, latitude, altitude, speed, camera action etc., - read by Streaming Abstract Generator socket streaming and appends MissionItems to
flight plan dynamically restricted by ordinates convex hull. Convex Hull for a terrain image is obtained from ImageNet/ImageGraph_Keras_Theano.py. When drone is within convex hull airspace, its altitude is set to 0 (or minimal value) and landed
5.Dynamic flight trajectory remote-controlled by analytics are applicable to online shopping deliveries when multiple courier items have to be delivered en route which is NP-Hard Hamiltonian problem - Finding most optimal shortest tour from origin, through intermediate delivery point vertices and back to origin

6.This is not compileable, executable code but only a pseudocode and has not been tested on a drone because of lack of it and aviation licensing requirements

References:

637.1 GIS and Drones - https://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/SiteAssets/Pages/Events/2018/Drones-in-agriculture/asptraining/A_Session1_Intro-to-Drones-RS-GIS.pdf - GIS Image Classification, Rasters and Layers

---
638.(FEATURE) Software Analytics - Scheduler Analytics - /etc/sysctl.conf kernel.sched_* optional clause - 25 February 2019

1.DeepLearning_SchedulerAnalytics.py has been changed to include new clause for writing sysctl.conf kernel.sched_* variables instead of scheduler classes within learnt_scheduler_class() based on output layer of BackPropagation Neural Network(BPNN)
2.These variables are documented in:
      2.1 https://www.kernel.org/doc/Documentation/scheduler/
      2.2 https://doc.opensuse.org/documentation/leap/tuning/html/book.sle.tuning/cha.tuning.taskscheduler.html
3.Different values of kernel.sched_* are returned based on numpy mean of output layer of BPNN. For example:
["kernel.sched_latency_ns=9000000",
 "kernel.sched_migration_cost_ns=100000",
 "kernel.sched_wakeup_granularity_ns=2000000",
 "kernel.rr_timeslice_ms=10",
 "sched_rt_runtime_us=990000",
 "sched_nr_migrate=12",
 "sched_time_avg_ms=100"]
which increase CPU affinity of a process, its timeslice quantum and reduce cost of CPU loadbalancing and migration if it is CPU-intensive.

639.(FEATURE) Software Analytics - Scheduler Analytics - /proc/sched_debug support - 27 February 2019

1.Missing kernel prefix for some of the /etc/sysctl.conf variables have been included in return array.
2./proc/sched_debug support has been included by defining a new function to read /proc/sched_debug and
print the red-black tree of the Complete Fair Scheduler (CFS) [https://elixir.bootlin.com/linux/v3.2/source/kernel/sched_fair.c] runqueue for user defined iterations in a loop - reads the /proc/sched_debug periodically, reverses the list of lines to print the tail.
3./proc/sched_debug is a convenient alternative to BCC tools runqlat
4.Kernel CFS chooses the root of the red-black tree which has least virtual runtime to schedule it to run on CPU thereby prioritizing and fairly treating starved processes.
5.Tree Key field is the virtual runtime of the process id - Example below prints runqueue at any instant for a CPU:

runnable tasks:
            task   PID   tree-key switches prio   exec-runtime   sum-exec   sum-sleep
---

```
        metacity  2709   2070924.605059   1155243   120   2070924.605059
104482.007672  13723394.606366 /autogroup-149
  unity-2d-panel  2724   2070927.500112    232355   120   2070927.500112
58204.926440  14229127.111319 /autogroup-149
  gnome-terminal  3049   2070927.622210     59498   120   2070927.622210
22064.526687  14285212.740904 /autogroup-149
R            cat  7834    612871.102038         1   120    612871.102038
2.269097         0.000000 /autogroup-153
```

6.Logs for this have been committed to
testlogs/DeepLearning_SchedulerAnalytics.log.27February2019

640.(FEATUE) Software Analytics - Streaming Analytics + Scheduler Analytics - /proc/sched_debug -
28 February 2019

1.DeepLearning_SchedulerAnalytics.py - sched_debug_runqueue() has been changed to return a dictionary
of process runqueue and loop has been removed
2.Streaming Analytics support has been provided for Operating System performance statistics (e.g runqueue) by adding new data_storage "OperatingSystem" and data_source "SchedulerRunQueue" to Streaming_AbstractGenerator.py
3.New clause for previous Scheduler Analytics streaming has been added in __iter__() which invokes
sched_debug_runqueue() in a loop and yields the scheduler runqueue dictionaries.
4.This integrates Streaming and Software Analytics seamlessly and any Streaming or other machine learning algorithm can process Operating System Stats Dictionary Stream like a bigdata source.
5.DeepLearning_SchedulerAnalytics.py imports this Streaming Abstract Generator by "OperatingSystem" and "SchedulerRunQueue" parameters and reads the scheduler stats as input stream.
6.In future, any operating system data source other than scheduler runqueue can be plugged into Streaming Abstract Generator iterable and fulfils the foremost requirement for real time software streaming analytics.

References:

640.1 Load Averages - /proc/loadavg versus /proc/sched_debug - http://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html - Linux load averages include uninterrutible IO tasks too apart from CPU load.

641.(THEORY) Word Diophantine Equations, Intrinsic Merit defined by Word Equations - 4 March 2019 -
related to 635 and all sections on Merit and Complement Diophantines

---

Throughout this draft intrinsic merit of text has been analyzed by results from algorithmic graph theory and dense subgraph complexity measures. Renowned Word Problem in computability encodes every word from an alphabet into an equation. In the context of World Wide Web, Concatenations of words form a text. This leads to concatenation representation of word equations of text by Diophantines. This kind of word equation representation is not just limited to text but to other genre of merits - audio/music, video, social profiles of people encoded as words over binary alphabet - as well. Thus Diophantine representation unifies all 4 classes of merit - text, audio/music, video, people - into one thereby facilitating number theoretic analysis of merit drastically different from graph theoretic perspective earlier:

(*) Text - Text Graphs -> Text Word Equation Diophantines
(*) Video - EventNet Tensor Products -> Binary encoded Video Word Equation

Diophantines
(*) Audio/Music - DFA Learning -> Binary encoded Audio/Music Word Equation Diophantines
(*) People - Mistake Bound Experiential Learning Merit and Social Profile Analytics -> Binary encoded social profile Word Equation Diophantines

Decidability of Hilbert's Tenth Problem is still open for solutions to Word Diophantine Equations. Diophantine Word Equations can enforce arbitrary number of conditions on the solutions e.g length of words.Solutions to a Word Diophantine Equation - text, audio/music, video, people - are kind of unsupervised classifiers which cluster similar texts, audio/music, video and people. For example, Complement Word Equation Diophantines bifurcate the set of texts, audio/music, video and people.

References:

641.1 Word Equations, Fibonacci Numbers, and Hilbert’s Tenth Problem - [Yuri Matiyasevich] - Steklov Institute of Mathematics at Saint-Petersburg, 27 Fontanka, Saint-Petersburg, 191023, Russia - URL: http://logic.pdmi.ras.ru/~yumat - https://logic.pdmi.ras.ru/~yumat/talks/turku2006/FibonacciWordsAbstract.pdf.gz - "...Every word X = χn χn−1 . . . χ1 in the alphabet B can be viewed as the number x = χn un + χn−1 un−1 + · · · + χ1 u1 (1) written in positional system with weights of digits being the Fibonacci numbers u1 = 1, u2 = 1, u3 = 2, u4 = 3, u5 = 5, . . . (rather than traditional 1, 2, 4, 8, 16, . . .)...."
641.2 Collected works of Richard J.Buchi - https://books.google.co.in/books?id=9BfvAAAAMAAJ&q=inauthor:%22J.+Richard+B%C3%BCchi%22&dq=inauthor:%22J.+Richard+B%C3%BCchi%22&hl=en&sa=X&ved=0ahUKEwio19KMhejgAhXEfXAKHW6tB7oQ6AEIKjAA - Every existential formula in concatenation is Diophantine

---
642. (FEATURE) Merit of Audio/Music - Music Synthesizer from Random samples and State Machine - 5 March 2019

1.New function notes_to_audio() has been implemented in music_pattern_mining/AudioToBitMatrix.py
2.It creates a .wav file from random Scipy array or traverses a finite state machine to create a stream of notes in CDEFGABC notation which are translated to integer array of frequencies in hertz
3.Scipy.io.wavefile.write() is invoked to write out this ndarray to a .wav file of sampling rate 44100
4.Finite state machine has been implemented simply in a dictionary and supports quasi-non-determinism (array of possible previous states)
5.Two .wav files automaton_synthesized_music.wav and notes_synthesized_music.wav are written for the two clauses: automaton or notes array
6.logs for synthesizer have been committed to testlogs/AudioToBitMatrix.log.MusicSynthesizer.5March2019
7.It is pertinent to mention here that Stanford Music Information Retrieval at https://musicinformationretrieval.com/genre_recognition.html does a statistical genre recognition of music as opposed to automata theoretic analysis in NeuronRain
8.State machine has been hardcoded presently which have to be read from file and literal_eval()-ed.

---
643. (FEATURE) Merit of Audio/Music - Music Synthesizer - Music from mathematical functions - 6 March 2019

1.New clause has been added to notes_to_audio() for synthesizing music from

values of any mathematical function
2.Function expression string is eval()-ed in a lambda function and applied to a list of range values of variable x by map() and a wave file function_synthesized_music.wav is written to.
3.Error in prevstates has been corrected by prevprevstates and resetting prevstates for each state transition for an alphabet.
4.Logs for this have been committed to testlogs/AudioToBitMatrix.log.MusicSynthesizer.6March2019
5.The wave files do not play any human palatable music and are representative only.
6.Infact learning the right DFA (or in general a mathematical function) for language of music is the exact opposite of the previous and is measure of merit.

---
644. (FEATURE) Intrinsic Merit of Audio/Music - Mel Frequency Cepstral Coefficients - 11 March 2019

1.New function for Mel Frequency Cepstral Coefficients (MFCCs) of a music audio clip (invokes librosa) has been added in this commit.
2.MFCCs are kind of spectra of spectra of an audio waveform and have been widely used for speech recognition and music genre identification.
3.logs for example MFCCs of a music file are committed to testlogs/AudioToBitMatrix.log.MFCC.11March2019

References:

644.1 Stanford MIR - https://musicinformationretrieval.com/mfcc.html
644.2 MFCCs for Music Modelling - http://musicweb.ucsd.edu/~sdubnov/CATbox/Reader/logan00mel.pdf
644.3 Music Emotion Recognition (MER): The combined evidence of MFCC and residual phase - [NJ Nalini , S Palanivel] - Dept of Computer Science and Engineering, Annamalai University, Tamil Nadu 608002, India - https://www.sciencedirect.com/science/article/pii/S1110866515000419

---
645. (THEORY) Comparing Partitions - Related to 620 and all sections on Majority Voting, Theoretical EVMs, Separate Chaining Hashtable/LSH Set Partitions, Integer Partitions, Complementary Sets and Diophantines, Streaming Histogram/Dictionary Analytics, People Analytics, Software Analytics, Business Analytics - 15 March 2019

Previous sections deeply delve into the ubiquitous isomorphism between integer/set partitions and set partitions, particularly those induced by hash table separate chaining buckets. A Theoretical EVM has been formulated having separate chaining partition as basis which is an encrypted set partition of electorate into candidate buckets. Drawing analytic inferences from stream of set partitions has been hitherto a far less researched bigdata topic and applies to whole gamut of theory and analytics - For example:

(*) Business Analytics - Customer base intelligence is often represented as dynamic stream of histogram set partitions or probability distributions of product influences. Hashtable separate chaining can be picturised as histogram of buckets of hashed items

(*) People Analytics - Stream of voting patterns in Theoretical EVMs are dynamically changing set partitions of electorate - vote shares in Psephology

(*) Scheduler Queuing Analytics in Operating Systems - an example hashtable based timeout is described in https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt - finding patterns in stream of scheduler timeout hashtable queues

(*) Music Spectrograms are streams of Histograms
(*) Theoretically set partitions are complementary sets or exact cover of a universal set.

Comparing any two set partitions thus far has few standard measures (rooted in comparing two cluster partitions created by supervised or unsupervised classifiers) like Rand Index (RI), Adjusted Rand Index etc., (ARI) which compute distance between partitions as ratio of agreements to disagreements by measuring overlaps. For a stream of set partitions (which could be any of the above), Rand Index or Adjusted Rand Index between two consecutive partitions in the stream themselves create another stream of scalar indices which are easier to interpret. In people analytics or business analytics example above, highly fluctuating stream of rand indices for voting or customer histogram patterns is a measure of disarray in voting and consumers and points to a state of flux whereas a periodic or steady state of stream of rand indices indicates voters and consumers have limited options or are content with status quo. Similarly a chaotic stream of rand indices for Scheduler implies a heavily thrashed system.

Theoretically, rand indices define distance metric between two complementary sets and therefore diophantines (e.g beatty functions for size two partition) defined on them - in other words, diophantines are embedded in a metric space by rand index distance metric (similar to Reproducing Kernel Hilbert Space of Functions mentioned in 624.14).

References:

645.1 Comparing Partitions - Journal of Classification 2:193-218 (1985) - Springer-Verlag New York Inc.
- [Lawrence Hubert ,The University of California, Santa Barbara - Phipps Arabie, University of Illinois at Champaign] -
https://link.springer.com/article/10.1007/BF01908075
645.2 Adjusted Rand Index -
https://en.wikipedia.org/wiki/Rand_index#Adjusted_Rand_index

---
646. (THEORY) Complementary Equations, Spectra of Numbers, Complementary Integer Sequences - related to 645 and all sections on Complement Functions, Complementary Sets and Diophantines - 16 March 2019

Increasing sequences which partition set of natural integers are known as complementary sequences and equations which define both of them are complementary equations. This definition of complementary equations differs technically from complement functions which are diophantine equation representations of either a set or its complement and not both at once. OEIS sequences b(n)=A005228 and a(n)=A030124 are complementary and both are defined simultaneously by complementary equation b(n) = a(n − 1) + b(n − 1)

Spectra of numbers are integer sequences and defined in terms of rounding off set of real multiples to nearest integers. For example:
If r is a positive real number, spectra of r = S(r) = { [r],[2r],[3r],...} and [x] is the nearest
integer <= x.

There are some special qualities of Spectra of integers - e.g. Spectra of any three real numbers S[r1],S[r2],S[r3] have some pairs which have infinitely many elements in common and thus cannot be complementary and thus is a prima facie condition for feasibility of complementation.

References:

646.1 Complementary Equations - Example 3 - [Journal of Integer Sequences, Vol. 10 (2007),Clark Kimberling,Department of Mathematics,University of

Evansville,1800 Lincoln Avenue,Evansville, IN 47722,USA] - https://cs.uwaterloo.ca/journals/JIS/VOL10/Kimberling/kimberling26.pdf
646.2 Complementarity in Philosophy - Dualism - Taichi - Yin-Yang (Dark-Bright) - https://en.wikipedia.org/wiki/Yin_and_yang
646.3 OEIS - Hofstadter Sequence - https://oeis.org/A005228
646.4 OEIS - Complement of Hofstadter Sequence - https://oeis.org/A030124
646.5 Spectra of Integers - [Graham-Lin-Lin] - https://www.jstor.org/stable/2689998?seq=1#page_scan_tab_contents

---

647. (THEORY and FEATURE) Bounded Error Factorization (Approximate Factors), Ray Shooting Queries, Transition from Quantum to Classical Computation, Derandomization - related to 597,631 - 21 March 2019, 2 August 2019

In Computational Geometric Factorization algorithms described previously, Approximate Factors are found by prime factor estimation involving various number theoretic results which narrow down the location of exact factor to large extent. By searching a circular radius vicinity surrounding an approximate factor, probability of finding a factor is amplified. Error probability of not finding an exact factor by ray queries has been derived previously as:

$$\frac{N/kloglogN - 2q(logN)^l}{N/kloglogN}$$

for integer-to-factorize N and constants k,q and l. This error probability though exponential can be brought down to a desired bound by suitable choice of radius and values for constants k,q and l or by precision tile summation ray shooting. Shor's BQP factoring algorithm can be made a PP classical probabilitstic polytime algorithm by [Adleman-DeMarrais-Huang] theorem - BQP in PP. By invocation of Ray Shooting Oracles as subroutines in this PP factoring algorithm, PP is error-bounded to a BPP or BPNC algorithm (by doing ray shooting queries on parallel RAMs). Derandomization of BPP to P has known Hitting Set Generator algorithms. BPNC derandomization has some restrictions - only derandomizing BPNC1 is known. But this could be sufficient because classical PRAM computational geometric factorization is achievable by Parallel Planar Point Location NC1 algorithms of O(logN) parallel time. This pipelined derandomization of BQP through BQP-PP-BPP(or BPNC)-P(or NC) points to prima facie feasibility of wave function superposition collapse (a quantum universal gate) to some classical eigenstate (PRAM).

References:

647.1 Nisan's PRG and Derandomization of BPNC1 - [Periklis Papakonstantinou] - Section 3.4 - "...We consider logspace machines without a stack, and we parametrize on the number of passes over the random tape. This defines a hierarchy of classes between LogSpace and BP∗LogSpace ⊇ BPNC1 [Nis93b]. Therefore a full derandomization of this hierarchy derandomizes BPNC1 ..." - Section 3.4.1 - http://papakonstantinou.org/periklis/pdfs/phd_thesis.pdf - "...Observe that a PRG that fools logspace machines with two-way access over the input, also fools NC1 circuits. The classical PRG of Nisan [Nis92] is shown to fool logspace machines that make r(n) = 1 pass over the random or pseudorandom input. Using a simple analysis, we show that for every k > 0 there exists k' such that for seeds of length log^k'(n) the original Nisan’s PRG is also secure against logspace machines with r(n) = log^k(n) passes..."
647.2 Simplified Derandomization of BPP by Hitting Set Generator - [Goldreich-Vadhan-Wigderson] - http://www.wisdom.weizmann.ac.il/~oded/COL/gvw.pdf
647.3 Quantum Factorization and Decoherence - https://arxiv.org/pdf/quant-ph/9503007.pdf - [I. L. Chuang, R. Laflamme, P. Shor and W. H.Zurek] - "...As a quantum system evolves, information about its states leaks out into the environment, causing them to loose their purity, and, consequently, their ability to interfere....The decoherence process has been

proposed as a mechanism for enforcing classical behavior
in the macroscopic realm. Decoherence results in environment-induced
superselection[5, 6, 7] which destroys superpositions between the states of
preferred pointer basis[6]. Classical computers are already decohered – "
647.4 Proof of Ambainis-Aaronson Conjecture - "Quantum Speedups need structure"
- "...Let f : {−1, 1}^n → [−1, 1] be a multilinear polynomial of degree d. Then
there exists a variable xi whose influence on f is at least poly(Var(f)/d)...."
- [Nathan Keller∗ and Ohad Klein†] - https://arxiv.org/pdf/1911.03748.pdf - "...
Let Q be a quantum algorithm that makes T queries to a Boolean input and let ϱ,
δ > 0. Then there exists a deterministic classical algorithm that makes poly(T,
1/ϱ, 1/δ) queries to the input and that approximates Q's acceptance probability
to within an additive error ϱ on a 1 − δ fraction of inputs. In other words, any
quantum algorithm can be simulated on most inputs by a classical algorithm which
is only polynomially slower, in terms of query complexity...." - proof of this
conjecture lends credence to the pipelined decoherence of BQP factorization to
BPP or BPNC mentioned earlier, in terms of query complexity - BQP-PP-BPP/BPNC -
effectively implying decohered reduction from quantum error to classical error
while classical error still depends on success of derandomization from BPP or
BPNC to P or NC.

---
648. (THEORY) Voting in Knot Theory, Gordion Knot, One-Way Functions, Hardness
Amplification Lemma for Majority + VoterSAT Boolean Function Composition,
Separate Chaining or LSH Partition based Theoretical EVMs - related to 318, 517,
620 - 21 MArch 2019

Knot theoretic formulation of One-Way-Functions (as a Gordion Knot) has been
mentioned in
https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/
AdvancedComputerScienceAndMachineLearning/
AdvancedComputerScienceAndMachineLearning.txt. Previous sections describe a
hard-to-invert MajorityInverse() function which extracts the identity of the
voters who voted for or against a candidate (for simplicity binary 0 or 1).
Majority function can be formulated as a knot polynomial by following example
procedure:

(*) Electorate are points on a straightline string

(*) Each voter has a decision function or VoterSAT of arbitrary complexity
class

(*) Knot on this string of voters maps each voter to a point on locus of a
knot polynomial in
3 dimensional space. This mapping obfuscates the identity of the voter and
vouchsafes secrecy.

(*) Knot polynomial is defined on 3 variable axes - Voters, Candidates,
VoterSATs

(*) Each point on the knot is a tuple (obfuscated_voter_id,
obfuscated_VoterSAT, candidate_id) denoting an anonymous vote in ballot

(*) MajorityInverse() for this Knot ballot is defined as:

MajorityInverse(obfuscated_voter_id, obfuscated_VoterSAT,
candidate_id) = voter_id

(*) Previous example in 3-dimensions is only for illustrative purposes
(Obfuscated VoterSAT in the vote tuple may not be necessary) and knot ballot
polynomial can be embedded in arbitrary topological space other than R^3.

(*) Hardness of previous knot version of MajorityInverse() depends on
difficulty in unravelling the knot (to retrieve voter who voted for a candidate)
so that secret ballot is not compromised.

(*) Thus a Gordion Knot Ballot is hard-to-invert MajorityInverse()
function.

(*) Number of votes for a candidate is determined by the number of points
on the knot polynomial intersecting the 2-dimensional plane for a candidate axis
point.

---
649. (THEORY) Planar Graph Theoretic Circle Packing, Space Filling/Grid Filling, CSP/Linear Programming, Tile Cover and LSH/Separate Chaining Partitions - related to 135 - 25 March 2019, 1 April 2019

Previous algorithms for packing/filling space by circular solids/fluids were based on solving Linear Program and Cellular Automata by a Parallel Pseudorandom Generator. In this section a random graph theoretic circle packing algorithm is described:

Circle Packing problem is defined as filling a grid with non-overlapping circles of maximum possible radii. In other words, sum of radii of the non-overlapping tangential circles are maximized by a constraint satisfaction problem(CSP) or LP. Rectangular grid is mapped to a linear program x11 + x12 + ... + xmn for each ordinate x(ij) of the grid. Points on the grid are randomly filled by a Parallel Pseudorandom Generator and these points are centroids of circles of arbitrary radius which intersect the grid. All grid ordinates intersected by or within the circles are set to 1. Circle Packing Theorem implies this set of tangential circles draw a planar intersection graph or coin graph. Conversely for every planar graph, a set of tangential circles can be demarcated which are solutions to a CSP/LP. This version of circle packing has an added parallel randomness ingredient and therefore is a Parallel Random Graph Circle Packing:

```
      while (there are unfilled grid ordinates)
      {
            (*) Centroids of tangential circles (which are vertices of random
intersection graph) are chosen by a PRG in parallel
            (*) edges are created at random amongst centroid vertices which
simulate tangential circles and have length equal to sum of radii of two
tangential circles
            (*) any overlap of circles is avoided by shrinking the sum of radii
as necessary.
            (*) all ordinate variables within random tangential circles are set
to 1.
            (*) Grid ordinates are stored in a hashmap and an ordinate is
removed from map after being set to 1.
            (*) At any instant, Parallel PRG choooses only from remaining
ordinates in the hashmap which is exhausted when grid is completely filled by
circles.
      }
```

Previous Parallel Random Graph Circle Packing is in BPNC/RNC and indirectly probabilistically solves the constraint satisfaction problem for sum of radii in parallel and is an online algorithm (whereas sequential CSP/LP solution guarantees that the tangential osculating circles have maximum radius possible apriori).

Circle Packing is also an exact cover set partition of 2-D space (e.g infinite number of circles in Apollonian Gasket). Set partition and space filling (by Circle packing, Tile cover etc.,) have an isomorphism. Separate Chaining Hashtable and LSH partitions are isomorphic to exact tile cover of a rectangular 2-D region and there exists a connection to factorization if Number of elements of a set S is composite and size of every bucket in set partition of S is composite i.e every bucket in histogram is mapped to a rectangular tile in tile cover of S plotted as a rectangle. Size of every bucket can be written as sum of four squares by Lagrange Four Squares Theorem and thus every set in a set partition can be mapped to 4 juxtaposed squares which reduces Balls-Bins/Set partition problems to Tile Packing.

References:

649.1 Cicle Packing Survey - Apollonian Gasket - Computational Effort in finding radii - Figure 2 - Packing and Simplicial Complexes - Discrete Riemann Mapping Theorem - Maximal Packing - http://www.math.utk.edu/~kens/Notices_article.pdf -

Previous algorithm for Circle Triangulation Packing in Apollonian Gasket is a greedy algorithm where maximum possible radii are chosen initially which are fractally reduced
649.2 Descartes Circle Theorem - https://en.wikipedia.org/wiki/Descartes%27_theorem - Every four kissing quadratic circles satisfy the quadratic equation - (b1 + b2 + b3 + b4)^2 = 2(b1^2 + b2^2 + b3^2 + b4^2) where bi are bends or curvatures (radii) of the four circles. Quite useful in previous randomized parallel algorithm in finding the radius of inscribed fourth circle.
649.3 An example Constraint Satisfaction Problem for unequal Circle Packing in rectangle by 0-1 Integer Linear Programming - Integer Programming Formulations for Approximate Packing Circles in a Rectangular Container - https://www.hindawi.com/journals/mpe/2014/317697/

---

650. (THEORY and FEATURE) Streaming Set Partition (Histogram) Analytics - 28 March 2019

1.New Python implementation for multipurpose analytics of streamed set partitions (histograms) has been committed - Streaming_SetPartitionAnalytics.py
2.Two new bigdata sources have been defined in Streaming_AbstractGenerator.py - "TextHistogramPartition" and "DictionaryHistogramPartition" which respectively:
      (*) Map a list of textfiles to list of histograms by finding word frequency in the text and yield partition datastructure as iterable which is a dictionary of words to buckets in histograms (occurrences are mapped to array of 1s e.g occurence 5 for word "word" is mapped to binned bucket ('word', [1,1,1,1,1]))
      (*) Read a text file - Streaming_SetPartitionAnalytics.txt - containing list of dictionaries, literal_eval() it and yield the elements as iterable
3.TextHistogramPartition is a fancy datasource which facilitates an alternative measure of streaming text analytics and text similarity
4.DictionaryHistogramPartition is the conventional datasource for succinctly representing a set partition which could be anything ranging from LSH partition, Exact Cover, Balls-Bins, Separate Chained Hashtables, Electronic Voting Machines based on LSH and Separate Chaining, Business Intelligence Histograms, SMS voting histograms, Music Spectrograms etc.,.
5.adjusted_rand_index() in Streaming_SetPartitionAnalytics.py instantiates both these datasource iterators and computes Adjusted Rand Index for two consecutive partitions in the stream by SciKitLearn adjusted_rand_score()
6.Because of the fact that every set partition is set of class labels in some classifier, various other clustering performance measures - mutual information, silhouette etc.,
(https://scikit-learn.org/stable/modules/clustering.html#clustering-evaluation) could compare partitions too.
7.tocluster() in Streaming_SetPartitionAnalytics.py maps a partition to an array of labelled elements in scikitlearn format
8.An example streaming set partition analytics for linux kernel logs (TextHistogramPartition) and list of dictionaries (DictionaryHistogramPartition) is committed to Streaming_SetPartitionAnalytics.log.28March2019 which show how Adjusted Rand Index works for similar and dissimilar partitions.
9.For circumventing sklearn exception two compared partitions are truncated to minimum size of the two

---

651. (THEORY and FEATURE) Reduction from Set Partition to Exact Square Tiles Cover of Rectangular Region,
Lagrange Four Square Theorem, Complementary Sets, Computational Geometric Factorization, Space Filling, Tile Cover, Packing, Set Partition/Histogram Analytics, LSH Partitions, Separate Chaining Hashtables - 3 April 2019, 17 November 2019 - related to 649 and 650

1.Python implementation for Set Partition Analytics (Streaming_SetPartitionAnalytics.py) has been updated for a new function which reduces a histogram set partition to exact square tiles cover of rectangular region which has area equal to number of elements in the histogram.
2.Function setpartition_to_tilecover() loops through histogram buckets (set partition) and maps each bucket size to sum of four squares by Lagrange Four Square Theorem essentially mapping each 1-dimensional bucket to 4 2-dimensional square tiles.
3.Collection of these sets of 4 square tiles covers the rectangular region of area equal to number of elements partitioned by histogram/LSH/Separate Chaining.
4.Example logs in testlogs/Streaming_SetPartitionAnalytics.log.3April2019 demonstrate reduction of histogram buckets [11,12,13,14,15] of total number of 65 elements to tile cover of rectangular region which has composite area 13*5=65:

```
square tiles for partition  11 : set([(0, 1, 1, 3)])
square tiles for partition  12 : set([(1, 1, 1, 3)])
square tiles for partition  13 : set([(1, 2, 2, 2)])
square tiles for partition  14 : set([(0, 1, 2, 3)])
square tiles for partition  15 : set([(1, 1, 2, 3)])
Lagrange Four Square Tiles Cover reduction of Set Partition  [11, 12, 13, 14, 15] : [0, 1, 1, 9, 1, 1, 1, 9, 1, 4, 4, 4, 0, 1, 4, 9, 1, 1, 4, 9]
```

5.This reduction indirectly solves factorization problem (complexity depends on sum of four squares) geometrically by following square tiles cover algorithm:

5.1 Number to factorize N is partitioned arbitrarily by some integer partition and is isomorphic to a histogram

5.2 Square tiles are obtained from partition by previous Lagrange Four Square reduction

5.3 Square tiles are geometrically juxtaposed/arranged to create a tiled rectangle sides of which are factors of N

5.4 Finding an arrangement of tiles on a rectangular region can be formulated as a non-linear (quadratic) convex-concave optimization problem. If x1,x2,x3,...,xn are sides of square tiles:

N = area of rectangle = sum of areas of square tiles = x1^2 + x2^2 + x3^2 + ... + xn^2

= product of sum of sides of peripheral tiles

= (c1*x1 + c2*x2 + ... + cn*xn)*(d1*x1 + d2*x2 + ... + dn*xn) for ci = 0 or 1, di = 0 or 1

5.5 Solving N = x1^2 + x2^2 + x3^2 + ... + xn^2 = (c1*x1 + c2*x2 + ... + ck*xk + cn*xn)*((1-c1)*x1 + (1-c2)*x2 + ... + ck*xk + (1-cn)*xn) by Integer Programming for binary ci amounts to factorizing N (because tiles of the sides have to be mutually exclusive save corner tile)

5.6 The corner square tile is common to both sides and ck = dk for exactly one square tile xk.

If Factorization is doable in PRAMs (and therefore in Nick's Class) as per previous sections on various parallel RAM polylogarithmic time algorithms for factoring (e.g Parallel Planar Point Location), geometric arrangement of square tiles in a rectangle is of polynomial time - find factors which are sides of rectangle, successively tile the rectangle by choosing largest square remaining. Many randomized algorithms are available for finding Lagrange Sum of squares. If the factors of N = p*q are known in polylogarithmic time finding the tile arrangement is by solving two linear programs:

c1*x1 + c2*x2 + ... + ck*xk + ... + cn*xn = p

(1-c1)*x1 + (1-c2)*x2 + ... + ck*xk + ... + (1-cn)*xn = q

which are underdetermined linear equations (n variables and 2 equations) and constraints are specified by complemented binary coefficients ci and 1-ci and ck=dk in the two equations for mutually excluding tiles in the sides of the rectangle except corner tile. This boolean coefficient complement constraint can be ignored if two sides of the rectangle can have tiles of similar squares. Another relaxation: Number of tiles can be unequal in either sides. Equations then are easier to interpret as:

c1*x1 + c2*x2 + ... + ck*xk + ... + cm*xm = p

d1*x1 + d2*x2 + ... + dk*xk + ... + dn*xn = q

subject to one constraint for corner tile ck=dk and number of variables m and n could be unequal. Number of variables m and n can be equated by additional |m-n| slack variables of zero coefficients in equation of lesser variables. From Rouche-Capelli theorem number of solutions to underdetermined system of equations is Infinite because rank of augmented matrix is < rank of coefficient matrix for underdetermined system of equations. Solutions can be found by many algorithms prominent being Gauss-Jordan elimination (Gaussian Elimination by Partial Pivoting-GEPP). GEPP has been proved to be P-complete problem. Previous system of equations are solved by GEPP post-factorization (sides of rectangle p,q are factors of N found by PRAM computational geometric factorization). If NC=P, both sides of rectangle and arrangement of square tiles on the sides of the rectangle can be done in NC by PRAM GEPP which is a depth-two parallelism. Without knowledge of factors, previous product (c1*x1 + c2*x2 + ... + cm*xm)*(d1*x1 + d2*x2 + ... + dn*xn) = pq = N is a quadratic program solving which is NP-Hard.

Point location by parallel construction of wavelet trees has been described in previous sections,which consider the set of points of a geometric variety (e.g points on rectified hyperbolic arc) on the plane as a string of points represented as a wavelet tree. Point Location by Parallel Wavelet Tree construction is more obvious than Parallel Point Location by Kirkpatrick Triangulation, Planar subdivision, Bridge Separator Trees etc.,. The string of points formed from rectified hyperbolic arc are concatenated straightline segments each of which is an arithmetic progression. Query select(B, N, i) returns the position p of the i-th occurrence of N in concatenated hyperbolic line segments string B from which i-th factors of N=(p, N/p) are obtained.

Previous reduction from set partition to tile cover by Lagrange Four Square Theorem is restricted to tiling 2-dimensional space. It can be generalized to filling/tiling arbitrary dimensional space by Chinese Remainder Theorem as follows:

(*) Chinese Remainder Theorem states that there is a ring isomorphism between x mod N and (x mod n1, x mod n2, ..., x mod nk) for pairwise coprime integers n1,n2,...,nk and N = n1*n2*n3*...*nk (or) x mod N <=> (x mod n1, x mod n2, ..., x mod nk) which is an isomorphism between ring of integers mod N and direct product space of rings of integers modulo nk

(*) By Chinese Remainder Theorem, there is a bucket of size x < N (so that x mod N = x) of the set partition which can be mapped isomorphically to a k-dimensional hypercube of sides (x mod n1, x mod n2, ..., x mod nk) modulo an integer N=n1*n2*...*nk (requires solving for x).

(*) This Chinese Remaindering Reduction reduces 1-dimensional set partition to tiling arbitrary dimensional space

References:

651.1 Wang Tiling - https://en.wikipedia.org/wiki/Wang_tile - Previous Lagrangian Tiling can be contrasted to Wang Tiles which is about feasibility of tiling a 2-D plane by multicolored square tiles by matching colors of adjacent tiles.

651.2 Alternative partition distance measures - https://arxiv.org/pdf/1106.4579.pdf

651.3 Clustering for set partitioning - k-set partitioning and centroid clustering are equivalent - https://www.microsoft.com/en-us/research/wp-content/uploads/2016/11/OPT2015_paper_32.pdf

651.4 Sage Set Partitions Implementation - http://doc.sagemath.org/html/en/reference/combinat/sage/combinat/set_partition.html#sage.combinat.set_partition.SetPartition

651.5 Sage Ordered Set Partitions Implementation - complement of a partition - http://doc.sagemath.org/html/en/reference/combinat/sage/combinat/set_partition_ordered.html

651.6 Jacobi Sum of Four Squares Theorem - https://en.wikipedia.org/wiki/Jacobi%27s_four-square_theorem - "...The number of ways to represent n as the sum of four squares is eight times the sum of the divisors of n if n is odd and 24 times the sum of the odd divisors of n if n is even ..."

651.7 Succinct and Implicit Data Structures for Computational Geometry - [Meng He,Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 4R2, Canada,mhe@cs.dal.ca] - Figure 1 - "A wavelet tree constructed for the string 35841484 over an alphabet of size 8.This is also a wavelet tree constructed for the following point set on an 8 by 8 grid:{1, 3}, {2, 5}, {3, 8}, {4, 4}, {5, 1}, {6, 4}, {7, 8}, {8, 4}" and Section 3 - Planar Point Location - https://web.cs.dal.ca/~mhe/publications/ianfest66_succinctgeometry.pdf
651.8 Simple,Fast and Lightweight Parallel Wavelet Tree Construction - [Fischer-Kurpicz-Lobel] - https://arxiv.org/pdf/1702.07578.pdf - Comparison to Domain Decomposition Parallel Wavelet Tree Construction and Variant of Wavelet Tree (Wavelet Matrix) - In the context of Computational Geometric Parallel Planar Point Location, size of the alphabet for wavelet tree is very small in the range [N-delta, N+delta] because hyperbolic tile segments have product of ordinates which are more or less equal to N.
651.9 Planar Point Location - Algorithms - https://cw.fel.cvut.cz/b181/_media/courses/cg/lectures/02-pointloc.pdf
651.10 Segment Trees and Point Location Query - http://www.cs.umd.edu/class/fall2016/cmsc754/Lects/cmsc754-fall16-lects.pdf
651.11 Rouche-Capelli Theorem for Underdetermined system of equations - https://en.wikipedia.org/wiki/Underdetermined_system#Solutions_of_underdetermined_systems, https://en.wikipedia.org/wiki/Rouch%C3%A9%E2%80%93Capelli_theorem
651.12 Parallel Complexity of Gaussian Elimination by Partial Pivoting - https://algo.ing.unimo.it/people/mauro/strict.ps
651.13 CGAL 5.0 Planar Point Location implementation - https://doc.cgal.org/latest/Arrangement_on_surface_2/index.html#title14
651.14 Parallel CGAL - Multicore parallelism in CGAL is achieved by Intel TBB - https://github.com/CGAL/cgal/wiki/Concurrency-in-CGAL

---

652. (FEATURE) Histogram-Set Partition Analytics - Adjusted Mutual Information - 3 April 2019

1.adjusted_mutual_info_score() from sklearn.metrics has been invoked in adjusted_rand_index() for comparing two set partitions.
2.Logs in testlogs/Streaming_SetPartitionAnalytics.log2.3April2019 show the difference between two measures - adjusted rand index and adjusted mutual information

---

653. (FEATURE and THEORY) Computational Geometric Factorization - Tile Search - Spark 2.4 -SparkContext
range() - 16 April 2019

1.DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been changed to replace xrange() by Spark Context range() function in Spark 2.4.
2.Python xrange() is sequential and local while Spark range() is resilient distributed dataset which can be computed on cloud in parallel
3.Some ray query functions have also been transitioned to Spark Context range().
4.An example integer is factorized and logs in testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.log.16April2019 show some accurate ray shooting by Hardy-Ramanujan approximate factoring and exact factors.
5.Spark 2.4 has Barrier Synchronization feature (mentioned experimental) which is BSP model (like Pregel) than PRAM - https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.BarrierTaskContext.get
6.Greenwich Mean Time (gmtime()) is printed after every factor is found for duration analysis of finding all factors.

---

654. (THEORY and FEATURE) EventNet creation from Video, Actor Model, PetriNets, Video EventNet Tensor Products Intrinsic Merit - 19 April 2019 - related to all sections on Causality-Logical Time and Merit
---
1.ImageNet Keras-Theano Python implementation has been changed to write EventNet Vertices and Edges
files from causally related frame events of the video which can be read by GraphViz or NeuronRain EventNet Python implementation
2.EventNet differs from traditional PetriNet in following respects:
	2.1 PetriNets have vertices having tokens called "Places" while EventNet has vertices having "actors" or "participants" in an event (Actor-Event Model)
	2.2 PetriNets have transitions which move the tokens from Place-a to Place-b while EventNet has directed edges which connect a causing event vertex to an effect event vertex which is akin to flow of tokens.
	2.3 EventNet is an Actor Model where actors interact in an event, modify local states and send messages to other actors.
3.Video EventNet vertices and edges are written to Video_EventNetEdges.txt and Video_EventNetVertices.txt

References:

654.1 Actor Model - https://en.wikipedia.org/wiki/Actor_model
654.2 PetriNets - https://www.techfak.uni-bielefeld.de/~mchen/BioPNML/Intro/pnfaq.html

---
655. (THEORY and FEATURE-BUGFIX) EventNet creation from Video - bug resolutions, corrections to EventNet vertices and edges text files - 21 April 2019

1.ImageNet Keras Theano Python Implementation had some bugs in creating EventNet vertices and edges text
files which have been resolved in this commit.
2.Frame prevfix has been included in EventNet edges text file.
3.Extracting EventNet from video facilitates computing usual dense subgraph complexity measures for Video
similar to TextGraphs - e.g k-core, cycles, strongly connected components, algebraic connectivity etc.,

---
656. (FEATURE) Medical Imageing Analytics - ElectroCardioGram (ECG), EventNet from Video - 23 April 2019

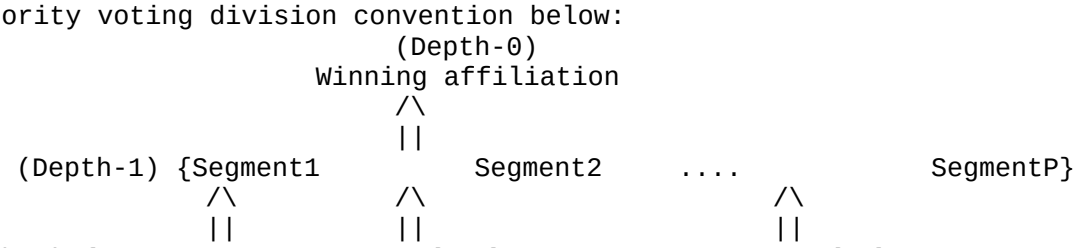
1.Extracting patterns in medical imagery (ECG, MRI, Echo Cardio Gram Scans etc.,) is one of the most obvious analytics that can be performed on a Medical Image dataset which has widespread requirement in healthcare.
2.ImageNet Keras-Theano Python implementation has been augmented with a medical_imageing() function which takes an image source (presently ECG) and thresholds the image to retrieve contours from it by CV2 findContours().
3.Image Contours are prominent features in an image which are more pronounced and distinguish it from other images.
4.Two example ECG images (Normal and Infarction) are compared by creating contours (i.e extract the ECG waveform from image) and computing the hausdorff distance between the two image contours:
5.Example comparison from logs in testlogs/ImageGraph_Keras_Theano.log.MedicalImageing.23April2019:
Distance between Normal ECG and Normal ECG: (0.0, 0, 0)
Distance between Normal ECG and Infarction ECG: (1741.9586677071302, 0, 0)
6.EventNet edges and vertices numbering in text files Video_EventNetEdges.txt and Video_EventNetVertices.txt has been corrected.

References:

656,1 Medical Imageing Analysis - https://www.pyimagesearch.com/2018/12/03/deep-learning-and-medical-image-analysis-with-keras/
656.2 ECG Library - https://ecglibrary.com/ecghome.php

---
657. (THEORY) Depth-2 Majority Function, Theoretical EVMs, Set Partitions, People Analytics, Axiom of Choice - related to 546,490 and all sections on majority voting - 25 April 2019

It has been mentioned in earlier sections on set partitions that every theoretical majority voting partitions the population into subsets or buckets indexed by the candidate voted for. Most realworld democracies follow the depth-2 majority voting division convention below:



and each EVM is defined by the hiearachy of partitions(candidates per segment might vary as n(1),n(2),...,n(P)) - votes per candidate k in q-th EVM of p-th segment (EVMpq) is denoted by ckpq :

```
EVM11 = c111 + c211 + c311 + ... + cn(1)11
EVM12 = c112 + c212 + c312 + ... + cn(1)12
...
EVM21 = c121 + c221 + c321 + ... + cn(2)21
...
EVMP1 = c1P1 + c2P1 + c3P1 + ... + cn(P)P1
...
```

Previous partition tree of depth-2 (i.e a tree whose nodes are set partitions) generalizes depth-2 boolean majority function to non-boolean setting and is an arithmetic circuit. At Depth-2 EVM set partitions per segment are summed up partwise (because all partitions within a segment should have equal number of parts which is the number of candidates per segment) and passed on to Depth-1 - in previous per segment partition identities summing vertically columnwise yields Depth-1 partition per segment. At Depth-1 index for candidate of maximum part/bucket size in segmentwise partition sum is computed and passed on to Depth-0. At Depth-0, single set partition is created which contains buckets per affiliation of the candidates. Maximum size part in Depth-0 set partition is the winning affiliation.

Whole depth-2 majority voting democratic process rests on the correctness of Axiom of Choice which avers: For a collection X of subsets, there always exists a choice function f such that for every set A in X, f(A) is an element of A. In previous example, at Depth-1, maximum() is the choice function applied across the board for all segment set partitions for choosing the winning candidate per segment (ruling out ties). It is readily obvious that each segment is a cluster of partitions and empirically distance between majority of EVM partitions of same segment (partition rank, rand index, mutual info) should not be abnormally high and reflects trend per segment.

By Berry-Esseen Central Limit Theorem, if vote for each candidate (ckpq) is a random variable, normalized sum of the votes for all candidates or average number of votes per candidate tends to normal distribution. In other words,

every part in each of the EVM partitions previously is a random variable which are summed up. This implies distances between any two EVM partitions (which is the function of distance between individual parts in two voting partitions) also tend to distance between two normal distributions (i.e almost a constant on the average). Most importantly, columnwise sum of parts leading to per segment partition in Depth-1 is also an averaged normal distribution.

Ramanujan's Congruences for EVM partitions imply, for certain size of electorate (5k + 4, 7k + 5, 11k + 6) number of possible voting patterns (partition number or number of possible partitions) abides by following congruences:

```
p(5k + 4) mod 5 = 0
p(7k + 5) mod 7 = 0
p(11k + 6) mod 11 = 0
```

References:

657.1 Ramanujan Congruences - https://en.wikipedia.org/wiki/Ramanujan%27s_congruences
657.2 Distance measures between partitions - Hamming Distance, Maximum Matching Distance, Variation of Information - https://pdfs.semanticscholar.org/d8a2/936cc4bcb8bda106e3b307104c3ee824b4a8.pdf
657.3 Microsoft ElectionGuard OpenSource EVM - https://blogs.microsoft.com/on-the-issues/2019/05/06/protecting-democratic-elections-through-secure-verifiable-voting/ - "...First, ElectionGuard provides each voter a tracker with a unique code that can be used to follow an encrypted version of the vote through the entire election process via a web portal provided by election authorities. During the process of vote-casting, voters have an optional step that allows them to confirm that their trackers and encrypted votes accurately reflect their selections...."
657.4 Ballot Marking Devices Cannot assure the will of voters - [Appel-DiMello-Stark] - https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3375755

---

658. (THEORY and FEATURE) Video EventNet Causality Graph Complexity Measures, Graph Minors, Merit of
Large Scale Visuals - 29 April 2019

1.This commit invokes networkx Graph API and creates NetworkX EventNet causality graph object for an example video. Connectivity of the Video EventNet is printed.
2.Example video in matrovska format which records the facebook profile of the author https://www.facebook.com/shrinivaasan.ka (containing NeuronRain commits twitter feeds) is comverted to mp4 by ffmpeg:
ffmpeg -i ExampleVideo_Facebook_GRAFIT_29April2019.mkv -strict experimental ExampleVideo_Facebook_GRAFIT_29April2019.mp4
3.Usually most videos have set of frames which are consecutive and closely related. These frames create straightline linear paths of vertices of degree 2 in the EventNet Graph underneath it.
4.These straightline paths can be coalesced or edge-contracted to single vertex forming Graph Minors of EventNet.
5.Connectivity of a Video is a necessary but not sufficient condition for merit.
6.Rationale for Video EventNet Connectivity as Merit Measure: Video of disconnected underlying EventNet causality has scattered non-coherent information which is not appealing to viewer.

---

659. (FEATURE) Music Pattern Mining - Music Synthesis by Automaton - Read from textfile - 30 April 2019

1.Hardcoded states2notes DFA has been removed and DFA is read from a textfile

NotesStateMachine.txt and
ast.literal_eval()-ed to a dictionary.
2.Scaling of notes have been commented and note values are magnified (multiplied by 1000). Waveform timeseries can be visually analyzed by loading .wav file to a player and displaying waveform oscilloscope in visual effects.
3.Logs for this have been committed to
testlogs/AudioToBitMatrix.log.MusicSynthesizer.30April2019

---
660. (THEORY and FEATURE) DeepLearning Convolution BackPropagation - handwriting recognition - utility function - 5 May 2019, 9 May 2019
- related to 159

1.DeepLearning_ConvolutionNetwork_BackPropagation.py has been changed to define a new utility function handwriting_recognition() which
loads two images of handwritings by CV2, thresholds them, finds contours of the two images and approximates the contours by Douglas-Peucker Polynomial (DP) and computes the hausdorff distance between the two DP approximate polynomials of the handwritten images.
2.testlogs/DeepLearning_ConvolutionNetwork_BackPropagation.log.5May2019 shows how 2 pairs of handwritten digits fare against hausdorff distance measure - 2 pictures of handwritten digit 1 (similar), 2 pictures of handwritten digits 1 and 8 (dissimilar)
3.Logs show DP polynomials of contours of 1 and 1 are close enough while DP polynomials of contours of 1 and 8 are far apart:
```
##############################################
Handwriting Recognition
##############################################
[[[1279  719]]]
[[[1276  719]]

 [[1279  719]]]
Distance between DP polynomials approximating two handwriting contours: (3.0, 0, 0)
[[[1279  719]]]
[[[  0 564]]]
Distance between DP polynomials approximating two handwriting contours: (1288.3578695378083, 0, 0)
```

4.Though image Contours and DP approximation are not traditional deep learning, previous contouring and DP approximation could as well be performed on convolution layers and maxpooling map layers of the two images as against images themselves.
5.Function handwriting_recognition() is supervised learner and takes two arguments - training (labelled) and test (unlabelled) images - and compares the unlabelled image to training image (printed or handwritten)
6.In the topological sense, each approximate DP polynomial for handwriting contours of same alphabet or digit are equivalent homeomorphic deformations in R^2. Handwritings of same person for similar text create a cluster of deformations which are close enough and create outliers for different persons.
7.Pattern grammars described earlier define grammatical rules similar to context free grammars for shape description. For example handwritten alphanumeric 'b' is defined as:

      <b> := <|> <operator> <o>

where <operator> could be <prefix> which tangentially attaches <|> before <o>. Following the convention above, a family or cluster of close-enough DP approximate polynomials fit in as operands within <> - e.g Coffee mug in 2 dimensions is homeomorphic to the letters 'D' and 'O' and all three (Mug, D, O) have close-enough DP Approximate polynomials of small hausdorff distances amongst them. Alternatively, Mug can be written as pattern grammar:

      <Mug> := <I> <prefix> <)>

and <I> and <)> are defined by DP polynomials.

References:

660.1 Ramer-Douglas-Peucker Curve Decimation Algorithm - approxPolyDP() - https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html
660.2 Coffee Mug is homeomorphic to Donut Torus - Graphic illustration - https://en.wikipedia.org/wiki/Homeomorphism

---
661.(THEORY and FEATURE) Music Pattern Mining - Weighted Automaton Learning from Music - 11 May 2019 - related to all sections on
Intrinsic Merit of Music

1.Previous sections described ascertaining merit of music by various analyses of waveforms e.g MFCC, learning DFA from music notes. Apart from these there are standard measures viz., zero crossings, chroma which quantify genre or merit of music (https://librosa.github.io/librosa/feature.html).
2.Of these learning a deterministic finite automaton from strings of music notes is a qualitative intrinsic merit measure of music which does not involve any statistical processing.
3.Learning DFA is a computational learning theoretic problem (PAC Learning, Angluin Model etc.,). Weighted Deterministic Finite State Automata are generalizations of DFA in which state transitions have an associated numeric weightage in addition to alphabets from a language.
4.For example, weighted DFA below has the weighted transitions:
      (state1, a, state2) - transition from state1 to state2 on symbol a of weight 1
      (state2, 2b, state3) - transition from state2 to state3 on symbol b of weight 2
5.Weighted automata are apt formalisms to represent music notes strings as DFA because notes have subdivisions - sharp and flat - which can be likened to weightage for state transitions (https://en.wikipedia.org/wiki/Musical_note - CDEFGABC - ascending and descending graphic).
6.For example string of music notes - "CD(sharp)DEEGAF(flat)F(sharp)" - and its accepting weighted music automaton could be:
      state1, C, state2
      state2, Dsharp, state3
      ...
      state10, Fflat, state5
      state6, Fsharp, state2
      ...
where sharp and flat are weights for the transition symbol.
7.Scikit Learn SPLearn (scikit-splearn) provides a weighted automaton learning framework based on Spectral Theory.
8.This commit implements a Music Weighted Automaton python code in MusicWeightedAutomaton.py which accepts a set of training string of music notes and fits them to a weighted automaton using scikit-splearn. It makes predictions later based on training data.
9.Learnt weighted automaton for music notes string is drawn as DOT graph file by graphviz in MusicWeightedAutomaton.gv and is rendered to a pdf - MusicWeightedAutomaton.gv.pdf
10.Weighting the transition in music notes has an another advantage: Some portions of the music waveform are more impressive and catchy which makes them to be listened again - https://en.wikipedia.org/wiki/Earworm - and transitions for these could have heavy weights.
11.Alphabets in sample music notes strings for scikit-splearn are numeric encoded as - A:0,B:1,C:2,D:3,E:4,F:5,G:6

References:

661.1 Weighted Automata Theory -

https://perso.telecom-paristech.fr/jsaka/CONF/Files/IWAT.pdf

---
662. (FEATURE) Intrinsic Merit of Music - Learning Weighted Automata from Music - Example - 14 May 2019

1.MusicWeightedAutomaton.py has been updated to define a new function audio_notes_to_samples() to read a music clip of certain duration by librosa, convert the waveform to notes, create notes string samples from them and return samples as word frequency dictionary and numeric encoded sample notes strings
2.SplearnArray is instantiated from encoded notes strings of variable size equal to number of columns in the notes matrix
3.Weighted automaton is learnt from these Sample dictionary and numeric encoded notes strings
4.automaton graph drawn in MusicWeightedAutomaton.gv.pdf shows a condensed weighted automata for an example music clip
5.Logs for this are committed to testlogs/MusicWeightedAutomaton.log.14May2019

---
663. (FEATURE) Named Entity Recognition - Conditional Random Fields - Update - 16 May 2019

1.NamedEntityRecognition_HMMViterbi_CRF.py has been changed to removed hardcoded PoS tagged sentences and instead to read states and observations from two files: NamedEntityRecognition_HMMViterbi_CRF.states and NamedEntityRecognition_HMMViterbi_CRF.observations which have the semi-PoS tagged sentence PoS states and sentence word observations respectively.
2.Emission Probabilities for each state are computed by a skew normal distribution which moves the mean from left to right of the sentence as states progress.
3.This skew normal distribution gives more weightage to respective state than others while each state is observed.
4.An example log in testlogs/NamedEntityRecognition_HMMViterbi_CRF.log.16May2019 shows the skewing of probability weights from left to right of the sentence
5.For example, Name-Verb-Object PoS tag is weighted as : First 1/3 of sentence has more weightage for emission probabilities of "Name",second 1/3 of sentence has more weightage for emission probabilities of "Verb" and third 1/3 of sentence has more weightage for emission probabilities of "Object"

---
664. (FEATURE) Spark 2.4 Structured Streaming - Windowed Stream - 21 May 2019

1.SparkGenericStreaming.java in java-src/bigdata_analytics/spark_streaming has been changed to do windowed structured stream if URL socket is false i.e data is read from plain socket stream.
2.New boolean flag "windowed" has been added to the java class as static member based on which Windowed streaming spark code is executed.
3.Windowed streaming spark code fragment uses readStream()/writeStream() facility in Spark to read data written over socket.
4.Netcat utility creates the socket server in port 8080.
5.Java lambda functions in FlatMapFunction tokenize the incoming stream of lines to words and organize them as word-timestamp rows dataset.
6.Next groupBy() parses word-timestamp dataset from previous lambda invocation and creates structured stream of tables of columns slidingwindow_timestamp-word-count.
7.Logs in testlogs/SparkGenericStreaming.log.21May2019 show dataframe tables for both urlsocket=false(windowed=true) and urlsocket=true boolean option settings.
8.CLASSPATH for compilation of SparkGenericStreaming.java (javac

SparkGenericStreaming.java) has been updated to:
CLASSPATH=/media/Ubuntu2/spark-2.4.0-bin-hadoop2.7/jars/spark-streaming_2.11-2.4.0.jar:.:/media/Ubuntu2/spark-2.4.0-bin-hadoop2.7/jars/scala-library-2.11.12.jar:/media/Ubuntu2/spark-2.4.0-bin-hadoop2.7/jars/spark-sql_2.11-2.4.0.jar:/media/Ubuntu2/spark-2.4.0-bin-hadoop2.7/jars/spark-core_2.11-2.4.0.jar:/media/Ubuntu2/jdk1.8.0_171/lib/jsoup-1.11.3.jar:/media/Ubuntu2/spark-2.4.0-bin-hadoop2.7/jars/spark-catalyst_2.11-2.4.0.jar:/media/Ubuntu2/spark-2.4.0-bin-hadoop2.7/jars/scala-reflect-2.11.12.jar:/media/Ubuntu2/jdk1.8.0_171/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
9.sparkgenericstreaming.jar is packaged as bin/jar cvf sparkgenericstreaming.jar *.class
10.SparkGenericStreaming is executed as:
/media/Ubuntu2/spark-2.4.0-bin-hadoop2.7/bin/spark-submit --jars /media/Ubuntu2/jdk1.8.0_171/lib/jsoup-1.11.3.jar --class SparkGenericStreaming --master local[2] sparkgenericstreaming.jar "https://twitter.com/search?f=tweets&vertical=news&q=Chennai&src=typd"

/media/Ubuntu2/spark-2.4.0-bin-hadoop2.7/bin/spark-submit --jars /media/Ubuntu2/jdk1.8.0_171/lib/jsoup-1.11.3.jar --class SparkGenericStreaming --master local[2] sparkgenericstreaming.jar "localhost" "8080"

for URLSocket and Windowed streaming respectively
11.Reference: Spark Structured Streaming Example in https://github.com/apache/spark/blob/v2.4.3/examples/src/main/java/org/apache/spark/examples/sql/streaming/JavaStructuredNetworkWordCountWindowed.java

---
665. (FEATURE) Named Entity Recognition - Conditional Random Fields - Skew Normal Distribution Update - 29 May 2019

1.Miscellaneous changes in NamedEntityRecognition_HMMViterbi_CRF.py for corrections to skew normal distribution which shifts mean of the distribution for each part of speech to the right by some constant multiples.
2.Skew normal distribution mean shift for each state (part of speech) is shown in NamedEntityRecognition_HMMViterbi_CRF.29May2019.png
3.Motivation for skew normal distribution mean shift from left to right for emission probabilities is: For some part of speech state, successive word observations in the sentence receive more weightage corresponding to the peak value at mean i.e one word in the sentence peaks at mean and is more likely candidate observation emitted for respective Part of Speech state.
4.In this example 5 words in the sentence peak for 5 part of speech states.
5.logs for this commit have been committed to testlogs/NamedEntityRecognition_HMMViterbi_CRF.log.29May2019

---
666. (THEORY) Approximation of Majority Voting, Streaming majority, Forecasts, Bertrand Ballot Theorem, Theoretical EVMs, Set Partitions - 4 June 2019

Bertrand Ballot Theorem states: In a majority voting involving 2 candidates A and B receiving p and q votes respectively of total electorate p+q,probability of candidate A always being ahead of candidate B in counting of votes is (p-q)/(p+q). Approximation of Majority voting has important applications in psephological forecast (pre-poll and post-poll surveys) which samples electorate to predict the winner. In pure complexity theoretic perspective, as has been mentioned earlier, boolean majority gate can be alternatively represented by a set partition of 2 candidate buckets indexed 0 and 1 and maximum size candidate bucket index is the output of set partition boolean majority gate. It is also worth mentioning here that approximating a majority gate of n inputs by a much smaller gate of m inputs (m << n) is the complexity theoretic equivalent of forecasting a real-world majority voting of 2 candidates by sampling m voters

from an electorate of size n. It is obvious that process of counting of votes creates instantaneous snapshots of the electorate set partition and any m sized (less than n) voter set partition snapshot is equivalent to a sampling of total electorate. In terms of set partition definition of majority gate, both buckets of voter set partition 0 and 1 grow over time and at any instant the set partition of total size m is a sampling of n voters.Votes for 2 candidates 1 and 0 can be drawn as timeseries polynomial curves (votes versus time) and intersection points of 2 curves are trend reversals contributing to the probability (p-q)/(p+q). Complexity theoretic majority gate assumes all the inputs at the leaves are immediately available which is not possible in approximation in which only m of n votes are known. Majority approximation gate predicts output of majority gate of n inputs by a majority gate of m inputs for m << n with some error probability. By Bertrand Ballot Theorem probability that a majority approximation gate of m inputs (snapshot at some time point) has the same output as an exact majority gate of n inputs is (p-q)/(p+q) for total electorate p+q=n and p voters for 0 or 1 and q voters for 1 or 0. If m=a+b for a voters for 0 or 1 and b voters for 1 or 0 in the sample approximation, sample betrand probability is assumed to be proportional to bertrand probability of total electorate:

```
      (p-q)/(p+q) = k*(a-b)/(a+b)
      (p-q)/n = k*(a-b)/m
      (p-q) = n*k*(a-b)/m
```

which extrapolates the sample to the entire electorate.

References:

666.1 Bertrand Ballot Theorem - https://en.wikipedia.org/wiki/Bertrand%27s_ballot_theorem
666.2 Digging into Election Models - https://windowsontheory.org/2020/10/30/digging-into-election-models/

---

667. (FEATURE) Social Network Analysis, People Analytics, PDF file parsing - 13 June 2019

---

1. Changed SocialNetworkAnalysis_PeopleAnalytics.py to parse PDF files of social network profiles by PyPDF2 which was not working earlier.
2. PyPDF2 expects PDF files to be written by PDF writer or otherwise blank text is extracted
3. As example CV of self written by LaTeX2PDF has been parsed to print profile text to testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.13June2019
4. Recursive Lambda Function Growth import has been commented because of WordNet error:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/dist-packages/wn/__init__.py", line 11, in <module>
    from wn.info import InformationContentSimilarities
  File "/usr/local/lib/python2.7/dist-packages/wn/info.py", line 7, in <module>
    from wn.reader import parse_wordnet_ic_line
  File "/usr/local/lib/python2.7/dist-packages/wn/reader.py", line 22
    offset, lexname_index, pos, n_lemmas, *the_rest = columns_str.split()

SyntaxError: invalid syntax
```

5.LinkedIn PDF Profile has been updated by linkedin pdf export

---

668. (THEORY) Computational Geometric Factorization - Average case Sequential Optimization, Stirling Approximation of Gamma Function,
Sum of binary search times of each tile and some observations - related to 605 - 14 June 2019

```
------------------------------------------------------------------------------
-----------------------------------------------------
(*) Average number of tiles between two approximate prime factors m and (m+1)
found by tile summation ray shooting:
           = (m+1)/[kloglogN - m - 1] - m/[kloglogN - m]
because number of tiles till m-th prime factor = n = m/[kloglogN - m] for
m=1,2,3,...,kloglogN
(*) Thus total average sequential time to sweep binary search tiles in
N/kloglogN spacing between 2 consecutive approximate factors found by ray
shooting:
      = O([(m+1)/[kloglogN - m - 1] - m/[kloglogN - m]] * logN)
(*) Maximum number of tiles in N/kloglogN spacing occurs by minimizing
denominator and setting m=kloglogN - 2:
      = O([(kloglogN - 2 + 1)/[kloglogN - kloglogN + 2 - 1] - (kloglogN -
2)/[kloglogN - kloglogN + 2]])
      = O(kloglogN - 1 - (kloglogN - 2)/2)
      = O(kloglogN)
=> Maximum time to binary search tiles in N/kloglogN spacing = O(loglogN * logN)
=> Each of O(loglogN) spacings between approximate factors can be searched in:
      O((loglogN)^2*logN)
average case sequential time with no necessity for parallel processing.
(*) If each tile segment arithmetic progression is searched individually and
sequentially, sum of binary search times for first k adjacent tile segments is:
      log N/(1*2) + log N/(2*3) + log N/(3*4) + ... + log N/(k*(k+1))
which can be written as:
      log N - (log 1 + log 2) +
      log N - (log 2 + log 3) +
      ...
      log N - (log k + log (k+1))
by logarithmic identities and becomes:
      k log N - (log 1 + log (k+1) + 2 log k!)
But Gamma function is:
      Gamma(x+1) = x!
and by Stirling approximation:
      log k! ~= log Gamma(k+1) = (k+0.5)*log k - k -1 + 0.5*log 2*pi
Substituting in the summation:
Total time for binary searching k adjacent tile segment arithmetic progressions
= k log N - log (k+1) - (2k + 1)log k + 2(k+1) - log (2*pi)

A variant approximation of this search time summation has been described in
section 5 of
http://sourceforge.net/projects/acadpdrafts/files/DiscreteHyperbolicPolylogarith
micSieveForIntegerFactorization_PRAM_TileMergeAndSearch_And_Stirling_Upperbound_
updateddraft.pdf/download

if k is O(loglogN) from previous derivation of maximum number of tiles between 2
approximate prime factors, in average case atleast one factor can be found in
average time:
      loglogN * log N - log (loglogN + 1) - (2loglogN + 1)logloglogN +
2(loglogN+1) - log (2*pi)
      <= O(logN * loglogN)

References:
----------
668.1 Stirling Approximation of Factorial and Gamma Function -
https://en.wikipedia.org/wiki/Stirling
%27s_approximation#Stirling's_formula_for_the_gamma_function

------------------------------------------------------------------------------
-----------------------------------------------------------
669. (THEORY) Computational Geometric Factorization, Ramsey Coloring and
Arithmetic Progressions in the Tile Segments, Van Der Waerden
Numbers - related to 636 - 19 June 2019
------------------------------------------------------------------------------
```

By Van Der Waerden theorem, for any positive integers r and k there is number N such that if the numbers {1,2,3,...,N} are colored, each with one of the r different colors then there are atleast k integers in arithmetic progression whose elements are of same color. Computational Geometric Factorization is based on rectification of hyperbolic curve into straightline or rectangular polygonal tile segments (of dimensions 1 * length_of_segment) and reducing the factorization problem to parallel RAM planar point location of factor points (i.e hyperbolic curve is rectified to a polygon containing arithmetic progression of products of ordinates as its rectangular faces, parallel planar point location finds a rectangular face of the polygon having the factor point N=pq in polylogarithmic time and the rectangular face arithmetic progression having factor point is binary searched in logarithmic time - thus in NC) in this rectified polygon. Each hyperbolic tile segment obtained by rectification are construed to be arrays of products of ordinates or arrays of ordered pair of ordinates in x-axis and y-axis. Unsorted Concatenation of these tile segments of ordered pair of ordinates has the x-axis or y-axis sorted ascending from 1 to N (or sqrt(N) to N). Since arrays of product or ordered pair of ordinates in each tile segment have one of the ordinates constant, each tile segment array is an arithmetic progression of ordinate products elements of which are colored monochromatically. Factor point location by parallel RAM sorting based on product of ordinates of ordered pairs in tile segments sorts the product of ordinates but shuffles the x-axis ordinate in ordered pair notation which was strictly ascending in unsorted concatenation. Sorted concatenation of tile segments of product of ordinates is the union of subsets of elements in each monochromatic arithmetic progression and creates a (pseudo)random coloring of sorted sequence of integers in the range {N-delta,N+delta}. In other words rectification of hyperbolic curve just inverts the Van Der Waerden theorem - instead of finding monochromatic arithmetic progressions in a colored sequence, it forms a mergesorted sequence from monochromatic arithmetic progressions in tile segments. Since rectification of hyperbolic curve for every integer creates unique set of tile segments, previous inverse of Van Der Waerden coloring is also unique.

---
670. (THEORY) Computational Geometric NC-PRAM Factorization, Tile Summation Ray Shooting Queries and Gamma approximation, Iterated addition for sum of tile segment lengths - 27 June 2019 - related to 506, 668

Tile summation ray shooting queries:

Sum of lengths of first n tile segments of pixelated hyperbolic curve derived previously:

N/[1*2] + N/[2*3] + ... + N/[n*(n+1)] = Nn/(n+1)

which can be equated to sum of distances between first m prime factors by Hardy-Ramanujan Theorem:

Nn/(n+1) = mN/kloglogN

=> n = m/(kloglogN - m) = number of tiles till m-th prime factor.

Gamma approximated tile summation ray shooting queries:

Alternatively, sum of lengths of tiles obtained approximately from sum of binary search times by Gamma function previously can be equated to Hardy-Ramanujan normal order estimate:

2^(n log N - log (n+1) - (2n + 1)log n + 2(n+1) + log (2*pi)) = mN/kloglogN

(or) (n log N - log (n+1) - (2n + 1)log n + 2(n+1) + log (2*pi)) = log(mN/kloglogN)

and solved for number of tiles n that have to be summed till m-th prime factor. Total Binary Search time has been raised to exponent to get approximate total length searched. Solving logarithmic equation previously might require some

logarithmic series approximation resulting in a cubic equation or higher degree in n depending on number of terms chosen from logarithmic series.

An important fact ignored so far is the summing of tile lengths while doing ray shooting queries which has to be parallelized too. But tile summation is an iterated addition of O(N) logN bit integers (tile lengths are rounded off) which is in NC1. Thus Computational Geometric NC-PRAM planar point location factorization invoking tile summation ray shooting query oracles is in NC again. It can be noted that most other ray shooting query algorithms do not have this iterated addition of tile lengths requirement.

References:

670.1 Iterated addition of n n-bit integers is in NC1 - Theorem 8.9 - https://courses.cs.washington.edu/courses/cse532/08sp/lect08.pdf

---
671. (THEORY and FEATURE) Intrinsic Merit of Music - MFCC analysis and zero crossing rate - 27 June 2019

1.AudioToBitMatrix.py MFCC function has been updated to scale MFCCs by sklearn and print mean and variance of MFCCs of an audio/music waveform
2.zero crossing rate of waveform is also printed which is measure of modulations in music.
3.One of the ways to measure Music merit and Music Genre or Emotion Classification is by clustering music waveforms by distance similarity of MFCCs.
4.MFCC similarity can be used as recommender system where a listener is recommended music of similar MFCCs based on historic listening habits.
5.MFCCs of music waveform drawn as images (e.g by librosa.display.specshow()) can be compared for similarities.

Reference:

671.1 Music Information Retrieval and MFCCs - https://musicinformationretrieval.com/mfcc.html

---
672. (FEATURE and THEORY) Intrinsic Merit of People/Professional Profiles - People Analytics - Tenure Histogram - 28 June 2019

1.People Analytics implementation has been updated to print Work and Academic Tenure stints as array (histogram or integer partition)
2.This essentially makes tenures as a set partition analytics problem because job and academic stint time durations can be picturized as histograms.
3.Tenures represented as Set Partitions can be quantified by set partition measure of partition rank:
      Length of largest part in partition - Number of parts in partition
4.In the context of People Analytics, previous rank measure is quite applicable practically as attrition metric because:
      4.1 High partition rank implies the professional profile has spent large duration in an organization or number of tenure switches are less (number of parts)
      4.2 Low Partition rank implies the professional profile has low duration per organization or high number of tenure switches
5.Tenure Partition can be printed as Ferrer or Young Diagram and size of Durfee Square of this tenure partition is also a measure of average time spent per organization and thus an attrition metric
6.Recursive Lambda Function Growth wordnet error has been resolved (pywsd is not used) and import uncommented.

7.logs for this have been committed to testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.28June2019

---

673. (FEATURE) People Analytics - Partition Rank Attrition Metric of Tenure Histogram - 29 June 2019

1.People Analytics implementation has been updated to compute partition rank described previously (Crank of a Partition - Freeman Dyson - https://en.wikipedia.org/wiki/Crank_of_a_partition) of tenure histogram set partition of a person's professional and academic profile.
2.Recursive Lambda Function Growth algorithm has been applied to text of the profile again.Closeness and Betweenness Centralities rankings are quite relevant than other Centrality measures.
3.Logs for this have been committed to testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.29June2019

---

674. (FEATURE) People Analytics - Rank Correlation based Attrition Model of Tenure Histogram - 1 July 2019

---

1. People Analytics implementation has been updated to include a function to compute Correlation Coefficients of Statistical Dependence between vectors of Tenure Histograms.
2. Tenure Histogram is deemed as a set of vectors of the form (organization/designation, income, duration) which captures the crucial factors of a tenure
3. New function tenure_partition_rank_correlation() has been implemented to compute rank correlations between each pair of variables of a tenure:
	3.1 Designations versus Remunerations
	3.2 Designations versus Durations
	3.3 Remunerations versus Durations
4.An example tenure histogram vectors of previous format has been analyzed (presently not parsed from profiles) for Kendall-Tau Rank correlation coefficient and logs are in testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.1July2019
5.Previous correlations can be interpreted as below:
	5.1 Designations versus Remunerations - If positive or high, professional profile has increasing career graph along with increasing remunerations (expected). If negative or low, professional profile has increasing career graph vis-a-vis reduced remunerations (not expected - implies profile preferred more challenging roles than remuneration) and vice-versa (not expected - implies profile preferred better remuneration than designations)
	5.2 Designations versus Durations - If positive or high, profile has increasing career graph vis-a-vis durations of tenure (expected). If negative or low profile has increasing career graph vis-a-vis decreasing durations per tenure (not quite expected - increased designations were not appealing to profile and person decided to leave) and vice-versa (not quite expected - decreasing responsibilities of the role were preferred by the profile which decided to stay longer)
	5.3 Remunerations versus Durations - If positive or high, profile has increasing emoluments versus increasing durations per tenure (expected). If negative or low profile has increasing remunerations against decreasing durations per tenure (not quite expected - profile was more inclined to salary) and vice-versa (not quite expected - profile stayed longer per tenure despite decreasing benefits).
6. Previous interpretations facilitate inferences (in parentheses) of why attritions could have happened.
7. From logs following positive and negative correlations are found and can be interpreted based on (5):

Kendall Tau Rank Correlations - Designations and Remunerations: tau= 0.7142857142857143 , pvalue= 0.0301587301587
Kendall Tau Rank Correlations - Designations and Durations: tau= -0.09759000729485331 , pvalue= 0.7612636496364197
Kendall Tau Rank Correlations - Durations and Remunerations: tau= -0.19518001458970663 , pvalue= 0.5434238636256696

---

675. (FEATURE) Urban Planning Analytics - Analysis of Remote Sensing GIS Imagery of Urban Sprawls - 3 July 2019

1.Analyzing Builtup land, Water bodies etc., of an urban sprawl is crucial for urban planners and is an apt application of image analysis. Urban sprawl is defined as expanse of urbanization in suburbs and is not measured by governance limits. For example rankings of metropolitan areas in India is: https://en.wikipedia.org/wiki/List_of_metropolitan_areas_in_India ("...

Rank	Metropolitan area	State/Territory	Population	Area (in km2/sq mi)
1	Central National Capital Region	Delhi, Haryana, Uttar Pradesh	25,735,000 (2016)[3]	2,163 (835)[3]
2	Mumbai Metropolitan Region	Maharashtra	20,800,000 (2005)[4]	4,354 (1,681)[5]
3	Kolkata metropolitan area	West Bengal	14,720,000 (2001)[6]	1,851 (715)[7]
4	Chennai metropolitan area	Tamil Nadu	13,300,253 (2011)	1,189 (459)...")

which is measure of urbanization.

2.Convex Hull based on Scipy Spatial for urban sprawls has already been implemented in ImageGraph_Keras_Theano.py for GIS images from SEDAC (http://sedac.ciesin.columbia.edu/mapping/popest/gpw-v4/) which is also invoked in Drone Online Shopping Delivery NeuronRain usecase
3.A Machine Learning (Random Forests) and entropy based analysis of Chennai Urban Sprawl - https://www.mdpi.com/1099-4300/19/4/163/pdf - studies the growth of built area and dwindling of water bodies from 1991 to 2016 and predicts scenario for 2027. Images referred to in this article are drawn from various Remote Sensing satellite sources (LandSat, ISRO-Bhuvan Geo Platform - https://bhuvan.nrsc.gov.in/bhuvan_links.php etc.,) and are great references for Chennai Urban Sprawl Land Usage,Water bodies,Greenery etc.,.
4.Images from https://www.mdpi.com/1099-4300/19/4/163/pdf have been extracted and are analyzed for Red, Green and Blue Channels in this commit. New python function analyze_remotesensing_RGB_patches() has been implemented for this which invokes OpenCV2 split() of the image to Red,Green,Blue channels.
5.Rationale for Red-Green-Blue Channel split of Urban Sprawl is most of the images are color coded in Red, Green and Blue for Built Area, Greenery and Water bodies respectively and splitting to respective color channels layers the image.
6.Each color channel - Red. Green and Blue - is a NumPy ndarray which is flattened and histograms are computed for each color channel.
7.Color channel images are written to JPG files of respective suffixes and read again to compute Numpy histogram
8.Each color channgel image is grayscale only and percentage of white pixels of value 255 is the sum total size of respective color patch in image.
9.Scikit Learn Image has extract_patches_2d() function - https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.image.extract_patches_2d.html - which can segment an image to patches or clusters of similar features. But a native implementation of patch extraction has been preferred for color coded GIS Remote Sensing images.
10.Ratio of white pixels to total number of pixels is computed from color channel image histograms as:
Size of 255 (white) bucket in image ndarray histogram / Sum of sizes of all buckets in image ndarray histogram

11.Images extracted from https://www.mdpi.com/1099-4300/19/4/163 and their color channel images have been committed to ImageNet/testlogs/RemoteSensingGIS/
12.Logs for this are committed to testlogs/ImageGraph_Keras_Theano.log.RemoteSensingGIS.3July2019 which print the percentage of water bodies (blue), built area (red) and greenery (green) inferred from color channels
13.Alternatively, image ndarray itself could have been mapped to histograms and ratio of respective pixel color value buckets could have been computed. But exact color value for each color coding is not known and color channels grayscale the images and just number of white pixels (255) have to be found.

---

676. (FEATURE) Image Analytics - ImageNet Keras-Theano - Random Forest Classification wrapper for images - 4 July 2019

1.New wrapper function has been implemented which invokes scikit learn Random Forests - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier.predict to classify set of test images based on
training images and training labels.
2.This wrapper function random_forest_image_classification() pre-processes the training and test images by loading to CV2 and flattening the
image ndarray and then reshaping and transposing it so that scikit learn accepts.
3.First the random forest classifier which is a forest of lot of decision trees which vote to decide the class label of a sample, is trained by fit() for training images and then test images are labelled by predict()
4.Logs for this are committed to ImageGraph_Keras_Theano.log.RandomForests.4July2019
5.Scikit learn provides train_test_split function which divides set of samples into training and test images based on which fit() and predict() are invoked.
6.Alternative version without training and test split also follows the above as in logs.

---

677. (FEATURE) Image Analytics - ImageNet Keras-Theano - Unsupervised Recursive Gloss Overlap classifier based on ImageNet - 5 July 2019

1.As opposed to Random Forest classifier implemented earlier based on scikit learn, NeuronRain unsupervised graph theoretic Recursive Gloss Overlap classifier has been applied for some test images
2.Recursive Gloss Overlap classify() function has been invoked from imagenet_imagegraph() which was earlier only creating text graph from Keras predictions text for the image.
3.Example Dense subgraph (core number) classifications for these images in the logs at testlogs/ImageGraph_Keras_Theano.log.ImageNetClassifier.5July2019 are reasonably accurate and classify the images better. No training labelled data are required.

---

678. (THEORY) Majority Hardness Amplification Lemma, Noise Sensitivity, Sensitivity, Block Sensitivity, Condorcet Jury Theorem - 6 July 2019, 6 October 2019 - related to 517

Hardness of a boolean function has been defined previously as how hard computationally it is to compute the function with a circuit of size s:
if Pr(C(x) != f(x)) = delta where C(x) computes boolean function f(x),

then f(x) delta-hard.
Hardness amplification for Majority + VoterSAT boolean composition has been derived earlier as:

Hardness Amplification for Majority+Voter Composition <= 1/(delta + or - error)*(1/2-1/3*pi) + 1/pi + 6k*pi*(delta + or - error)^0.5
for delta-hard voter SAT boolean functions (i.e Probability of a circuit incorrectly computing a voter SAT = delta)

Noise Sensitivity of Majority function has been previously mentioned as O(1/sqrt(n*epsilon)) where epsilon is probability of flip per bit. Therefore epsilon is nothing but probability that a voter boolean function is incorrectly computed and sent as input to Majority function. This implies Pr(f(x) != C(x)) = delta = epsilon = hardness of all voter functions.

In all sections related to majority voting and majority+voterSAT boolean composition, measure of sensitivity used is Noise Sensitivity which is Pr(f(x) != f(y)) for 2 correlated boolean strings x and y in {0,1}^n and sensitivity and block sensitivity measures have been ignored. Lemma 15 in reference 678.1 provides a simple boolean hypercube random walk proof of relation between sensitivity and Noise sensitivity from which:

NoiseSensitivity(f) <= 2*delta*sensitivity for probability of a bit flip = delta = epsilon
=> NoiseSensitivity(Majority) = O(1/sqrt(n*epsilon)) <= 2*delta*sensitivity

Since sensitivity conjecture (block sensitivity <= poly(sensitivity)) has been proved recently, sensitivity in previous inequality can be written as a function of block sensitivity. Majority Hardness Amplification ratio then can be written as:

$$\frac{\text{Hardness of Maj+voter composition}}{\text{Hardness of voter function}} = \frac{[c/\sqrt{n*\text{delta}}] +/- \text{randomerror}}{\text{delta}}$$

$$\leq \frac{2*\text{delta}*\text{sensitivity}}{\text{delta}} +/- \text{error} \leq 2*\text{sensitivity} +/- \text{error}$$

which upperbounds the maximum hardness amplification of Majority+VoterSAT composition almost as twice of sensitivity of majority function.

Recent versions of Condorcet Jury Theorem by [Black] and [Ladha] lay emphasis on correlated voting in which voters are not independent but are statistically dependent and swayed by peer opinions en masse (this is the most prevalent "Herding" phenomenon in real world voting and in social networks), and from [Ladha] performance of majority voting decreases as correlation increases. In boolean majority function this herding correlation of bits is formalized by Block sensitivity and flipping a contiguous block of bit positions (contiguous set of voters flip their decisions) changes the outcome of the majority function. From [Ladha] more block sensitive (more correlated voters) majority function is less effective it is and towards less goodness it tends to.

NOTE: Throughout all sections on Majority Voting + VoterSAT boolean composition and its hardness amplification, Sensitivity measures have been assumed as main contributors to hardness because by definition a hard boolean function is not 100% correctly computable by a circuit and sensitivity to input correlation (flipped bits) is the major factor preventing the correct computation by circuit. There could be other factors intrinsic to boolean function itself too apart from sensitivity measures which prevent a circuit from being perfect characterized by (+/-) error term above - randomized decision tree evaluation being one of them.

References:

678.1 Smooth Boolean functions are easy - [Parikshit Gopalan,Noam Nisan,Rocco A.Servedio,Kunal Talwar,Avi Wigderson] - Lemma 15 - http://www.math.ias.edu/~avi/PUBLICATIONS/GopalanNSTW.pdf

678.2 Proof of Sensitivity Conjecture - [Huang] -
http://www.mathcs.emory.edu/~hhuan30/papers/sensitivity_1.pdf
678.3 Journal of Economic Behavior & Organization Volume 26, Issue 3, May 1995, Pages 353-372, Information pooling through majority-rule voting: Condorcet's jury theorem with correlated votes - [Krishna K Ladha] - https://www.sciencedirect.com/science/article/pii/016726819400068P - "...Does majority-rule voting steer an imperfectly informed assembly of people towards the full-information outcome? Condorcet's jury theorem provides an affirmative answer under certain conditions. A key condition is that the votes be statistically independent; however, it is unrealistic, and hence, unacceptable. This paper generalizes the jury theorem to certain general models of correlated voting, viz., normal, hypergeometric and Polya distributions. The paper proves that the effectiveness of majority-rule voting decreases as the correlation between votes increases. Potential applications are indicated...."
678.4 The Theory of Committees and Elections. Cambridge University Press - [Duncan Black] - https://www.springer.com/gp/book/9780792381105

----
679. (FEATURE and THEORY) Fraud Analytics - Principal Component Analysis and KMeans++ clustering of an example Credit Card Transactions Dataset - 8 July 2019

1.This commit implements a rudimentary primitive fraud analytics Python class
2.Fraud Analytics in the context of NeuronRain has already been implemented to some extent in USBmd as wireless LAN traffic analytics and ftrace based analysis which is a low level intrusion fraud analytics
3.To broaden the purview new file FraudAnalytics.py has been added to AstroInfer repositories and a basic fraud analysis function for credit card data set from https://www.kaggle.com/mlg-ulb/creditcardfraud/downloads/creditcardfraud.zip/3 has been implemented - creditcard_fraud_analytics()
4.Traditional algorithm to analyze frauds is to find the outliers in a dataset which are deviations from the normal.
5.This function applies kmeans++ clustering from scikit learn to classify the previous kaggle creditcard transactions dataset into two clusters 0 (normal) and 1 (potential fraud).
6.Kaggle dataset is in CSV format and first 500 rows of it are read by python CSV reader and a NumPy ndarrays are created from it which splits the transactions into 150 rows training set and 350 rows test set.
7.Columns of the CSV file are Time, 28 columns which obfuscate the identity of the cardholder, Amount, label 0 or 1 (normal or fraud) which are real transactions happened in 2013 and each column is considered as a dimension for tranasaction vector
8.Classifying high dimensional data has variety of clustering algorithms including kmeans++ (https://scikit-learn.org/stable/modules/clustering.html)
9.To simiplify high dimensional data, convention followed is to apply Principal component Analysis to reduce the number of dimensions and then do clustering. e.g
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
,
https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html#sphx-glr-auto-examples-cluster-plot-kmeans-digits-py
10.Therefore previous credit card transactions vectors are clustered by fit() of the train dataset followed by predict() for test dataset.
11.Both predictions- with and without PCA - have been printed for 350 rows test dataset (0 - normal,1 - fraud):

```
kmeans estimator pred() labels: [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
kmeans estimator pca pred() labels: [1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

12.Another alternative is to consider the dataset as multiple concurrent timeseries and plot the curves per cardholder identity. Spikes in timeseries are outliers which could be potentially fraudulent.

---

680. (FEATURE) Intrinsic Merit of Audio - Speech Recognition Integration and Graph Tensor Neuron Merit of the recognized speech text - 9 July 2019

1.AudioToBitMatrix python implementation has been updated to include a new function which reads an audio WAV file of human speech and invokes speech recognizer (PocketSphinx) on the audio to get speech text and finds the Recursive Lambda Function Growth intrinsic merit of the speech text
2.Audio merit is split into two - Speech and Music
3.An example speech audio file (voice of author - K.Srinivasan - explains the 6 types of intrinsic merit analytics implemented in NeuronRain AstroInfer - text,video,speech,music,people-experiential-intrinsic, people-attrition or HR analytics) is used for speech recognition - music_pattern_mining/testlogs/AudioGraphExample_SpeechRecognition_2019-07-09-103018.wav
4.Prerequisite packages to be installed are Python SpeechRecognition, PocketSphinx, Swig, ALSA development library, PulseAudio development library
5.PocketSphinx recognized speech text is printed (though not accurate) and Recursive Lambda Function Growth grow_lambda_function3() is invoked on recognized speech text to create textgraph, grow lambda functions from textgraph random walks or cycles and print various merit quantities
6.This assumes Speech Recognizer is accurate to large extent.

---

681. (THEORY and FEATURE) Intrinsic Merit of Music - MFCC Earth Mover Distance similarity based Music Recommender - 12 July 2019

1.Mel Frequency Cepstral Coefficients of a music waveform are concise measures of music genre and have been included in AudioToBitMatrix implementation
2.Music Similarity Measure based on MFCC spectral signatures is described in http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.8524&rep=rep1&type=pdf which clusters music files based on earth mover distance between MFCCs of any two music files
3.Earth Mover Distance also known as Wasserstein distance is a function in SciPy - https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wasserstein_distance.html
4.Earth Mover Distance is the amount of work required to transform one distribution (set partition or histogram) to another distribution (another set

partition or histogram) and thus is distance measure between two set partitions.
5.A miniature music recommender systems has been implemented based on MFCC earth mover distance which analyzes pair of music files for MFCCs and finds the earth mover similarity between the mean of MFCCs of two music files.
6.Pairwise earth mover distances (EMD) are sorted by values of similarity (low EMD values imply high similarity)
7.3 music files have been pairwise analyzed for MFCC EMD similarity for example (2 files of Bach Sonata and 1 file of Veena Instrumental - S.Balachander's Amritavarshini - respective copyrights attributed)
8.Though number of files analyzed is small, Bach Sonata files have high EMD similarities compared to Bach and Veena Instrumental suggesting a clustering of oriental and western classical music in terms of spectral analysis.

---
682. (THEORY) KRW Conjecture, Majority + VoterSAT Boolean Function Composition, BKS Conjecture, Computing any function by Parity - 12 July 2019, 17 July 2019, 18 July 2019, 19 July 2019, 22 July 2019 - related to 365, 368 and all sections on Majority Voting
---
Boolean function composition of Majority and any Voter boolean function and its relevance to KRW Conjecture (Depth(f+g) ~= Depth(f) + Depth(g) or FormulaSize(f+g) ~= FormulaSize(f) * FormulaSize(g)) and Communication Complexity - KW Relations - which is an open problem - have been described in detail earlier and an inequality below for lower bound on depth of circuit computing Majority+VoterSAT composition has been derived assuming KRW conjecture:

D(Maj + Voter) = CommunicationComplexity(R(Maj,Voter)) >= 5.3*logm + n - O(m*logm/n)

for m voters having voter SATs of n variables per voter.

KRW Conjecture for composing any boolean function f and parity has been proved by [Hastad] and [Dinur-OrMeir]. Proving the general case of KRW conjecture implies P != NC1 (not all sequential polytime algorithms are logdepth parallelizable). If every Voter boolean functions are computable by circuits of AND, OR and Parity function, previous composition of Majority + VoterSAT is reduced to Majority + function_of(Parity) composition replacing Voter SAT circuit by a circuit of AND, OR and parity gates. Recent result in reference 682.2 delves on upperbounds and lowerbounds for computing symmetric and threshold boolean functions by parity gates (Razborov approximation of AND and OR by low degree polynomials on F[2] and composing them). Voter decision function can also be thought of as a threshold function (weighting the variables). Stability of Interview TQBF has been derived earlier in the context of BKS Conjecture by formalizing a question-answering in machine learning and learning theory as a weighted linear threshold function on answers to questions and their respective weights. Generalizing proof of KRW Conjecture for composing a function and parity to composition of a function and function_of(parity) proves the general case of KRW Conjecture. In reference 682.4 previous Communication Complexity (CC) Lower bound of composition has been refined to CC(KWf + Un) ≥ log L(f) + n − 2 log m − 10 implying CC(Majority+Voter) ≥ 5.3*log m + n − 2*log m − 10 assuming a universal Voter relation.

Assuming Voter SATs are indeed Linear or Polynomial Threshold Functions by which each voter scores the candidate to vote (0 or 1) based on some variables (e.g answers to questions) and votes for or against based on if the evaluated LTF or PTF exceed threshold or not, is a learning theoretic perspective as opposed to SAT based decision making. An experimental proof of BKS Conjecture (Majority is least stable) previously is restricted to LTFs only. Polynomial Threshold Functions have a circuit representation of depth 2 consisting of Parity and Threshold gates - layer 1 of parity gates for each monomial and layer 2 of a single threshold gate. From classic result of [Minsky-Papert], any computable function can be written in terms of threshold function. Circuit representation of Polynomial Threshold Functions are thus helpful to compute any Voter Decision

Function in terms of Parity and Threshold gates. By Chow's Theorem for threshold functions, any 2 functions f (LTF) and g (any function) are equivalent if the degree-0 and degree-1 Fourier coefficients in Fourier expansion of f and g are equal.

Assuming KRW Conjecture, Formula Size of the Majority + VoterSAT boolean composition is:
FormulaSize(Majority + VoterSAT) ~= FormulaSize(Majority) * FormulaSize(VoterSAT)
=> FormulaSize(Majority + VoterSAT) ~= m^5.3 * FormulaSize(VoterSAT)
(Formula Size implies the circuit composition has fanout 1)
It is known that set of languages of Turing Machine deciding an input in time(t) is contained in set of languages accepted by a circuit of size(tlogt). Combining this to the previous composition formula size bound assuming KRW Conjecture an estimated runtime of a Turing Machine equivalent to Majority + VoterSAT composition can be obtained by solving for t in the equality:
tlogt = m^5.3 * FormulaSize(VoterSAT)
(or) t^t = 2^(m^5.3 * FormulaSize(VoterSAT))

if t=O(FormulaSize(VoterSAT)):
logt = k*m^5.3 (or) t = 2^(k*m^5.3) for some constant k which is an EXPTIME Turing Machine (i.e exponential in number of voters) for Majority Voting Circuit Composition, implying unbounded depth.

PTF can encode a CNF MAX k-SAT - Example:
(x1 V !x3 V x4) /\ (x2 V x3 V x5) /\ (x1 V !x2 V x3) can be encoded as 3-sparse PTF (x1)(1-x3)(x4) + (x2)(x3)(x5) + (x1)(1-x2)(x3) >= k

References:

682.1 KRW Conjecture results survey - [OrMeir] - https://simons.berkeley.edu/sites/default/files/docs/10492/presentation.pdf
682.2 Parity helps compute Majority - Theorem 4 and Theorem 8 on computing symmetric boolean functions and threshold functions by parity - [Igor C. Oliveira∗ Department of Computer Science University of Oxford, Rahul Santhanam† Department of Computer Science University of Oxford, Srikanth Srinivasan‡ Department of Mathematics IIT Bombay] - https://eccc.weizmann.ac.il/report/2019/073/download
682.3 Proof of KRW Conjecture for composing a function and parity - [Dinur] - http://drops.dagstuhl.de/opus/volltexte/2016/5841/
682.4 Improved Composition Theorems for Functions and Relations - [Koroth-OrMeir] - Theorem 8 - http://drops.dagstuhl.de/opus/volltexte/2018/9452/pdf/LIPIcs-APPROX-RANDOM-2018-48.pdf
682.5 Toward Better Formula Lower Bounds: An Information Complexity Approach to the KRW Composition Conjecture∗ - [Dmitry Gavinsky† Or Meir‡ Omri Weinstein § Avi Wigderson¶] - http://www.math.ias.edu/~avi/PUBLICATIONS/GavinskyMeWeWi2016.pdf - Section 1.2.1 - Formula size of Composition of a function and Parity by [Andreev] and [Hastad] - "...L(g # ⊕n) = L(g) · n^2/poly log m = L(g) · L(⊕n)/ poly log(m),.."
682.6 Linear and Polynomial Threshold Functions - http://www.contrib.andrew.cmu.edu/~ryanod/ - Page 114-116 - Chapter 5 - Chow's Theorem 5.1 and Theorem 5.8, Threshold of Parities Circuit (depth 2 comprising parity gates in layer 1 and threshold gate at layer 2) computing a Polynomial Threshold Function
682.7 Perceptrons - [Minsky-Papert] - (1972:232): "... a universal computer could be built entirely out of linear threshold modules. This does not in any sense reduce the theory of computation and programming to the theory of perceptrons."
682.8 Circuit Complexity Lecture Notes - http://www.cs.umd.edu/~jkatz/complexity/f05/lecture4.pdf - Section 1.2 - "Theorem 1 If t(n) ≥ n, then time(t) ⊆ size(tlog t)...."
682.9 Satisfiability and Derandomization for Small Polynomial Threshold Circuits - CircuitSAT for circuits of PTF gates -

https://eccc.weizmann.ac.il/report/2018/115/download/ - s-sparse PTFs having s monomials can encode MAXSAT of s clauses - Previous Boolean Composition of Majority and VoterSAT can alternatively be stated as a Circuit Composition of Majority and Voter PTF circuits - section 1.4 - "...for fixed sparsity s, much more expressive. For example, the polynomial p(x1, . . . , xn) = Qn i=1(xi ⊕1) = Qn i=1(1−xi) has sparsity 1 by our definition, but sparsity 2^n by the usual definition. Sakai, Seto, Tamaki and Teruyama [SSTT16] recently reported a faster-than-brute-force algorithm for MAX-kSAT for any constant k with arbitrary weights (which implies a satisfiability algorithm for degree-k PTFs)..."

682.10 Bounded Depth Circuits with Weighted Symmetric Gates: Satisfiability, Lower Bounds and Compression - [Sakai-Seto-Tamaki-Teruyama] - https://pdfs.semanticscholar.org/084f/27ca498158992b5e5a5a61dbaf4161503b5c.pdf

682.11 Learning DNF - [Klivans-Servedio] - https://pdf.sciencedirectassets.com/272574/1-s2.0-S0022000000X01572/1-s2.0-S0022000003001363/main.pdf?X-Amz-Security-Token=AgoJb3JpZ2luX2VjEJn%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLWVhc3QtMSJIMEYCIQDpY%2BJ8ysvIVFtFSRPRVv6S2w42%2FtCoesbH0lj0MJMoawIhAKgdNj9DXXjJN1myHJPiU14ZKBUvHmphnWHLaGd%2F0LCFKtoDCBEQAhoMMDU5MDAzNTQ2ODY1Igwd1ipPK634MOjEQmcqtwPLCTvVsLs90BxtDUtf9IbmHo6ay73ITn2Thu7F%2F%2BdbUWumBXt4nTo%2FyRZle3Bz4V2GoWLCRjc6AEDJ1GaC4Enc3HMXOtC3PPiKo5DaQ9EH%2BaGSUqb1Nm1fu4OiohaH%2F7%2FqLSau0TqH5QbnoCzqb9F8cZwI4%2FNhIFdm6IbL3YhOiizEDKzd5tylsXH46lNIZNOcPouFRCZC%2B6korSB0yEhTzp%2BxlOBosz5%2BknhQOdJGsgog7aHP5O7iB2Aml3V%2BIwQAg8Pl%2BFZ9Dc8WyAtxiQU%2Bh%2BcdJh9faDjqDbjG7VkzSuhKfVH%2B6tQiwbSPyLSLvosFne65KYI%2FdPytCJ5C7sHMuRot3V4JGJHhmcmyz5h8YeVK3eTwigD4j1cPzZQpifS33XIaca1do32RJNs9%2BaWCC6dKWlOl6s4wHxVQE%2BcJ6ixOSsbjKuRo%2FRS9dFJ5%2F%2FCfVBoHG2F%2B9rDDktiSZiegDoK7ndITZvPeymzrTbAVZqIHHx83sL3TfJiFLPPwrScwhLkfoF3I32griAkFIURzNWYF%2B6bzxsh1QxQUgNIdshGhmyRRca9ktj33sPdyvFDBEHLYHdAhMPHM5ekFOrMBR1vs5b47SxPOuk17oLpowsH6xCobvsX76f2ShEhvoQNLqNQis%2F6LYtV6QNLgUQj%2BjUobapU2SaT2pMqPrwMvshW06K5Kt1GQMWnF84OiPAxss85P45Za7aCwifUX1GCflCxtm%2FWFqmXuHDrq2uxze6LlzExKN0mglTcusR7rZXTolUTqXT2zZ1mjtRXn3mWYIE5uoAS320ECsACejGBOtTCISfjA2GYNQl1Je%2FqASIPd2aM%3D&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20190725T091916Z&X-Amz-SignedHeaders=host&X-Amz-Expires=300&X-Amz-Credential=ASIAQ3PHCVTYQWHGSGT4%2F20190725%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=b04a101db0c513928d40565cf895dfc601324123b95c08e2938f461bd76c3490&hash=eb5b8aed06292daa7502dca41a62db9110c81a7c336d1543c86683f1bb58faa3&host=68042c943591013ac2b2430a89b270f6af2c76d8dfd086a07176afe7c76c2c61&pii=S0022000003001363&tid=spdf-f4e32f52-621c-4768-8b49-b835a3261877&sid=31838a2e23fe724bb06b9d4-5a4c8715f5a8gxrqa&type=client - "...Using techniques from learning theory, we show that any s-term DNF over n variables can be computed by a polynomial threshold function of degree O(n^(0.33)log s): This upper bound matches, up to a logarithmic factor, the longstanding lower bound given by Minsky and Papert in their 1968 book Perceptrons...."

682.12 New Degree bounds for Polynomial Threshold Functions - [O'Donnell-Servedio] - https://kilthub.cmu.edu/articles/New_degree_bounds_for_polynomial_threshold_functions/6607712/files/12098267.pdf - "...We prove that any Boolean formula of depth d and size s is computed by a polynomial threshold function of degree √s(log s)^O(d)...."

682.13 Block Composed Functions and Hardness Amplification - https://arxiv.org/pdf/1710.09079.pdf - "...The approximate degree of a Boolean function f : {−1, 1}^n → {−1, 1}, denoted deg g(f), is the least degree of a real polynomial p such that |p(x) − f(x)| ≤ 1/3 for all x ∈ {−1, 1}^n ... Until very recently, the method of dual polynomials had been used exclusively to prove hardness amplification results for block-composed functions. That is, the harder function g would be obtained by block-composing f with another function h, i.e., g = h ◦ f. Here, a function g : {−1, 1}^n·m → {−1, 1} is the block-composition of h: {−1, 1}^n → {−1, 1} and f : {−1, 1}^m → {−1, 1} if g interprets its input as a sequence of n blocks, applies f to each block, and then feeds the n outputs into h...." - Majority+VoterSAT boolean composition can be viewed as block composition by approximating Majority and VoterSAT by low-degree polynomials h and f

---
683.(THEORY) Proof of KRW Conjecture for Majority+VoterSAT/VoterPTF composition - related to 682 - 23 July 2019

Caution: This section is experimental and subject to errors

Previously a reduction from CNF MAX-k-SAT to PTFSAT was described with an example. Any Voter SAT can be encoded as a Polynomial Threshold Function g and any Polynomial Threshold Function can be computed by a Threshold-of-Parities circuit of depth 2 described in previous section:

g = threshold + parity

If Majority function is denoted by f, Majority + VoterSAT/VoterPTF composition can be written in terms of parity as:

f + g = f + threshold + parity

But composition is associative and therefore:

(f + threshold) + parity = f + (threshold + parity)

But composition (f + threshold) previously can be any function h:

(f + threshold) + parity = h + parity

which is composition of some function and parity and KRW conjecture for composing any function and parity has been already proved ([Hastad],[Andreev], [Dinur]) and thus KRW Conjecture for Majority + VoterSAT/VoterPTF composition is proved immediately because previous breakdown of any function as threshold of parities reduces Majority + VoterSAT/VoterPTF composition to composition of any function to parity.

For a general case of composing any function f to another function g, applying previous threshold of parities representation to both f and g:

f + g = (threshold + parity) + (threshold + parity)

and applying associativity of composition:

f + g = (threshold + parity + threshold) + parity

---
684. (FEATURE) Urban Planning/GIS Image Analytics - scikit learn extract_patches_2d() integration - 25 July 2019
---
1. Scikit Learn extract_patches_2d() has been invoked in a wrapper function analyze_remotesensing_2d_patches() for patches analysis of Remote Sensing Imagery
2. Patch ndarrays are flattened and pixel histogram is computed for each patch of an image
3. Maximum pixel bucket in the patch histogram is an indicator of class of the patch.
4. This is in addition to RGB patch analysis function implemented earlier for Red-Green-Blue channel classification of an image
5. Logs of extract_patches_2d() and the patch histograms for 2 example remote sensing images are at ImageNet/testlogs/ImageGraph_Keras_Theano.log.RemoteSensingGIS.25July2019

---
685. (FEATURE) People Analytics - Kaggle LinkedIn Dataset Analysis - 26 July 2019
---
1. This commit introduces a new function in People Analytics implementation - linkedin_dataset_tenure_analytics() - which parses Kaggle LinkedIn dataset from https://www.kaggle.com/killbot/linkedin and computes various correlation coefficients between column attributes.
2. Following are the People Analytics field attributes in LinkedIn obfuscated

dataset which has almost 15000 rows:
['avg_n_pos_per_prev_tenure', 'avg_pos_len', 'avg_prev_tenure_len', 'c_name', 'm_urn', 'n_pos', 'n_prev_tenures', 'tenure_len', 'age', 'beauty', 'beauty_female', 'beauty_male', 'blur', 'blur_gaussian', 'blur_motion', 'emo_anger', 'emo_disgust', 'emo_fear', 'emo_happiness', 'emo_neutral', 'emo_sadness', 'emo_surprise', 'ethnicity', 'face_quality', 'gender', 'glass', 'head_pitch', 'head_roll', 'head_yaw', 'img', 'mouth_close', 'mouth_mask', 'mouth_open', 'mouth_other', 'skin_acne', 'skin_dark_circle', 'skin_health', 'skin_stain', 'smile', 'african', 'celtic_english', 'east_asian', 'european', 'greek', 'hispanic', 'jewish', 'muslim', 'nationality', 'nordic', 'south_asian', 'n_followers'] which include average variables for tenure, positions or designations per tenure, number of previous tenures, tenure lengths etc.,
3.Pairwise kendall-tau Correlation coefficients of 3 important variables are computed:
- average number of positions per previous tenure
- average length of a position
- average length of previous tenures
4.Some preprocessing of the linkedin CSV dataset was required to cast the type to integer
5.VectorAssembler is instantiated to create necessary subset projections of the linkedin dataset columns
6.5 important columns of the Spark CSV dataframe are sliced by list comprehension to 5 arrays for (ethnic, emotional, physical and racial fields are ignored):
- average number of positions per previous tenure
- average length of a position
- average length of previous tenures
- number of previous tenures
- tenure length
and mapped to a Pandas DataFrame which supports correlation matrix for all datatypes
7.Pandas correlation matrix is printed for these 5 fields:

	avg_n_pos_per_prev_tenure	avg_pos_len	avg_prev_tenure_len	n_prev_tenures	tenure_len
avg_n_pos_per_prev_tenure	1.000000	0.804373	0.836112	0.836637	0.832915
avg_pos_len	0.804373	1.000000	0.983461	0.978873	0.993264
avg_prev_tenure_len	0.836112	0.983461	1.000000	0.968333	0.975659
n_prev_tenures	0.836637	0.978873	0.968333	1.000000	0.981673
tenure_len	0.832915	0.993264	0.975659	0.981673	1.000000

8.In addition to basic designation(position),remuneration, tenure duration correlations implemented earlier previous variables provide fine grained pattern analysis.
9,From previous correlation matrix, an obvious negative or low correlation is:
avg_n_pos_per_prev_tenure - versus - (avg_prev_tenure_len, n_prev_tenures, tenure_len)
implying increased average number of designations(positions) in previous tenures implies low average previous tenure lengths, low number of previous tenures and low tenure length which (exluding low number of previous tenures) is against what intuition suggests.

---

686. (FEATURE) People Analytics - Kaggle LinkedIn DataSet - Experiential and Degree Intrinsic Merit - 27 July 2019

1.In continuation to previous commit for linkedin dataset analytics, this commit implements computation of lognormal experiential intrinsic merit and degree

based experiential intrinsic merit for all profiles in linkedin dataset.
2.Pandas DataFrame has been expanded to include two new columns - lognormal_experiential_merits and degree_experiential_merits - and correlation matrix is computed for this 7 rows * 7 columns.
3.Experience of the n-th profile is derived from dataset columns as:
        experience = avg_n_pos_per_prev_tenure[n] * avg_pos_len[n] * n_prev_tenures[n] + tenure_len[n]
4.Correlation coefficients for all pairs of 7 variables from logs:

```
linkedin_dataset_tenure_analytics(): pandas correlation coefficient =
avg_n_pos_per_prev_tenure  avg_pos_len  ...  lognormal_experiential_merits
degree_experiential_merits
avg_n_pos_per_prev_tenure                       1.000000     0.804373  ...
0.687246                   0.116707
avg_pos_len                                     0.804373     1.000000  ...
0.566021                   0.112341
avg_prev_tenure_len                             0.836112     0.983461  ...
0.673678                   0.156701
n_prev_tenures                                  0.836637     0.978873  ...
0.553288                   0.058373
tenure_len                                      0.832915     0.993264  ...
0.532364                   0.076770
lognormal_experiential_merits                   0.687246     0.566021  ...
1.000000                   0.426055
degree_experiential_merits                      0.116707     0.112341  ...
0.426055                   1.000000

[7 rows x 7 columns]
```

5.Previous matrix shows highly negative or low correlation between first 3 tenure variables and degree experiential merits and a medium correlation between first 3 variables and lognormal_experiential_merits.

---

687. (THEORY) Approximate Polynomials and Approximate Degree of Boolean Functions, Dual Polynomials, Hardness Amplification, KRW Conjecture, Boolean Composition of Majority and VoterSAT/VoterPTF, Communication Complexity, Linear Program Primal/Dual - related to 365, 368, 682 -
27 July 2019

From 682.13, the approximate degree of a Boolean function f : {−1, 1}^n → {−1, 1}, denoted deg g(f), is the least degree of a real polynomial p such that |p(x) − f(x)| ≤ 1/3 for all x ∈ {−1, 1}^n. Existence of approximate polynomials implies that the function is easy. Therefore non-existence of approximate polynomials implies the function is hard. Fact that f can be approximated by a real polynomial is equivalent to feasibility of a Linear Program on Fourier coefficients. Feasibility of dual of this linear program on Fourier Coefficients implies the primal boolean function is hard to approximate by a real polynomial and thus is an alternative way to prove hardness of boolean functions and their compositions especially block-compositions and is quite relevant to Majority+VoterSAT composition and KRW-Communication Complexity based size and depth lowerbounds of it previously derived based on Sensitivity measures. Approximate degree of t-threshold functions is sqrt(t(n-t)) and approximate degree of Majority or Parity is Theta(n). From [Nisan-Szegedy], approxdegree(f) <= degree(f) <= DecisionTree(f) < 1296*degree(f)^8. From [Sherstov] theorem, approxdegree(Majority + VoterSAT) = approxdegree(Majority) * approxdegree(VoterSAT) <= DecisionTree(Majority)*DecisionTree(VoterSAT).

References:

687.1 Dual Polynomial for OR function - [Robert Spalek, Google, Inc.] - http://www.ucw.cz/~robert/papers/dualor.pdf

687.2 Degree of Polynomials Approximating Symmetric Boolean Functions - [Ramamohan Paturi] - https://cseweb.ucsd.edu/~paturi/myPapers/pubs/Paturi_1992_stoc.pdf
687.3 Approximate degree of composition of boolean functions - Theorem 8 - [Sherstov] - The Polynomial Method Strikes Back:
Tight Quantum Query Bounds via Dual Polynomials - [Mark Bun Princeton University mbun@cs.princeton.edu, Robin Kothari Microsoft Research robin.kothari@microsoft.com,Justin Thaler Georgetown University justin.thaler@georgetown.edu] - https://arxiv.org/pdf/1710.09079.pdf

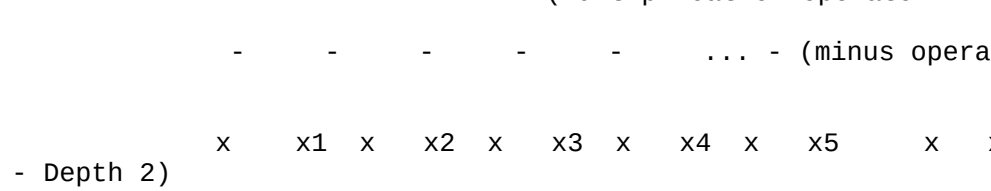
---

688. (THEORY) Approximate Polynomials and Approximate Degree of Boolean Functions, Algebraic Circuits and KRW Conjecture for Boolean composition of Majority and VoterSAT/VoterPTF - related to 365, 368, 682, 687 - 28 July 2019, 1 August 2019

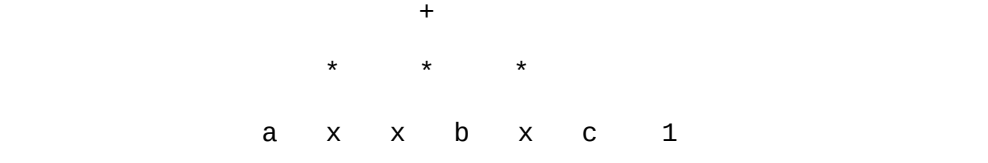
Approximate Polynomial for Majority exists from 687.2 and has approximate degree theta(n) and therefore has n roots from Fundamental Theorem of Algebra. This approximate polynomial for Majority can be written as factorized product (assuming reducibile polynomials over field of reals):

Product(x-xi) i=1,2,3,...,n

Previous product for approximate polynomial for majority can be expressed by an Algebraic/Arithmetic Circuit of depth-2 having 2n leaves (2 leaves per factor) and n intermediate minus gates in depth-1. Root is a multiplication gate of fanin n:



Similarly if there exists an approximate polynomial (assuming reducible over field of reals) for an arbitrary VoterSAT/VoterPTF arithmetic circuit of approximate degree d it can be factorized and represented by an Algebraic/Arithmetic circuit of depth-2 having 2d leaves, d intermediate minus gates at depth-1 and multiplication gate at root. Since both arithmetic circuits for Majority and VoterSAT/VoterPTF have fanout 1, both can be termed arithmetic formulas. Thus approximate polynomials for Majority and VoterSAT/VoterPTF have arithmetic circuits of size O(approxdegree). Block Composing Majority arithmetic formula and VoterSAT/VoterPTF arithmetic formula is fairly straightforward in this special setting and ArithmeticFormulaSize(approxpoly(Majority) +approxpoly(VoterSAT)) = ArithmeticFormulaSize(approxpoly(Majority)) * ArithmeticFormulaSize(approxpoly(VoterSAT)) = O(n*d). This is an approximate version of KRW conjecture and O(n*d) could be Theta(n*d) because factorization lowerbounds the number of arithmetic gates. Over field of reals, every irreducible polynomial has degree 1 or 2 at most. Previous algebraic circuit assumes degree 1 irreducibility. For degree 2, instead of factors of the form (x-xi), factors are ax^2 + bx + c which require arithmetic subcircuits as below increasing depth by 1 and size multiplies by 4:



An already known result in reference 688.4 implies intersection of set of polynomials computed by non-uniform arithmetic circuits of polynomial size and set of polynomials of constant degree is equal to intersection of set of polynomials computed by nonuniform arithmetic circuits of polynomial size and

logN*logN depth and set of polynomials of constant degree for all fields F. This implies each of the arithmetic circuit for approximate univariate polynomial over field of reals of a boolean function is a logN * logN depth circuit. Assuming degree 1 irreducible factors throughout previously for approximate real polynomials of both functions f and g, an estimate of depth lowerbound of the block composition of f and g can be arrived at:
      Depth(f + g) ~= Depth(f) + Depth(g) ~ 2*logN*logN

References:

688.1 Algebraic Circuits - [Sanjeev Arora-Boaz Barak] - https://theory.cs.princeton.edu/complexity/book.pdf - Section 14.1 and Example 14.3 (Arithmetic circuits for determinants)
688.2 Fundamental Theorem of Algebra - https://en.wikipedia.org/wiki/Fundamental_theorem_of_algebra - "... every non-zero, single-variable, degree n polynomial with complex coefficients has, counted with multiplicity, exactly n complex roots..."
688.3 Irreducible polynomials over reals have degree 1 or 2 - https://en.wikipedia.org/wiki/Irreducible_polynomial
688.4 Arithmetic Circuits versus Boolean Circuits - [Joachim von zur Gathen† - Gadiel Seroussi‡] - Section 2 - https://www.sciencedirect.com/science/article/pii/089054019190078G

---
689. (FEATURE) Fraud Analytics - Credit Card Dataset - Pandas DataFrame Correlation Coefficient Matrix - 28 July 2019

1.Fraud Analytics implementation has been updated to instantiate Pandas DataFrame from credit card dataset and print Pandas
Correlation matrix for all the columns.
2.Columns V1-V28 are Obfuscated Names of the cardholders which are numeric unique id(s)
3.Correlation between unique identities/names and amount drawn is a behavioural pattern and any deviation is an alarm (most swipes are periodic and personality oriented).
4.Correlation matrix logs are committed to testlogs/FraudAnalytics.log.28July2019
5.Correlation between Timestamp and Amount drawn is also measure of fraud.

---
690. (FEATURE) Time Series Analysis - ARIMA implementation - 1 August 2019

1.Time Series implementation has been made to a new python source file containing time series algorithms implementations for ARMA and ARIMA - Streaming_TimeSeriesData.py
2.Auto Regressive Integrated Moving Averages (ARIMA) has been implemented as a function ARIMA in Streaming_TimeSeriesData.py and factoring has been implemented for a parameter lag in autoregression_factored()
3.ARIMA differs from ARMA in factoring the autoregression polynomial into two factor polynomials one which is dependent on a lag operator.
4.Because Yahoo finance and Google Finance stock data are no longer available, ystockquote does not work and throws urllib exception and hence a hardcoded array of timeseries stock close data has been used.
5.ARMA and ARIMA functions from Streaming_TimeSeriesData.py are invoked in Streaming_StockMarketData.py.
6.Logs for ARIMA and R+rpy2 graphics plot have been committed to DJIA_ARIMA_Time_Series.pdf and testlogs/Streaming_StockMarketData.ARIMATimeSeries.log.1August2019

---
691. (FEATURE) Time Series Analysis - ARMA and ARIMA implementations - Rewrite - 2 August 2019
---
1. ARMA and ARIMA related functions in Streaming_TimeSeriesData.py have been rewritten based on ARMA and ARIMA polynomials in https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average
2. Projection iterations have been removed and instead are done in the invoking source file e.g Streaming_StockMarketData.py for randomly chosen regression weights
3. ARMA polynomial is computed as:
    arma=timeseries[len(timeseries)-1] - autoregression(time_series_data[pprime:]) - moving_averages(time_series_data[q:], 5)
which subtracts the autoregression and moving averages from the latest timeseries point. ARMA parameters pprime and q are sliced to be as distant from the latest timeseries point
4. ARIMA polynomial is computed as:
    arima=autoregression_factored(time_series_data[len(time_series_data)-d:],p) - moving_averages(time_series_data[q:], 5)
which computes autoregression_factored differently as:
5.Lag operator for lag d is applied to slice the timeseries data from latest point t to d as t-d and autoregression_factored() is invoked to compute regressions from slice t-d-p to t-d
6.Finally the autoregression is subtracted from latest timeseries point Xt and moving averages is added to it
7.Logs for rewrite are available for few iterations at testlogs/Streaming_StockMarketData.ARIMATimeSeries.log.2August2019
8.R+rpy2 function plots have been commented

---
692. (FEATURE) Time Series Analysis - ARMA and ARIMA implementations - Rewrite 2 - 3 August 2019
---
1. ARMA and ARIMA functions in Streaming_TimeSeriesData.py have been again refactored to implement lag operator and binomial expansion of lag factor in ARIMA polynomial
2. autoregression_factored() has been rewritten to invoke a new function lag_factor_binomial_expansion() which in turn expands the factor power term in ARIMA polynomial by binomial series expansion of (1-L)^d.
3. lag_factor_binomial_expansion() loop invokes functions binomial_term() which computes the numerator and denominator factorial of each binomial term and lag() function for each binomial term.
4. Example logs for Stock quotes timeseries data is at Streaming_StockMarketData.ARIMATimeSeries.log.3August2019 which has comparative projections of rewritten ARMA and ARIMA functions.
5. ARIMA has been invoked by parameters (2,2,2)

---
693. (FEATURE) Time Series Analysis and Fraud Analytics - Credit Card Transactions Dataset - 3 August 2019

1.FraudAnalytics.py has been updated to import TimeSeries ARMA and ARIMA functions from Streaming_TimeSeriesData.py and compute ARMA and ARIMA polynomials on amounts withdrawn timeseries in credit card transactions dataset.
2.Logs for these projections are at testlogs/FraudAnalytics.log.3August2019
3.Though timeseries is a perfect fit for analyzing credit card amounts withdrawn as timeseries, projection curves for ARMA and ARIMA in logs are skewed widely for multiple iterations of random weights in autoregression and moving averages

but values of ARMA and ARIMA are uniformly same almost.

---
694. (FEATURE) Linear and Logistic Regressions - Rewrite - 4 August 2019, 6 August 2019

1.LinearAndLogisticRegression.py has been rewritten to accept arbitrary number of weights and variables for computing Linear and Logistic Regressions.
2.Linear And Logistic Regressions are estimators of Intrinsic Economic Merit of Nations in Economic Networks. Thus far following classes of intrinsic merit have been researched and implemented in NeuronRain:
1. Text
2. Audio - Speech
3. Audio - Music
4. Visuals - Images
5. Visuals - Videos
6. People - Professional Networks - Experiential Intrinsic Merit
7. People - Social Networks - Log Normal Least Energy Merit - Education/Wealth/Valour e.g Rankings of Universities based on various factors including research output, Rankings of Students based on examinations/contests, Sports IPR (Elo ratings in Chess, Player rankings in sports)
8. People - Professional Networks - HR Analytics - Attrition model and Economic merit of nations is the 9th class of intrinsic merit applicable to Economic Networks. Regression is one amongst the set of econometric techniques for ascertaining growth of economies:
9. Nations - GDP, HDI etc.,
3. Logs for the previous are at testlogs/LinearAndLogisticRegression.log.4August2019

References:

694.1 India’s GDP Mis-estimation: Likelihood, Magnitudes, Mechanisms, and Implications - [Arvind Subramanian] - Working Paper - https://growthlab.cid.harvard.edu/files/growthlab/files/2019-06-cid-wp-354.pdf - "...We divide the sample into two periods, pre-and post-2011. For each period we estimate the following cross-country regression: $GDP\ Growth_i = \beta 0 + \beta 1 Credit\ Growth_i + \beta 2 Electricity\ Growth_i + \beta 3 Export\ Growth_i + \beta 4 Import\ Growth_i + \beta 5 India + \varepsilon_i$ ..."

---
695. (FEATURE) Gradient Ascent and Descent - Rewrite - 4 August 2019

1.PerceptronAndGradient.py has been rewritten for computing Gradient Ascent and Descent for arbitrary number of variables.
2.Earlier loop for partial derivative delta computation for just 2 variables has been generalized for any number of variables
3.Gradient Ascent has also been added based on computed value of weight updated perceptron (+ instead of -)
4.Logs for Ascent and Descent are at testlogs/PerceptronAndGradient.log.4August2019
5.Hardcoded variable,weights,bias,rho,output values have been removed and made as arguments.

---
696. (FEATURE) Named Entity Recognition (NER)-HMM-Viterbi-CRF - Rewrite - 12 August 2019

1.Named Entity Recognition by Conditional Random Fields Hidden Markov Model has

been updated significantly for resolving some errors in Viterbi computation
2.Prior Emission Probabilities computation by skew normal distribution has been updated to renormalize by dividing the skewnorm probability by sum of probabilities
3.New part of speech "adverb" has been included in transition probabilities,emission probabilities and states
4.weighted sknorm has been done away with and replaced by previous renormalized skewnorm distribution
5.Error in argmax() computation has been corrected - label and maximum are reset in the loop
6.logs for this rewrite have been committed to testlogs/NamedEntityRecognition_HMMViterbi_CRF.log.12August2019
7.Transition probabilities are updated as below:

```
transition_probabilities={ 'noun':{'noun':0.0, 'verb':0.3, 'object':0.2, 'adjective':0.1, 'adverb':0.1, 'conjunction':0.3},
                           'verb':{'noun':0.1, 'verb':0.0, 'object':0.4, 'adjective':0.2, 'adverb':0.1,'conjunction':0.3},
                           'object':{'noun':0.0, 'verb':0.1, 'object':0.4, 'adjective':0.1, 'adverb':0.1,'conjunction':0.3},
                           'adjective':{'noun':0.4, 'verb':0.2, 'object':0.2, 'adjective':0.0, 'adverb':0.1, 'conjunction':0.1},
                           'adverb':{'noun':0.1, 'verb':0.4, 'object':0.1, 'adjective':0.0, 'adverb':0.1, 'conjunction':0.3},
                           'conjunction':{'noun':0.2, 'verb':0.4, 'object':0.1, 'adjective':0.1, 'adverb':0.1, 'conjunction':0.1}
                          }
```

7.It has to be mentioned here that previous state transition probabilities priors are hardcoded heuristics and based on the following rationale:
      7.1 verb follows nouns more often
      7.2 object follows verb more often
      7.3 object follows object more often
      7.4 noun follows adjective more often
      7.5 verb follows adverb more often
      7.6 verb follows conjunction more often (e.g infinitives)
8.Previous heuristics are minimal and could be replaced by more sophisticated probability distribution learnt from a training dataset once and loaded everytime.

---
697. (THEORY-FEATURE) Named Entity Recognition and Text Graph Algorithms in NeuronRain AstroInfer - a non-statistical alternative - 12 August 2019

TextGraph algorithms in NeuronRain are two-fold:
      697.1 Recursive Gloss Overlap - Project WordNet to a Subgraph by traversing the Synsets recursively
      697.2 Recursive Lambda Function Growth - Project WordNet to a Subgraph by traversing the Synsets recursively, learn AVL tree based Graph Tensor Neural Network and Lambda expressions from cycles and random walks of the WordNet subgraph - this is the maximum attainable theoretical limit of learning a formal language (Turing Machine or a lambda function) from natural language

Usual Named Entity Recognition algorithms are prior probabilities and Conditional Random Fields-HMM based which relies heavily on statistics ignoring linguistic deep structure. Instead if missing Part-of-Speech tags could be inferred from a textgraph representation of a partially PoS annotated text, no language PoS priors are required. Following algorithm explores this alternative:
      697.3 Partial PoS annoted text is mapped to a text graph by previous algorithms and each word vertex in the partial annotated text is labelled by either a part of speech or ? for missing PoS tag.
      697.4 Resultant text graph thus has mixture of word vertices which have either PoS known (labelled by a PoS name) or PoS unknown (labelled by ?)
      697.5 WordNet has support for list of all possible PoS tags for a word

vertex
697.6 It is then a problem of inferring most probable disambiguating PoS tag for a word vertex.
697.7 Set of all wordnet paths between a PoS annotated word vertex and an adjacent PoS unannotated word vertex provide set of sequences of PoS tags for the paths.'
697.8 For example phrase "can be called as a function" annotated as "conjunction conjunction verb conjunction conjunction ?" and unannotated "function" has three possible wordnet PoS tags which have to be disambiguated - Noun, Verb and Object and set of wordnet paths between "as-a" and "function" could be - {"connective","entity","work"},{"article","entity","event"}, {"one","entity","happening"} having PoS sequences as {noun,noun,verb}, {noun,noun,noun} and {noun,noun,noun} if function has lemma synsets "work","event" or "happening"
697.9.Of the three possible paths PoS "noun" for "function" wins by majority voting of 2 to 1 because 2 paths have a penultimate vertices which imply "function" is a noun.

---
698. (FEATURE) GIS Remote Sensing Image Analytics - Image Segmentation - 13 August 2019

1.New function image_segmentation() has been defined in ImageGraph_Keras_Theano.py which segments an image into similar regions or patches
2.As opposed to RGB patches and extract_patches_2d() of scikit learn, segmentation of an image is generic and segments/patches of arbitrary irregular shapes and containing similar pixels can be extracted
3.Example code in OpenCV2 documentation - https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_watershed/py_watershed.html - has been used as reference
4.This example filters the image into foreground and background by:
4.1 applying OTSU threshold filter
4.2 morphological opening for removal of noise
4.3 Background - Dilation to increase object boundary to background and demarcate background accurately
4.4 Foreground - Distance transform and then thresholding is applied to demarcate the foreground
4.5 Markers - Connected components are found from foreground of 4.4
4.6 Watershed - watershed segmentation algorithm is applied on the marked image to find segments.
5.Watershed algorithm works by considering an image as topographic surface of valleys and dams between valleys. Valleys in image are filled with water till no two adjacent valleys merge creating "dams" or segments bounding the valleys.
6.Watershed segmentation is graphically illustrated in http://www.cmm.mines-paristech.fr/~beucher/wtshed.html
7.EventNet Tensor Products Merit algorithm for visuals described earlier depends on bounding box segmentation which are squares. Instead segmentation could be the way to go if ImageNet and Keras support it.
8.Two segmented satellite images of Chennai Metropolitan Area Urban Sprawl are committed at:
8.1 testlogs/RemoteSensingGIS/ChennaiUrbanSprawl_Page-10-Image-15_segmented.jpg
8.2 testlogs/RemoteSensingGIS/ChennaiUrbanSprawl_Page-9-Image-13_segmented.jpg

---

699.(THEORY-FEATURE) Computational Geometric Factorization - Tile Search Optimization - Spark 2.4.3 and QuadCore benchmarks - 13 August 2019

1.This is the first genuine benchmark of Computational Geometric Factorization on Spark 2.4.3 and QuadCore. Earlier execution on local[4] was a simulation of 4 spark executor threads on single core
2.SparkContext has been replaced by Spark context of SparkSession
3.logs for factorization are at testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.log.13August2019 which have some notable estimates of approximate factors by Hardy-Ramanujan (1 exact factor ray query), Baker-Harman-Pintz and Cramer ray shooting queries

---

700.(THEORY) Computational Geometric Factorization, Complement Diophantines, Covering a set by Arithmetic Progressions, Ramsey Coloring - related to 24,477,491,636 - 14 August 2019

Computational geometric factorization works by parallel planar point location - rectification of Hyperbolic arc bow into straightline segments which are then made to segment tree, wavelet tree, interval tree etc., by parallel RAM construction or segments are parallel RAM mergesorted and binary searched to find factors and thus factorization (finding all factors of an integer) is in Nick's class. It has been mentioned earlier that each segment of rectified hyperbolic arc bow is an arithmetic progression which facilitates local binary search without necessity for parallel mergesort. Thus set of all segments of rectified hyperbolic arc create an arithmetic progression cover of set of points on rectified hyperbolic curve. Finding a minimum size set of arithmetic progressions covering a set of integers is a hard problem. Previous rectification process for factorization solves a special case of arithmetic progression cover of a set. It has to be noted that finding an exact arithmetic progression cover of a set is the complement diophantine problem because the set of arithmetic progressions represents each of the diophantine subset in the exact cover partition. Known result in 700.1 implies covering a set by arithmetic progressions is NP-complete while the previous rectification for factorization solves this special case of arithmetic progression cover of set of rectified hyperbolic points in NC.This also implies complement diophantine problem is NP-complete for the special case of arithmetic progression diophantines for some partition of a set.

References:

700.1 Covering a set by arithmetic progressions is NP-complete - [Lenwood Heath - Virginia Tech] - https://vtechworks.lib.vt.edu/bitstream/handle/10919/19566/TR-89-25.pdf?sequence=3&isAllowed=y

---

701.(THEORY) Category Theory formulation of EventNet - Objects, Morphisms, Functors - related to 582, 588, 654 - 21 August 2019, 22 September 2019, 29 October 2019,6 December 2019

Category Theory of [Eilenberg-MacLane] abstracts whole of mathematics into following:

701.1 Category C is a Set of Objects

701.2 Every Category has set of morphisms f:a-b defined on objects a,b in Category C

701.3 Functors F are maps between two Categories C1 and C2 - for each object x in C1, F(x) is in C2 and for each morphism f:a-b in C1 morphism, F(f):F(a)-F(b) is in C2

EventNet Tensor Products defined and implemented in NeuronRain and applied as intrinsic merit measure for large scale visuals can be formalised in terms of

Category theory as follows:

701.4 Every Event vertex in EventNet is a category

701.5 Set of Actors in an Event Category are Objects of the Category

701.6 Every intra-event interaction between any two actor objects a,b of an event category E is a Morphism - f:a-b

701.7 Every inter-event causation between two Event Categories E1 and E2 (E1 causes E2) is a Functor - for each actor object x in E1, F(x) is in E2 and for each morphism f:a-b in E1, morphism F(f):F(a)-F(b) is in E2

As mentioned in previous sections on EventNet Tensor Product merit of Large Scale Visuals, EventNet Tensor formalizes logical time and causation. Rank of a Tensor T is the minimum number of Simple Tensors that sum to T where Simple Tensors are factorizable products of tensors. This implies Logical Time Tensor can be written as sum of products of Tensors implying seemingly linear 1-dimensional time is decomposable to tensor components basis. This has implications for physical sciences and experiments involving spacetime - Time could be multidimensional and any spatial object can be projected to any of the time tensor component axes similar to space. From previous Category Theory definition, EventNet Causality Functor Graph can also be written as sum of products of Tensors. Tensor Decomposition of Video EventNet Tensor yields component videos i.e video can be split into linearly independent component videos.

Another theoretical implication of EventNet is the possibility of retrocausality - effect causing the cause and arrow of time can be bidirectional. EventNet DAG can be partitioned into past, present and future disjoint subgraph components by 2 cuts/vertex separators. To preclude the possibility of retrocausality/time reversal, EventNet must be a directed acyclic graph devoid of loops and backedges (edges of right to left if Events flow left to right). If logical time is unidirectional (retrocausality is false) following "future" functions f1 and f2 defined on EventNet graph partition components (past,present,future) must be hard to invert implying one-way functions, Pseudorandom Generators and P != NP (left inverse does not exist for EventNet category):

f1(past) = present

f2(present) = future

Ramsey Theorem for Graphs implies any large graph G has a Ramsey number R(s,t) such that a graph G of R(s,t) vertices has an independent set of size s or a clique of size t. In the context of EventNet which is a huge causality graph, Ramsey Theorem implies there always exists a clique of mutually dependent events or mutually independent set of events in EventNet of R(s,t) vertices howsoever random the causality is.

References:

701.8 Category Theory - https://en.wikipedia.org/wiki/Category_theory

701.9 Tensor Products as Functors - https://en.wikipedia.org/wiki/Functor - "... Tensor products: If C denotes the category of vector spaces over a fixed field, with linear maps as morphisms, then the tensor product {\displaystyle V\otimes W} V\otimes W defines a functor C × C → C which is covariant in both arguments. [7] ..."

701.10 Tensor Decomposition in Python - Tensorly and Tensorlab - https://medium.com/yadb/tensor-factorization-for-graph-analysis-in-python-590df44c9f6c - Graphic illustrations of Non-Negative Canonical Polyadic Decomposition (NCPD) of a 3-way tensor to sum of products of multiple rank-1 tensors - Figure 1 (3-way tensor) decomposed as Figure 2 (sum of products of rank-1 tensors) - "...The rank of a tensor T, denoted rank(T), is defined as the smallest number of rank-one tensors that generate T as their sum..."

701.11 Tensor Decomposition and its applications - [Tamara G. Kolda and Brett W. Bader] - https://epubs.siam.org/doi/abs/10.1137/07070111X?journalCode=siread - "...A tensor is a multidimensional or N-way array. Decompositions of higher-order tensors (i.e., N-way arrays with N >= 3) have applications in psycho-metrics, chemometrics, signal processing, numerical linear algebra, computer vision, numerical analysis, data mining, neuroscience, graph analysis, and

elsewhere ... "
701.12 Ramsey Theorem for Graphs - http://math.mit.edu/~fox/MAT307-lecture05.pdf - Complete disorder is impossible - Theorem 1 - "...Theorem 1 (Ramsey’s theorem). For any s, t ≥ 1, there is R(s, t) < ∞ such that any graph on R(s, t) vertices contains either an independent set of size s or a clique of size t. In particular, R(s, t) ≤ (s + t − 2,s − 1) ..."

702.(THEORY-FEATURE) Non-statistical Ontology Semantic Paths based Named Entity Recognition - related to 697 - 22 August 2019

1.A non-statistical WordNet Paths based Named Entity Recognition alternative was theoretically described earlier which exploits linguistic deep structure.
2.This alternative has been implemented in NamedEntityRecognition_HMMViterbi_CRF.py which works as below:
	2.1 find the semantic path between a PoS annotated word and unannotated adjacent word in an ontology e.g WordNet - path_between() from WordNetPath.py has been invoked (instead the text graph path between the words is also sufficient)
	2.2 for each vertex in the path find all possible PoS tags from WordNet Synsets which is a 2 dimensional list - Words * Set-of-PoS-per-Word
	2.3 find the cartesian product of the 2 dimensional list from 2.2 which enumerates all possible PoS paths between annotated and unannotated words - itertools product computes cartesian product of lists for all possible PoS paths which is a trellis
	2.4 aggregate the penultimate vertices in PoS paths
	2.5 find the majority PoS tag Synset of 2.4 which is the most likely PoS of the unannotated word
3.Logs in testlogs/NamedEntityRecognition_HMMViterbi_CRF.log.22August2019 demonstrate previous PoS paths majority voting for NER of an unannotated vertex.
4.Some paths between annotated and unannotated words drew blank because path_between() did not find a path
5.In some respects, previous algorithm resembles a transformer which is an attention model neural network scoring a sequence pipeline in transduction - it maps a sequence of words to sequence of PoS and attention context (important annotated words influencing an unannotated word) is provided by the PoS paths between PoS annotated and unannotated words in the word sequence.

703.(FEATURE) Text Graph of a natural languge text from ConceptNet 5.7 - implementation - 23 August 2019

1.NeuronRain text graph implementations so far rely only on WordNet for deciphering intrinsic deep structure merit of a text.
2.This commit implements ConceptNet 5.7 text graph alternative which was long pending.
3.New function conceptnet_text_graph() invokes least_common_ancestor graph creating function and aggregates edges from paths between all pairs of words.
4.ConceptNet5.7 Text Graph has been plotted by matplotlib at testlogs/ConceptNet5.7.TextGraph.23August2019
5.Logs for ConceptNet text graph creation are at testlogs/ConceptNet5Client.log.23August2019
6.Quality of Paths in ConceptNet is an open problem e.g https://arxiv.org/abs/1902.07831

704.(FEATURE) ConceptNet 5.7 Text Graph - shortest path filter - 25 August 2019

1.ConceptNet 5.7 Text Graph implementation has been updated to find shortest paths from least common ancestor graph for each pair of
words in the text and merge the shortest paths to create the graph by NetworkX add_path()
2.This filtering improves the path quality to some extent.
3.Centrality measures and Dense Subgraph Classification (Core Numbers) are printed.
4.logs for this commit are at testlogs/ConceptNet5Client.log.25August2019

705.(FEATURE) ConceptNet 5.7 Text Graph - additional Graph Complexity Intrinsic Merit measures - 31 August 2019

1.ConceptNet 5.7 Text Graph implementation has been updated to print a comprehensive dictionary of graph complexity intrinsic merit
measures of the ConceptNet text graph similar to WordNet text graph in RecursiveLambdaFunctionGrowth implementation
2.grow_lambda_function3() in RecursiveLambdaFunctionGrowth python implementation has been changed to accept a textgraph as argument in the absence of a text and directly grow lambda functions from argument textgraph without creating it.
3.Text is read from a file ConceptNet5Client.txt and passed on as conceptnet textgraph function argument.
4.logs for this commit are at testlogs/ConceptNet5Client.log.31August2019
5.Special concept vertex '/c/en/\xb0_c' is in all the paths found by least common ancestor algorithm and therefore the graph has a hub-spoke star structure always. This node has been filtered by a new function remove_noise() to avoid star graphs. To increase path quality more vertices might be added whenever deemed necessary in remove_noise()
6.Exception handling for non-existent paths has been included.
7.Resulting filtered textgraph has been drawn in testlogs/ConceptNet5Client5.7.TextGraph.31August2019.png and is more like a sunflower (set of subsets of vertices having pairwise intersection forming petals).

706.(FEATURE) Recursive Lambda Function Growth - Machine Translation - updated implementation - 5 September 2019

1.machine_translation() function in RecursiveLambdaFunctionGrowth.py has been changed to create a summary text in the destination language
similar to create_summary()
2.As done in create_summary() summary translated text in target language is either created by traversing the textgraph or without traversing the textgraph
3.Presently translation creates only simple sentences. Complex sentences could be created in next iteration by compounding two neighbouring sentences.
4.Goslate and googletrans python libraries have been imported for translation because of frequent Invalid Word errors in PyDictionary()
5.Clauses specific to Goslate, googletrans and PyDictionary have been created along with exception handlers
6.Caveat is rate limitations of Google Translate REST API underlying all previous three translate libraries which might or might not work depending on load and requested time point.
7.If there are exceptions, source language edges are populated in translation text graph
8.logs for previous commit are at testlogs/RecursiveLambdaFunctionGrowth.log.MachineTranslation.5September2019 - because of Google translate REST API ratelimit translation textgraph could not be created
9.nondictionaryword() filter has been commented in RecursiveGlossOverlap Classifier because of slowdown.

707.(FEATURE) Recursive Lambda Function Growth - Machine Translation - simplified - 6 September 2019

1.machine_translation() function in RecursiveLambdaFunctionGrowth.py has been simplified to remove unnecessary arguments and clauses and do the graph traversal and create summary in target language from it by default. This is because relevance_to_text() function which chooses sentences from text based on some distance measure to create summary is not necessary during translation
2.3 clauses for choice of translator - googletrans, goslate, PyDictionary - have been simplified to translate the connectives "and" and "are" to respective target language.
3.3 logs of translation for Hindi, Kannada and Telugu have been committed to testlogs:

testlogs/RecursiveLambdaFunctionGrowth.log.Telugu.MachineTranslation.6September2019

testlogs/RecursiveLambdaFunctionGrowth.log.Kannada.MachineTranslation.6September2019

testlogs/RecursiveLambdaFunctionGrowth.log.Hindi.MachineTranslation.6September2019
4.Previous logs have some intermittent errors because of Google translate API ratelimit.
5.ConceptNet is an alternative to Google Translate which is a multilingual ontology.
5.Translated text printed has some weird words not relevant to the text subject because of encoding/decoding problems of the language fonts.
6.Normal english language summary of the text - from create_summary() - is also printed
7.Usual machine translations are rule based and recently transformers (sequence attention neural network) are gaining traction.Main motivation of NeuronRain is to implement an alternative, non-statistical, formal languages theory based framework for analyzing texts exploiting deep linguistic structure in them
8.This formal languages focussed approach to measure merit of texts - NER, translation, extracting lambda expression from a text etc., is superior to inferring a neural network from text because text is approximated to a Turing machine indirectly by recursively grown lambda expressions which has strong cognitive psychology basis - Grounded Cognition - "phrasal structures are embedded recursively"
9.As far as machine translation is concerned, instead of sequence transducers, text graph of one language is mapped to text graph of another language bijectively thus preserving deep structure and meaning irrespective of language - minimizes "lost in translation".

---
708. (FEATURE) Social Network Analysis - People Analytics - PIPL.com python API integration - Syllable based name clustering - 11 September 2019 and 14 September 2019

1.PIPL.com (people dot com) is a people search engine which tries to unqiuely identify a person based on name, email etc.,
2.PIPL.com python API have been imported in SocialNetworkAnalysis_PeopleAnalytics.py and invoked in a newly defined function pipldotcom_analytics() which has query parameters: first name, last name and email
3.pipldotcom_analytics() has been invoked on 3 example emails and names and JSON response is printed in logs at

testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.11September2019 which contains identities, photos, followers and relationships sourced from multiple social networks (linkedin, twitter, facebook, google, outlook etc.,)
4.Example email queries are: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com, kashrinivaasan@live.com for the name of the author - "Srinivasan Kannan"
5.Previous query is a perfect example of multiple english name spellings and online identities author has for the single Hindi/Tamizh/Sanskrit name of the author "Sri-Ni-Va-San": Shrinivas Kannan, Srinivasan Kannan, Shrinivaasan Kannan (Shrinivaasan Ka)
6.This multiple name spelling problem arises in English because of multiple alphabets per syllable (hyphenated previously) and could easily cause identity confusion while in Tamizh, Hindi and Sanskrit name syllables have single vowel+consonant compound alphabets
7.Finding and clustering similarly named people and proving their uniqueness is a challenge in BigData because of spelling conflicts in English for similarly pronounced names.
8.Syllable based (acoustic) linguistic clustering of names by hyphenation previously as opposed to edit distance based clustering is thus a good model to classify similar names of different spellings.
9.Apart from Syllables, the 3 name spellings in previous example have same meaning which requires a Name ontology (NameNet) for semantically relating names of same meaning.
10. Names are non-dictionary words and creating a NameNet could be exhaustive and daunting.

References:

708.1 Word Hy-phen-a-tion by Com-put-er - [Liang - Ph.D Dissertation] - http://www.tug.org/docs/liang/liang-thesis.pdf - describes some difficult hyphenations and syllable boundaries in typesetting

---
709. (FEATURE) Social Network Analysis - People Analytics - Contextual Name Parsing - 14 September 2019

1.Name Parsing is an open problem and there are many tools already available (IBM Name Parser, Python Human Name Parser,...)
2.In the context of People Analytics, it is often required to find the first name, second name, middle name, surname, last name etc., from full names in a BigData set and there are cultural barriers e.g some countries have convention of <second name> <first name> and others have <first name> <second name> and <second name> is initial,surname or name of the parent/spouse.
3.Because of cultural barriers across nations parsing first name or second name in the absence of some context (e.g. an ID card) is difficult and prone to error.
4.New function nameparser() has been defined in SocialNetworkAnalysis_PeopleAnalytics.py which accepts full name, a regular expression and an ID contextual text as arguments.
5.nameparser() searches for the regular expression pattern and the tokenized full name in the ID contextual text and tries to infer First Name and Second Name.
6.An example name parsing for the author and failure of Python Human Name Parser has been demonstrated - Some past forum mails of the author from https://marc.info/?a=104547613400001&r=1&w=2 and https://marc.info/?l=apache-modules&m=105610024116012 sent from official mail id: Kannan.Srinivasan@Sun.COM have wrong first (Kannan) and last (Srinivasan) names while final salutation in the emails contain the name per Indian convention as "K.Srinivasan" and Python Human Name parser fails to parse the names correctly in the absence of this context.
7.NeuronRain nameparser() defined in this commit is passed the email texts as contexts and the nameparser() searches for places in the emails containing the

tokens of the full name by regular expression pattern.
8.Following lines define two named regex patterns which are searched in the email ID contexts mentioned in (6) as per Indian Cultural Convention of Second Name or Initial followed by First Name:
hranal.nameparser("Kannan Srinivasan",r"(?P<second_name>\w+).(?P<first_name>\w+)",emailcontext_text)
hranal.nameparser("Kannan Srinivasan",r"(?P<second_name>\w+) (?P<first_name>\w+)",emailcontext_text)
9.nameparser() returns the regex groupdicts for the matched named-parentheses
10.nameparser() has been made generic to suit all kinds of regular expressions for names and ID contexts and can be invoked for any social or professional network names. ID Context could be text from any source (e.g emails, texts, ID card texts) and regex pattern argument must comply with the ID context - if Context argument is an ID card text having Name, Address and Parent Names, UniqueID etc., regex pattern must accordingly follow e.g "Parent Name:(?P<parent_name>\w+)"
11.Logs for this commit are at testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.14September2019 and email ID contexts at testlogs/SocialNetworkAnalysis_PeopleAnalytics_NameParsing/

References:

709.1 Python nameparser - https://nameparser.readthedocs.io/en/latest/
709.2 NameAPI - https://www.nameapi.org/en/demos/name-parser/ - REST API which supports culture specifics
709.3 IBM Name Parser - https://www.ibm.com/support/knowledgecenter/en/SSEV5M_5.0.0/com.ibm.iis.gnm.parsingnames.doc/topics/gnr_np_con_parsingnamesusingnameparser.html

---
710. (THEORY and FEATURE) Compressed Sensing for Texts - Syllable Vector Vowelless Text Compression - 18 September 2019 - Related to 2, 708

1.Compressed Sensing which is mostly restricted to digital signal and image processing, reconstructs original signal from lossy signal with high probability.(https://en.wikipedia.org/wiki/Compressed_sensing)
2.Vowelless Text Compression which compresses english texts by stripping vowels also creates a lossy compressed signal from which actual text has to be recovered with high probability and thus is a Text Compressed Sensing.
3.Existing Vowelless Text Compression implementation in neuronRain relies on Hidden Markov Model estimation to infer maximum likely missing vowels.
4.Syllables of a string in any language are phonetic divisions in the string.
5.Syllable boundary name clustering for clustering similar Names differing in spellings mentioned previously maps each string to a vector of syllables which is audio/linguistic representation of the string
6.Thus every n-syllable string irrespective of languages can be represented in an n-dimensional syllable space as a syllable vector and distance between two strings is the L2 norm in this syllable space - each syllable is phonetically represented than by a script. This is an alternative word2vec embedding model and is a better distance measure than levenshtein edit distance because of language script independence - 2 strings of different natural languages can be compared by phonetic syllable distance.
7.Syllable space embedding for similarity of names in NeuronRain is different from Phonetic Match Rating algorithm (https://en.wikipedia.org/wiki/Match_rating_approach) which also removes vowels but does not consider syllable boundaries or hyphenations.
8.CompressedSensing Python implementation has been changed to include a new function syllable_boundary_text_compression() which invokes PyHyphen python hyphenator and gets syllable vector of a string and compressed syllable vector by removevowels() in TextCompression implementation.

---
711. (FEATURE) Social Network Analysis - People Analytics - Contextual Name Parsing and Syllable Vector Compression of Strings
- 18 September 2019

1.Regex Pattern Matching in nameparser() has been changed to ignore case by flag re.IGNORECASE
2.Few more ID context texts are being regex matched in nameparser() which are parsed texts and OCRs (by pdftotext and gocr) from pdf and image IDs of the author uploaded in https://sourceforge.net/projects/acadpdrafts/files/?source=navbar - Public Distribution System ID card linked to Unique Aadhaar IDs and Permanent Account Number (PAN) which is Aadhaar linked - https://sourceforge.net/projects/acadpdrafts/files/NewRationCard2.pdf/download, https://sourceforge.net/projects/acadpdrafts/files/NewRationCard1.pdf/download and https://sourceforge.net/projects/acadpdrafts/files/PAN.jpg/download
3.These three ID cards are examples of Both Indian convention of Second name followed by First name and International standard of First name followed by Second name.
https://sourceforge.net/projects/acadpdrafts/files/NewRationCard2.pdf/download, https://sourceforge.net/projects/acadpdrafts/files/NewRationCard1.pdf/download have Unique IDs linked to both formats but in different languages - Srinivasan Kannan கண்ணன சீனிவாசன
4.nameparser() is invoked on previous three ID contexts for full name "Kannan Srinivasan" and first and second names are printed
5.syllable_boundary_text_compression() from CompressedSensing implementation is invoked for 3 different spellings of same name "Shrinivaasan, Shrinivas, Srinivasan" and syllable vectors (and their vowelless versions) are printed:
==========
##########
Vowelless Syllable Vector Compression for text - Shrinivaasan : ([u'Shrini', u'vaasan'], u'Shr_n_-v__s_n')
##########
Vowelless Syllable Vector Compression for text - Shrinivas : ([u'Shrini', u'vas'], u'Shr_n_-v_s')
##########
Vowelless Syllable Vector Compression for text - Srinivasan : ([u'Srini', u'vasan'], u'Sr_n_-v_s_n')
==========
6.Distance between these syllable vectors has to be phonetic by finding syllable-wise audio similarity - every syllable is an audio waveform and two names are compared by syllable distances between vectors of waveforms e.g ["Shrini","vas"] and ["Srini","vasan"]. This has been omitted presently because syllable-to-audio tools are required.
7.Comparison to Match Rating Codex from JellyFish is also printed:
Match ratings for same name of differing spellings - [Shrinivaasan,Shrinivas,Srinivasan]: [u'SHRVSN', u'SHRNVS', u'SRNVSN']
8.logs for this commit and previous are at testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.18September2019

---
712. (THEORY) Computational Geometric Factorization - Gamma Approximation of Sum of Binary Search times - Summation for first sqrt(N) tiles and why non-average case sequential factorization is difficult - related to 668 - 19 September 2019

For sane binary search length of the tile must be > 1 => N/(k^2 + k) > 1 (or) k^2 + k < N
=> k^2 + k - N < 0
=> solving the quadratic, k = -1 + sqrt(1 + 4N) / 2

=> k = -0.5 +/- 0.5*sqrt(1 + 4N)
For large N, k < 0.5*sqrt(4N) (or) k < sqrt(N)

Total time for binary searching k=sqrt(N) adjacent tile segment arithmetic progressions by gamma approximation = sqrt(N)log N - log (sqrt(N)+1) - (2sqrt(N) + 1)log(sqrt(N)) + 2(sqrt(N)+1) - log (2*pi)
= sqrt(N)logN + 2sqrt(N) + 2 - log(sqrt(N)+1) - 2sqrt(N)log(sqrt(N)) - log(sqrt(N)) - log(2*pi)
<= sqrt(N)logN + 2sqrt(N) + 2 - log(sqrt(N)) - 2sqrt(N)log(sqrt(N)) - log(sqrt(N)) -log(2*pi)
<= 2sqrt(N)log(sqrt(N)) + 2sqrt(N) + 2 - log(sqrt(N)) - 2sqrt(N)log(sqrt(N)) - log(sqrt(N)) - log(2*pi)
<= 2sqrt(N) + 2 - 2log(sqrt(N)) - log(2*pi)
<= 2*(sqrt(N) - log(sqrt(N)) + 1) - log(2*pi)
<= 2*(sqrt(N) - log(sqrt(N)))
which is exponential in logN still shaving off only a logarithm from squareroot of N. This necessitates finding approximate factors close to actual factors in average case sequential optimization previously implying factorization (finding atleast one factor) is in average-P which is an average lowerbound for all possible integers and different from P. Therefore deterministic factorization to find one factor or all factors requires NC (PRAM or BSP) parallelism - NeuronRain Spark Factorization which is both a PRAM (multicore) and BSP(cloud) model finds all factors. It has to be noted previous average case sequential time is for finding atleast one factor.

---
713. (THEORY and FEATURE) People Analytics - Set Partition based Electronic Voting Machine implementation and a NeuronRainApps Drone usecase - 24 September 2019 - related to 620,648

Voter Received Encrypted Paper Audit Trail (VREPAT) Theoretical Voting Machine based on Set Partition Bucketing described previously is partially implemented as a futuristic drone usecase (Autonomous Delivery Usecases - https://syncedreview.com/2019/08/24/autonomous-delivery-moves-from-research-labs-to-the-streets/) in this commit - this is not a full-fledged implementation but conceptual:

(*) Usual state-issued unique id(s) are singletons stored in centralized cloud and vulnerable to breaches/fakes creating identity disputes.

(*) To redress this an alternative dual unique id (public id and private id) similar to public-key-infrastructure is assumed which is as strong as RSA and Diffie-Hellman protocols

(*) Public unique id is a hash digest of a string which uniquely identifies a person e.g concatenation of <VoterID><Full Name><Parent/Spouse Name><Permanent Address> is always a unique public id. Private id is a password for a voter issued along with a Public unique id.

(*) Polling Station is a Drone - it contains a set partition bucketing voting machine which based on addresses or public unique id(s) in the voter list automatically navigates to an address and lands on a voter doorstep. Voter casts h(is|er) vote on the drone booth by public unique id card and private password and receives encrypted receipt of vote.

(*) This Dynamic Drone Polling Booth is inspired by Autonomous Online Shopping Delivery and Mobile ATMs available already (which work on smartcard and password authentication) and has many advantages over traditional static polling stations which satisfy constraints of John Hopkins Paper to some extent:

(*) Voting need not be limited to a day and can sprawl multiple days.

(*) Voter can do advanced booking for drone and it lands on demand making the voting behaviour difficult to predict.

(*) All households are covered and thus maximizes polling percentage

(*) Completely automated election - Minimizes or obviates polling stations, cost, queueing and personnel and thus human tamper-proof - requires one drone per polling precinct of few hundred voters

(*) Ensures privacy
(*) Reduces malpractices because of no crowding, each voter independently votes, polling happens at doorstep at time of choosing which is unpredictable and unique demographic/geographic identity is electronically verified by drone navigation to unique address minimizing bogus voting (no voter is denied voting because it could not have been already cast in h(is|er) name - GPS drone navigation is an additional geographic uniqueness).

(*) This commit defines a new minimal voting function electronic_voting_machine() in Streaming_SetPartitionAnalytics.py which accepts voted_for, example voter id card contexts of the author from testlogs/Streaming_SetPartitionAnalytics_EVM/ (no text parsing is done) and makes ripemd160 hash digests from them. EVM is a dictionary set partition object mapping candidate voted_for to list of voters' public uniqueid hash digests - does not use counters and private id authentication.

(*) Complexity theoretically, drone voting is reducible to Motion planning (A-Star algorithm) and travelling salesman problem (TSP) because drone has to efficiently navigate the voter vertices spread across geographically (e.g hamiltonian) and thus NP-complete.

---
714. (THEORY and FEATURE) Topological Handwriting Recognition - Product Homotopies - Related to 159,660 - 30 September 2019

Pattern grammars for shape description for handwriting recognition previously as homeomorphic deformations in R^2 preserving genus (also defined by knots - https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt) have an alternative topological definition in terms of Product of Path Homotopy classes: Two paths f and f' are homotopic if they have same initial and final points x0 and x1 and there is a continuous map F:I * I ---> X for some topology space X such that F(s,0) = f(s) and F(s,1) = f'(s); F(0,t) = x0 and F(1,t) = x1. If f is a path in X from x0 to x1 of homotopy F and g is a path from x1 to x2 of homotopy G, product homotopy F*G by pasting lemma is the set of all continuous paths from x0 to x2 via x1. An example of product homotopy is handwritten digit '8' which has a pattern grammar:

<8> := <o> pasted on <o>

Set of closed paths (all possible ways of writing 'o' - initial and end points are same) for <o> form a path homotopy F and product homotopy F*F defines all possible ways of writing '8' by pasting together endpoints of two handwriting homotopies F and F (also referred to as path composition). Similarly every handwritten alphabet can be defined as product of homotopies by pasting together path initial-end points. For example letter 'p' has pattern grammar:

<p> := <|> pasted to the left of <o>

In other words homotopy is a continous deformation of a path having end points fixed. It has to be noted that previous homotopy definition does not distinguish handwritings of two humans - it just recognizes any written alphanumeric as a product homotopy. Distinguishing two humans requires fine definition of homotopies within thresholds (e.g handwritings of same person hardly deviate from thresholds and some paths do not exist in handwriting homotopy of one person but do in other person's). This topological information of handwritings can complement traditional neural network based recognitions.

References:

714.1 Topology - [James R.Munkres] - Second Edition - Page 108 - Theorem 18.3 - Pasting Lemma - Pages 322-326 - Path Homotopy and Product Homotopy - Figures 51.1, 51.5
714.2 Path Homotopy and Path composition - https://en.wikipedia.org/wiki/Path_(topology)#Homotopy_of_paths

---

---
715. (THEORY) People Analytics - Topological Face Recognition - Product Homotopies - related to 714 - 2 October 2019

Previous section defined Handwriting recognition topologically by Product Homotopies. Face Recognition and Handwriting recognition are crucial for unique identification of a human in people analytics. Extending Product Homotopies to Face Recognition is not straightforward because facial features are not continuous functions which can be unified by pasting lemma. Possible solution is to segment the image by either bounding boxes or watershed algorithm and find contours of facial features which are piecemeal discontinuous curves (approximable by Douglas-Peucker polynomial). Each bounding box or segment containing only continuous curves have to be unified by pasting lemma which define product homotopies of continuous deformations in 3-dimensions whereas handwriting homotopies are deformations in 2D. Example: set of images of all possible postures, expressions and facial distortions of a human face can be equated to union of product homotopies of continuous contour curves in bounding boxes or segments in the image - Contours of Eyes, Nose, Mouth and Ears of a face cannot be a single continuous curve and can be in separate segments or bounding boxes of an image each defined by a product homotopy of continuous contours. Union of product homotopies for Eyes, Nose, Mouth, Ears and any other feature recognizes a face including all possible facial deformations.

Example Shape Grammar for Face Recognition - Union of Product Homotopies (each <feature> defines a product homotopy for one continuous segment of facial feature contour deformations):

<Hair>
<Ear><Eye><Eye><Ear>
<Nose>
<Mouth>

---
716. (THEORY and FEATURE) Economic Intrinsic Merit - Gravity Model of Volume of Trade and GDP as fitness measure - 2 October 2019

1.Intrinsic Fitness or Intrinsic Merit has been traditionally applied to denote innate ability of a vertex in Social Networks. Fitness model is applicable in Economic Networks too e.g International Trading Network(ITN), Production Networks(https://economics.mit.edu/files/9790).
2.https://arxiv.org/pdf/1409.6649.pdf discusses various measures of Economic fitness including GDP and quantifying volume of trade between nations - Quoted excerpts:

2.1 "...The standard model of non-zero trade flows, inferring the volume of bilateral trade between any two countries from the knowledge of their Gross Domestic Product (GDP) and mutual geographic distance (D), is the so-called ‘gravity model’ of trade [21–25]. In its simplest form, the gravity model predicts that the volume of trade between countries i and j is Fij = α GDP^βi · GDP^βj / Dij^k..."

2.2 "...where zi ≡ e^−θi and pij ≡ zi*zj/(1+zi*zj). The latter represents the probability of forming a link between nodes i and j, which is also the expected value <aij> = zi*zj/(1 + zi*zj)= pij . (4) ..."

2.3 "...It should be noted that eq.(4) can be thought of as a particular case of the so-called Fitness Model [41], which is
a popular model of binary networks where the connection probability pij is assumed to be a function of the values of some ‘fitness’ characterizing each vertex. Indeed, the variables ~z∗ can be treated as fitness parameters [6, 37] which control the probability of forming a link. A very interesting correlation between a fitness parameter of a country (assigned by the model) and the GDP of the same country was found [37]. This relation is replicated here in Fig. 2, where the rescaled GDP of each country (gi ≡ GDPi/sum(GDPi)) is compared to the value of the fitness parameter z∗i obtained by solving eq.(5). The red line is a

linear fit of the type zi = √a · gi...."
3. This commit defines a new function EconomicMerit() in LinearAndLogisticRegression.py which computes trade link probabilities and gravity model between countries based on GDP as intrinsic fitness measure and distances between countries. Reference 694.1 defines GDP fitness as a linear regression dependent on macroeconomic variables.

References:

716.1 Another Fitness Model of GDP - Fitness-dependent topological properties of the World Trade Web - https://arxiv.org/pdf/cond-mat/0403051.pdf

---
717. (THEORY) Cognitive Element in the context of Category theory, Factorization and Set Partitions - related to 160 - 4 October 2019

Notion of "Cognitive element" for the lack of better name was defined earlier as an element in an array, set partition, group, category or any algebraic structure which "knows" or has "information" about other peer elements - this is an experimental abstraction in Category theory which might have overlaps to existing algebraic concepts (left inverse of a morphism - retraction/section - https://en.wikipedia.org/wiki/Section_(category_theory)). Information could be any equivalence relation or morphism - "member-of","factor-of","subset-of" etc., defined in a Category. Examples:

717.1 117 = 13*9 is an element of ring of integers as category - 13,9 are "factors-of" 117 or 13 "knows" about 9 by factorization. Factorization is a morphism relation defined on category of integer objects - Inverse([13,9])=117

717.2 Following set partition which maps timeout values to buckets of processes having that timeout value in an OS scheduler is a category(Hashing Dynamic Sets - https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt):

10 - 1,2,5
20 - 12,15,11
33 - 3,111
41 - 16
55 - 25,114

and every bucket defines a timeout morphism relation between objects in the processes category - Timeout Bucket 55 has processes 25, 114 which know each other.

717.3 Every morphism equivalence relation on any Category defines equivalence classes elements of which have "information" about peers.

717.4 Cognitive elements have useful applications in set partition equivalence relations - in previous set partition example, question of inverse - finding a bucket containing an element - is non-trivial e.g find the bucket having element 111. This requires an exhaustive search because set partition is not a bijective map and reverse lookup is not possible (but there are bidirectional dictionary implementations in Python which can retrieve a key from value - https://pypi.org/project/bidict/). Elements 3,111 are cognitive peers by inverse morphism relation "in the bucket of 33" - Inverse([111,3]) = 33. Implementing such an inverse requires knowledge of cognitive elements which are domains of the inverse morphism.

---
718. (THEORY and FEATURE) People Analytics - Set Partition Analytics based Drone Electronic Voting Machine - updated - 9 October 2019 - related to 713

1.Streaming_SetPartitionAnalytics.py has been changed to update electronic_voting_machine() function - New semaphore mutual exclusion

synchronized primitive block (BoundedSemaphore) has been added to guard the voting critical section. Maximum value of semaphore is set to 1 voter.
2.Voting code has been updated to check if voter has already voted by Voted list and skip voting if yes. Voter id(s) are appended to Voted array after voting within synchronized block.
3.Additional encryption has been introduced by invoking SHA256 Passlib encrypt on the ripemd160 publicuniqueid hex hash digest and append the encrypted voter id to EVM bucket.
4.This EVM assumes the function electronic_voting_machine() is invoked from a Drone python code which should have already navigated to the voter address and done the user/password authentication by smartcard.
5.Single Signon username/password authentication has already been implemented in NeuronRain GUI - python-src/webserver_rest_ui/NeuronRain_REST_WebServer.py - by passlib OAuth having MongoDB and Redis credential storage - which might as well be used as alternative caller for this EVM.
6.logs for preventing repetitive voting and SHA256 buckets are at testlogs/Streaming_SetPartitionAnalytics.EVM.log.9October2019
7.This Set Partition EVM is also a Dynamic Graph Partition (Similar Worst Case Execution Time - Survival Index - Dynamic Graph Partition OS Scheduler is described in https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt - Hashing Dynamic Sets) because voting is dynamic and voters related among themselves might vote for different candidates which creates edges of dynamic people partition graph (voter vertices related by edges are partitioned into candidate buckets)
8.As mentioned in previous section, Left Inverse of Voter Bucket Objects is the Candidate voted for connecting peer voters by an equivalence relation.

---

719. (THEORY) Complement Diophantines over Integers (Z), Polynomial Interpolation, Polynomial Reconstruction Problem, Error Correcting Codes, MRDP theorem, Lagranges Four Square Theorem and Decidability of Complementation, Ramsey 2-coloring of sequences, Set Partitions to Tile Cover reduction, Factorization, Lagrangian tiling of 2-dimensional space, ZF and Axiom of Choice (AOC) and Complement Choice functions, Depth-2 majority voting social choice circuits - an alternative interpretation based on diagonalization - 13 October 2019 - related to 345, 471, 617, 646, 651, 657

By MRDP theorem, every recursively enumerable (RE) set of Integers is diophantine representable and there are RE sets which are not recursive (R) = RE \ R (RE set difference R). RE \ R languages have diophantine representations but are not accepted/rejected by a halting turing machine (without looping) which includes a turing machine for finding the complement diophantine representation of a language in RE \ R. Such a halting turing machine (TM1) in RE \ R or an algorithmic equivalent for finding complement diophantine representation of other sets in RE \ R is a contradiction. This diagonalization proves undecidability of complement diophantine for RE \ R (diophantine representation exists but algorithm does not for RE \ R).

For Recursive sets (R), there are algorithms available or there exists a halting turing machine (TM2) in R for finding algorithmic equivalent of diophantine representation of other sets in R and diagonalization does not contradict e.g Polynomial Interpolation (exact), Polynomial Reconstruction Problem (error correcting polynomial agreeing on most points - reference 345.2), Lagrange four square theorem. Lagrange four squre theorem maps tuples in 4-dimensional space to points on recursive set (or) any integer can be written as sum of 4 squares. This 4-tuples to integer map from Lagrange four square theorem is infact an interpolation by a polynomial over 4 variables - f(x1,x2,x3,x4) = x1^2 + x2^2 + x3^2 + x4^2 = y. Lagrange Four Square interpolation indirectly solves factorization by solving an optimization problem - set partition is reduced to tile cover of a rectangular 2-dimensional space and factors are sides of the

rectangle which have to be solved for by an integer program.(Both diophantine representation and algorithm exist for R). Chinese Remainder Theorem generalizes this as an isomorphic map in multiple dimensions - x mod N <=> (x mod n1, ..., x mod nk) for coprime n1,n2,...,nk and N=n1*n2*...*nk.

This diagonal interpretation is more dichotomous than previous sections on complement diophantine undecidability which in a nutshell implies: Any set is a complementary set of some other set and thus any complementary set which is recursively enumerable but not recursive has a diophantine representation and not decidable by an algorithm, while any recursive complementary set is decidable by an algorithm and also has a diophantine representation.

As a special example complement of Axiom of Choice for depth-2 majoriy boolean circuits is phrased as: For set of electorate sets S={s1,s2,s3,...} there exists a choice function C'(si)=(x1,x2...) which flips a chosen majority (0 or 1) from every set si in S. Leaves are members of S (electorate sets or constituencies) while depth-1 are complemented/flipped bits from majority voting of leaves. This complements AOC democracy function as: "Who won" Vs "Who lost" (alternatively can be complemented as "Who voted in favour" versus "Who did not vote in favour" which has hardness amplification ramifications for proving P != NP). It is open if Complement AOC is decidable. Traditional democracy is depth-2 majority circuit and abides by AOC.

Complement Diophantines throughout this draft focus mainly on sets or sequences of Integers (Z) because MRDP theorem and Rayleigh-Beatty sequences theorem are limited to Z while complementation over Rationals(Q) is open problem and complementation over Reals (R) is solvable by Tarski/Sturm decidability.

---
720. (THEORY and FEATURE) People Analytics - Handwriting Recognition - Contour Homotopies - Matplotlib Rasterization - 16 October 2019 - related to 714 and 715

1.This commit updates handwriting_recognition() in DeepLearning_ConvolutionNetwork_BackPropagation.py to additionally raster the contours extracted from the handwritten alphanumeric.
2.Rasterization presently invokes matplotlib plot() having parameter rasterized=True and presets dots-per-inch parameter to 100 in figure() for restricting number of pixels.
3.Contours from each recognized written alphanumeric are looped and plotted by matplotlib plot() setting rasterized=True.
4.There are better alternatives to this primitive rastering - rasterio (https://rasterio.readthedocs.io/en/stable/) and wavelet-rasterization (https://github.com/ufoym/wavelet-rasterization/) and desired rasterization is similar to illustrations in https://github.com/ufoym/wavelet-rasterization which is a coarse rectification required for hyperbolic arc in Computational Geometric Factorization (But there is a windows DLL build dependency). Matplotlib dots-per-inch and rasterized=True creates fine grained rasterized curves.
5.Each contour is approximable by a Bezier-like curve (https://en.wikipedia.org/wiki/B%C3%A9zier_curve) which is an instance of homotopy because bezier curves have endpoints/control points unchanged and intermediate points on curve are puppet-animated (most applicable in Graphics).
6.Handwriting can be simulated by a Bezier curve having unchanged endpoints or control points which is a homotopic continuous deformation.
7.Original handwriting is reconstructible by homotopy and pasting lemma - by pasting together polynomials for individual contours.
8.Logs and images for this commit are at testlogs/ (handwritten 1(style 1) and 1(style 2), 1 and 8 are contour-rasterized and compared):
DeepLearning_ConvolutionNetwork_BackPropagation.log.tar.xz
DeepLearning_ConvolutionNetwork_BackPropagation.Raster.Homotopies_1_1.jpg
DeepLearning_ConvolutionNetwork_BackPropagation.Raster.Homotopies_1_2.jpg
DeepLearning_ConvolutionNetwork_BackPropagation.Raster.Homotopies_8_1.jpg
DeepLearning_ConvolutionNetwork_BackPropagation.Raster.Homotopies.pdf

9.Every contour can be fit to a bezier polynomial and two alphanumerics have to be verified if they are homotopic (has a dependency on topology libraries). This is non-trivial because every homotopically "similar" contour in two alphanumerics have to be pairwise compared
and number of contours per alphanumeric could be in hundreds.

---
721. (THEORY and FEATURE) People Analytics - Handwriting Recognition - Inner Product Space of Contour Interpolated Polynomials - 17 October 2019 - related to 714,715,720

1.In continuation of the previous commit for drawing rasterized contour curves, handwriting contours are interpolated by SciPy.interpolate splprep()
2.Points on contour curves are stack()-ed to 2 dimensional arrays and contours having more than 3 points are interpolated by spline polynomials of degree less than number of points per contour.
3.PdfPages "pdf damaged" error in DeepLearning_ConvolutionNetwork_BackPropagation.Raster.Homotopies.pdf has been resolved by PdfPages close() which flushes the buffer.
4.splprep (https://docs.scipy.org/doc/scipy-1.2.1/reference/generated/scipy.interpolate.splprep.html) creates B-Spline curves which are exact but less smooth as opposed to Bezier curves which are smoother, depend on control points and are approximate fit (passes through most of the points).
5.Boolean flag TopologicalRecognition=True/False has been introduced in __main__ to choose topological handwriting recognition instead of Convolution Neural Network Pattern Recognition.
6.Logs and Pdf images for contour polynomials are at:
testlogs/DeepLearning_ConvolutionNetwork_BackPropagation.log.17October2019
testlogs/DeepLearning_ConvolutionNetwork_BackPropagation.Raster.Homotopies.pdf
7.By defining an inner product space of contour polynomials <f,g>=DefiniteIntegral(f(x)g(x)dx), distance function is obtained for verifying if 2 contours belonging to different handwritten alphanumerics are "similar" or "orthogonal" (polynomials f and g are orthogonal if <f,g>=0). This is an algebraic but less topological similarity.

References:

721.1 Orthogonal Polynomials - https://www.math.tamu.edu/~yvorobet/MATH304-503/Lect4-04web.pdf

---
---
722. (THEORY and FEATURE) Update - Drone Electronic Voting Machine NeuronRainApps Usecase - 22 October 2019 - related to 713

722.1 ROS Drone Navigation Algorithms which depend on Google Maps for GPS geolocation have been cited as references in NeuronRain Documentation and Licensing: http://neuronrain-documentation.readthedocs.io/en/latest/ - Distance is computed by geodesic geometric equations (Bearing). Following are quoted excerpts from https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0214-3:

722.1.1 "...In the experiment that we did, the average of positional deviation of landing position between the actual landing position and the desired landing position in the flight tests of flying from start to goal is 1.1125 m and for the tests that use the algorithm which uses course-over-ground, the positional deviation has average of 2.39 m...."

722.1.2 "...The recent developments of drone for sprayer pesticide applications and used for delivering items, for example the Amazon Prime Air, where Amazon used an octocopter to deliver items with weighs less than 5 lb or around 2.3 kg..."

722.2 Commercially experimented drone online shopping grocery delivery mechanisms have payload of few kilograms.
722.3 Drone Electronic Voting Machine has an accompanied Point-of-Voting handheld device for smartcard voter id authentication.
722.4 Drone navigates to an address by Google maps from a voter list (powered by ROS or linux kernel PXRC drivers).
722.5 Once drone lands on doorstep, Voter authenticates in the Point-of-Voting machine with in drone by voter id smartcard.
722.6 Ballot is displayed in a touchscreen of the drone. At the end of the voting drone Point-of-Voting device issues a VREPAT encrypted receipt of the vote and retreats.
722.7 Drone EVM does not have payload like autonomous delivery - thus lightweight.
722.8 Landing deviation has to be specified upfront as config parameter.

References:

722.9 UAV Theory, Design and Applications - [AR Jha] - https://content.taylorfrancis.com/books/download?dac=C2014-0-36678-5&isbn=9781498715430&format=googlePreviewPdf

---
723. (THEORY) Update 2 - Drone Electronic Voting Machine - Voting Analytics - Set Partition Distance Measures - 2 November 2019 - related to 722

723.1 Drone Electronic Voting machine partial implementation (depends on drone navigation) in NeuronRain is an obfuscated set partition of electorate (voters identity per candidate bucket is encrypted)
723.2 Analytics inferences can be drawn from a stream of previous encrypted set partition EVMs without compromising voter privacy.
723.3 Variety of distance measures are available for comparing set partitions:
	723.3.1 Earth Mover Distance or Wasserstein Distance (OpenCV2 and SciPy) - which is amount of work necessary to move a mass of one shape to another - shape is a multidimensional object and two set partition EVMs are of different shapes - https://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html?highlight=emd#cv.CalcEMD2 and https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wasserstein_distance.html - EMD2 in OpenCV2 is a sophisticated version of EMD for comparing two visuals by mapping images to histogram probability distributions while EMD in SciPy is wasserstein distance
	723.3.2 OpenCV2 provides many other measures for comparing image histograms - https://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html - cvCompareHist() by Correlation, Chi-Square, Intersection,Bhattacharyya,Hellinger distance measures

---
724. (FEATURE) Update 3 - Drone Electronic Voting Machine - JSON persistence of EVMs - Datasource for Voting Analytics - 2 November 2019 - related to 723

724.1 electronic_voting_machine() function in Streaming_SetPartitionAnalytics.py has been updated to accept 3 arguments - voting machine set-partition obfuscated dictionary, unique id context and candidate voted for - for removing hardcoded voting dictionary and being generic across polling stations.
724.2 Voting machine dictionaries are iterated and mapped to histograms of Candidate to Votes Counter (size of bucket)
724.3 Voting machine dictionaries are JSON dumped to a file testlogs/Streaming_SetPartitionAnalytics.EVMs.json - this JSON persistence is only for analytics purposes and voting machine dictionary is still an in-memory object per drone EVM supposed to be passed on from an already navigated Drone python code which is lacking.

724.4 This JSON file can be read as a stream by Streaming Abstract Generator Dictionary datasource - Streaming_AbstractGenerator.StreamAbsGen("DictionaryHistogramPartition","testlogs/Streaming_SetPartitionAnalytics.EVMs.json") and any histogram analytics function can be invoked on the stream.
724.5 electronic_voting_machine() is invoked 9 times for 2 voting machines, 3 voter id contexts and 3 candidates and electronic_voting_analytics() function which implements the previous JSON persistence is invoked after - logs at testlogs/Streaming_SetPartitionAnalytics.EVM.log.2November2019
724.6 Streaming_AbstractGenerator.py "DictionaryHistogramPartition" __iter__() clause has been updated to do JSON load() of dictionaries instead of AST literal_eval() and yield the dictionaries as stream.

---

725. (THEORY) Drone Electronic Voting Machine, Voting Analytics and Forecasts, Bertrand Ballot Theorem, Money Changing Problem, Set Partitions, Depth-2 Boolean Majority Circuit - 3 November 2019 - related to 657, 666

Drone Electronic Voting Machine partial implementation previously is only a majority voting simulation - Voted dictionary hardcoded at present which prevents duplicate voting in Streaming_SetPartitionAnalytics.py (if publicuniqueid not in Voted:) has to be either in a centralized NoSQL store or localized per drone for global uniqueness so that no voter casts vote in 2 drone EVMs (which is already guaranteed by geographic uniqueness - drone autonomously navigates to an address and authenticates voter smartcard by point-of-voting device - in some aspects this is similar to old tech postoffice protocol and a betterment of postal ballots: Sealed paper mails are delivered uniquely to a name and address which is a public unique id and there is no password. Tampering is easily verifiable if there is damage to courier seal) - Localized Voted dictionary requires voter lists per drone. Clustering of EVM Set partitions by either Earth Mover Distance or other aforementioned distance measures in OpenCV2 classifies EVMs of similar voting patterns in single cluster (non-invasive because histogram depends on per candidate counters alone). This clustering of EVMs corresponds to depth-1 of depth-2 non-boolean Majority Voting Circuit described schematically in 657 and reflects per segment trend. In boolean setting, empirically EVMs might be clustered into 2 disjoint partitions for candidates 0 and 1. It has to be noted that set partition of votes is available only post-poll while prepoll numbers are only total votes - inferring post-poll votes set partition from pre-poll total votes is reducible to function:

f(n) = c1 + c2 + ... + ck (n votes partitioned to k candidates)

and has probability 1/p(n,k) where p(n,k) is restricted partition number of n for k parts - restricted partition and money changing problem/coin problem are equivalent. Probability of candidate of biggest part (c0) being always ahead in counting by Bertrand Ballot theorem is c0-c1/(c0+c1) if f(n) = c0 + c1 in boolean majority for candidates 0 and 1.

---

726. (THEORY and FEATURE) Drone Electronic Voting Machine - Pseudorandom Majority Voting Balls-Bins Simulation and Voting Analytics - 3 November 2019 - related to 620,725

726.1 electronic_voting_machine() function in Streaming_SetPartitionAnalytics.py has been updated to import random, sklearn packages for histogram analytics.
726.2 evm_histograms dictionary of EVM dictionaries has been made a list of EVM dictionaries which is dumped to JSON persistence. EVM stream is read from Streaming Abstract Generator - DictionaryHistogram JSON datasource testlogs/Streaming_SetPartitionAnalytics.EVMs.json and ARI(adjusted rand index), AMI(adjusted mutual information index) and EMD(earth mover distance) histogram distance functions are invoked from sklearn and SciPy for every consecutive pair of EVM histogram set partitions.

726.3 Exceptions for dissimilar or empty histogram shapes in ARI,AMI and EMD have been handled.
726.4 Majority Voting is simulated by a poll loop of 10 voters, 3 unique id contexts, 3 candidates and 3 electronic voting machines - only 3 votes are allowed and others are discarded as "already voted".
726.5 Majority Voting simulation is by python pseudorandom number function random() for randomly choosing candidate to vote for - bounds for the balls and bins coupon collector problem described in 620 apply.
726.6 These 3 EVMs are analyzed pairwise by electronic_voting_analytics() for ARI, AMI and EMD-Wasserstein distance.
726.7 Logs and EVMs JSON for previous simulated election are at Streaming_SetPartitionAnalytics.EVMs.json and Streaming_SetPartitionAnalytics.EVM.log.3November2019
726.8 calcEMD2() is not available on OpenCV2 Python (error: no name found) though recently released OpenCV3 Gold has its C++ counterpart - OpenCV2 histogram comparison function compareHist() requires UMat format.
726.9 Streaming Abstract Generator - Dictionary Histogram __iter__() clause yields dictionary instead of entries, restoring status quo ante.

---
727. (THEORY) Fame-Merit Equilibrium and Converging Markov Random Walk on World Wide Web, Condorcet Jury Theorem, Bipartite Decomposition of World Wide Web, Margulis-Ruzzo Threshold - Conjecture - 12 November 2019 - related to 572 and all sections on Intrinsic Merit, Fame and Majority Voting
---

727.1 Page Rank is a converging markov random walk based Fame ranking for World Wide Web and is a peer-to-peer markov chain voting mechanism.
727.2 Fame-Merit Equilibrium mentioned theoretically earlier, is an alternative reputation ranking which tries to find an equilibrium approximation point between absolute intrinsic merit and perceived fame of an entity.
727.3 Both Fame-Merit Equilibrium and PageRank are based on directed graph having weighted edges between perceived and perceiver vertices and thus are two formats of Condorcet Jury Theorem and its derivatives([Black],[KKLadha]) or Margulis-Ruzzo Threshold for boolean majority CJT (Voters or Perceivers err in judging intrinsic merit of the perceived and error probability tilts the group decision either way beyond a threshold) - mistake-minimized infinite voters/jury implies Nash equilibrium is attainable in either variants of reputation rankings - PageRank and Fame-Merit equilibrium are two sides of Condorcet Jury Theorem Coin.
727.4 Fame-Merit equilibrium in the context of algorithmic economics finds Market Clearing Prices for commodities (which are between intrinsic and perceived prices) by Convex Programs and is schematically drawn as bipartite graph of edges between two sets of vertices - Perceivers(Buyers) and Perceived(Items). Every perceiver has a total disposable price merit which has to be apportioned amongst perceived items by the convex program (Eisenberg-Gale, Fischer, Nash Equilibria). Similar equilibria can be found for any Perception-Absolute (Fame-Merit) problem irrespective of entity being judged - Fame is an image possibly distorted while Merit is real.
727.5 PageRank finds ranking of an entity by a converging markov random walk on peer-to-peer world-wide-web weighted directed graph of Perceiver-Perceived (huge graph of few billion vertices)
727.6 Equating PageRank and Fame-Merit equilibrium is of theoretical importance because the two are different mathematical gadgets - convex program and converging markov chain random walk.
727.7 Fame-Merit equilibrium convex program for world wide web should also be a huge bipartite graph of few billion perceivers and perceived on the internet.
727.8 Solutions to Fame-Merit equilibrium convex program strike an approximate midpoint between absolute intrinsic merit and perceived fame.
727.9 Natural question begotten by equating rankings from fame-merit convex program and PageRank markov chain is: Are two rankings equivalent? Answer to this question is conjectured as yes in 727.3 supra. Proof/Disproof of this conjecture might involve equating a generic convex program for a huge bipartite

graph of p perceivers and q perceived vertices and PageRank of the same graph of p+q vertices. Or Nash equilibria have to be found for both Fame-Merit and PageRank and checked for equality.
727.10 High Correlation coefficients between the two rankings prima facie might indicate coincidence. Challenge is to conceive the world-wide web as a whole to be a bipartite graph of perceivers-perceived though some components of the internet might be bipartite - this might require algorithms for bipartite decomposition - world wide web directed graph is decomposed to bipartite subgraph components and fame-merit equilibrium is computed for each bipartisan component.

References:

727.11 Bipartite Decomposition of Random Graphs - http://www.cs.tau.ac.il/~nogaa/PDFS/bipdec2.pdf - World Wide Web is theoretically modelled as a random graph. Its bipartite decomposition leading to computing fame-merit equilibrium for every bipartite component is stated as conjecture previously vis-a-vis PageRank.
727.12 Demand-Supply and Market Clearing Prices - Walrasian-Arrow-Debreu, Fisher Markets - [Yishay Mansour] - https://m.tau.ac.il/~mansour/advanced-agt+ml/scribe-10-market-EQ.pdf - Market Clearing Price Equilibria viz., Eisenberg-Gale apply to a market in which demand equals supply and all products are sold with no remaining demand. But actual scenario mostly is the inequality of demand and supply (Demand-Supply curves - https://en.wikipedia.org/wiki/Supply_and_demand) defined by equation Demand * Supply = constant (Demand is inversely proportional to Supply). Demand is also a perception and can be defined as perceived importance or utility of a commodity and thus a function of perception/utility:

```
utility = function_of(demand) or function_of(1/supply)
```

Glut of any entity diminishes perceived importance and dearth increases it. Demand-Supply equilibrium can be generalized to any entity - text,audio,video,people - merit of which has to be judged e.g rare texts are more valued, rare visuals are more sought after,rare earths are costly.
727.13 Artificial Intelligence and Economic Theories - https://arxiv.org/pdf/1703.06597.pdf - Individualized Supply-Demand curves for pricing in ecommerce platforms (Amazon, EBay et al), Granger Causality - one economic time series causes the other with a lag - https://en.wikipedia.org/wiki/Granger_causality, Pricing - Ricardo labour theory of value - price is proportional to labour went into manufacturing goods and services (Neuro MAC Cryptocurrency, Pricing and Market Equilibrium - https://github.com/shrinivaasanka/kingcobra64-github-code/blame/master/KingCobraDesignNotes.txt).

---
728. (THEORY and FEATURE) Set Partition to Tile Cover Reduction by Computational Geometric Factorization - Least Squares Approximate Solution for the underdetermined linear equations, Exact Money Changing Problem(MCP) Solution, Quadratic Programs, Pell Equation, Shell Turing Machines(Category of Topological Spaces) - related to 577,624,651 - 19 November 2019, 20 November 2019,6 December 2019, 24 January 2020, 26 January 2020, 27 January 2020

1.A random set is partitioned by SymPy set partition API and a partition of the set is chosen at random. Random set partition is mapped to a histogram by map(len,) and setpartition_to_tilecover() is invoked. New parameter for number to factorize has been added in setpartition_to_tilecover().
2.An integer is factorized by Spark Computational Geometric PRAM factorization by subprocess.call() invocation and factors of N=pq are read from JSON persistence.
3.Following underdetermined system of equations is solved approximately by LSMR least squares for finding arrangement of tiles on the periphery of the rectangle and the fractional solution vector is rounded off to 1 or 0 based on a threshold:

```
      #c1*x1 + c2*x2 + ... + ck*xk + ... + cn*xn = p
      #d1*x1 + d2*x2 + ... + dk*xk + ... + dn*xn = q
      #solve AX=B:
      #X = [c1 c2 ... cn] - unknowns (boolean include or exclude of a tile)
      #A = [[x1 x2 ... xn]  - knowns (sides of the square tiles)
      #     permutation_of[x1 x2 ... xn]]
      #B = [p q] - factors - pq=N
```

4.Solving underdetermined equations by most libraries including SciPy, NumPy, SymPy etc., yield only approximate solutions because of Rouche-Capelli theorem limitation. In the previous system of equations coefficient matrix A has two rows of square tile sides from Lagrange four square theorem and one of the rows is permuted for randomness (reverse() is invoked) and unknown tile choice vector (d1,d2,...,dn) should also be some permutation of (c1,c2,...,cn).

5.This commit approximately solves the previous underdetermined linear equations by lsmr() function of scipy.sparse.linalg and rounds the real solution vector to binary vector. Mutual exclusion of tiles is by an if-else clause which chooses 1 tiles for side1 and 0 tiles for side2 of rectangle. But product of sum of sides won't be equalling N always because of error allowed in LSMR and rounding. Sophisticated PRAM GEPP implementations might do better at finding the tile arrangment vector (c1,c2,...,cn)

6.logs for this commit are at testlogs/Streaming_SetPartitionAnalytics.log.19November2019

7.An example Lagranges Four Square reduction for integer 18 and its random partition=[4,10,4]:

```
            18 <=> [4,10,4] <=> [1+1+1+1]+[1+4+4+1]+[1+1+1+1]
            18 <=> 2*9 <=> Solutions for previous tile arrangement system of
```

equations for the rectangle periphery: [1+1]*[2+2+1+1+1+1+1+1]

8.Each of the two linear equations can be interpreted as instance of Money Changing Problem(MCP) or Coin Problem. Sides of the square tiles are coins and following linear system:

```
      c1*x1 + c2*x2 + ... + ck*xk + ... + cn*xn = p
      d1*x1 + d2*x2 + ... + dk*xk + ... + dn*xn = q
```

equates factors of N=(p,q) to Linear Sum of sides of lagrangian square tiles (1-dimensional) for the length and breadth of rectangular periphery. Both of the earlier MCP equations are NP-Hard (Strongly or Weakly depending on encoding) which involves finding the integer vector [c1,c2,c3,...,cn]. Equivalently, tile cover of the rectangle is the linear sum of lagrangian squares (coins are 2-dimensional lagrangian tiles themselves) and is a 2-dimensional Money Changing Problem which is again NP-Hard. Product of the equations is a Quadratic Program which is also NP-Hard. Solution vector [c1,c2,c3,...,cn] is the set of number of tiles for each lagrangian tile square.MCP solves the tile arrangement in exact non-deterministic polynomial time while LSMR least squares is a deterministic polynomial time approximation. An example greedy algorithm for minimum coins change problem is described in GRAFIT course material repository: https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt

9. Lagrangian Four Square reduction of a set partition to sum of squares can be equated to Pell Equation written in terms of factors of N=pq:

```
      integer or set partition of N = pq = n1^2 + n2^2 + n3^2 + ... + nk^2 = x^2
```

- y^2 = for x=(p+q)/2 and y=(p-q/2)

which in effect shatters or demolishes an one-dimensional set to a set of k 2-dimensional square tiles and equates the sum of squares to difference of two squares.

10. Shell Turing Machines defined in this draft can be formalized by Category Theory: Category S has various topological spaces of arbitrary dimensions as objects and morphisms are Conduit Turing Machines amongst these object spaces which act as linear transformations between one space to the other. Lagrangian set partition to tile cover reduction is an instance of such a Category of Topological Spaces (TOP) - For objects S1 and S2 in S, set partition defined in an one dimensional space S1 is lifted to a 2 dimensional space of square tiles S2 by Lagrange Four Square Conduit Turing Machine Morphism between object spaces S1 and S2. Application of Shell Turing Machines to previous reduction is unusual (NeuronRain Documentation,FAQ and Licensing: http://neuronrain-

documentation.readthedocs.io/en/latest/ mentions some references on TOP). Thus Shell Turing Machines (Category of Topological Spaces) perform a kind of kernel lifting - Lagrangian Tiling Conduit Turing Machines are the kernels

11. Nomenclature "Shell Turing Machines" is thus a universal mathematical abstraction which connects vector space embedding of alphabets and states transitions of Turing Machine and transformation of truth values of logical statements between spaces in many fields of STEM:

11.1 Hilbert Space Quantum Computation Machines - States and Alphabets of Quantum Turing Machine are embedded in a Hilbert Space (Section 624)

11.2 Linear Machines of Category theory - [Eilenberg-Maclane] - Categories (Objects,Morphisms,Functors) are embedded in a vector space (Conceptual graph of NeuronRain theory drafts - http://neuronrain-documentation.readthedocs.io/en/latest/)

11.3 Kernel Lifting in Support Vector Machines - dataset in smaller dimensional space S1 is lifted to a higher dimensional space S2 by a kernel (e.g Mercer polynomial kernel - https://github.com/shrinivaasanka/Grafit/blob/master/course_material/ComputerScienceMiscellaneous/ComputerScienceMiscellaneous_CourseNotes.txt) and decision hyperplane classifier is found in S2

11.4 Set Partition to Lagrange Tile Cover reduction - lifts an one dimensional set partition to a 2 dimensional square tile cover of a rectangle by Lagrange Four Square Theorem (sides of rectangle are found by Computational Geometric Factorization)

11.5 Word Embedding of datasets in Machine Learning

11.6 Reproducing Kernel Hilbert Space (RKHS) - functions (Turing machines) are embedded on a metric space and distance is defined between functions (Section 624).

11.7 Category of Topological Spaces - TOP - abstracts all the previous embeddings and lifting by defining topological spaces (e.g datasets embedded on vector space) as objects of category and morphisms amongst them. Morphisms between topological spaces are Conduit Turing Machines formalized by any of the previous computation models which lift data and truth values of logical statements between spaces - Linear Machines, RKHS, linear transformations, kernels, Lagranges four squares theorem, Word embedding. Examples:

11.7.1 TOP category T has two space objects S1 and S2. Conduit Turing Machine morphisms are defined between S1 and S2. Logical statement Biggest() which finds maximum of points in a toplogical space thus has two truth values in S1 and S2 - Biggest(S1) and Biggest(S2) - which might have to be teleported between spaces.

11.7.2 Another Good software engineering example of such lifting is the Unix Shell Category where object shells are arbitrarily created and environment variables are privy to a shell object unless otherwise communicated explicitly by EXPORT which acts as a Conduit Turing Machine Morphism between object shells. Outermost shell is the Root and a tree hierarchy of recursively created subshells forms a nested category of shell objects - S1 and S2 are two shell objects in this Shell Category theoretically embedded in some topological space, and EXPORT morphism EXPORT: S1 <=> S2 is a Conduit Turing Machine which exports environment variables of S1 embedded in space X and S2 embedded in space Y bijectively between S1 and S2 except Outermost shell which is unidirectional (because nothing needs to be lifted out of outermost Shell but only to subshells). Cutting across operating systems, such a Shell mechanism and EXPORT morphism has been universally implemented not just limited to Unix - any computation in STEM (Science-Technology-Engineering-Mathematics) is on some OS which has shell TOP category implementation.

11.7.3 Shell Turing Machines Category of Topological Spaces can be formulated as a space filling problem - Category of hierarchically nested topological space shells are embedded in and cover the outermost n-dimensional Shell topological space (Outermost n-sphere) and EXPORT morphisms are defined amongst these shell spaces. For 3-dimensions, this is akin to filling an Outermost spherical bubble container shell by smaller nested spherical shell bubbles which is a 3-dimensional variant of Apollonian Gasket.

11.8 Turing Degree is the equivalence class of Turing machines solving sets of similar difficulty. Turing Jump A' of A is the set of Turing machines which halt having oracle access to A or set of problems harder than A. Previous

Conduit Turing Machine morphisms in TOP abstraction define a Turing Jump between topological spaces of different Turing degree difficulties e.g Conduit Turing Machine Morphism might have an oracle access to a Turing machine defined in a domain topological space of lower Turing Degree lifting to a range topological space of higher Turing Degree. Every computation in STEM can be abstracted as Category of Topological Spaces and Turing Degree Hiearchy can be defined on them e.g Computational Physics problems in Relativity, Quantum mechanics and String theory involving theoretical abstractions for microcosm and macrocosm.

References:

728.1 Coin Problem, Frobenius Problem, Postage Stamp Problem - What This Country Needs is an 18c Piece - https://cs.uwaterloo.ca/~shallit/Papers/change2.pdf

---
729. (THEORY and FEATURE) Drone Electronic Voting Machine - Streaming Boyer-Moore Majority voting - 27 November 2019

1.This commit updates already existing streaming majority implementation of Boyer-Moore algorithm to accept arbitrary datasources (majority_voting() in Streaming_BoyerMoore_MajorityVoting.py)
2.Drone electronic voting machine JSON persistence datastorage written by Streaming SetPartitionAnalytics electronic_voting_analytics() function (Streaming_SetPartitionAnalytics.py) is passed on to Boyer-Moore function as DictionaryHistogramPartition datasource.
3.Persisted EVMs are looped through and majority candidate bucket per EVM is printed.

---
730. (THEORY and FEATURE) Drone Electronic Voting Machine - Streaming Voting Analytics - Bertrand Ballot Theorem - Approximation of Majority Voting, Streaming majority, Forecasts, Theoretical EVMs, Set Partitions - related to 666 - 28 November 2019

1.This commit implements the Betrand Ballot Theorem - probability p-q/p+q that one of the candidate might win of two rival candidates getting p and q votes respectively during the process of counting - as a live election analytics measure within electronic_voting_machine() function of Streaming_SetPartitionAnalytics.py
2.It assumes live counting of voting - votes are immediately counted after they are cast - is allowed as against deferred counting convention usually followed in elections because secrecy might be compromised.
3.But this measure is also useful for post-poll analytics during the process of counting itself for extrapolating the result - before election is called.
4.Random Majority voting is simulated on 3 fictitious candidates and probabilistic trend is printed while voting is underway.
5.logs are at testlogs/Streaming_SetPartitionAnalytics.EVM.log.28November2019
6.This measure applies only to binary election of 2 candidates - EVM dictionary is sorted and votes of top 2 rivals are compared dynamically (after each vote is cast), to print the Bertrand probability.

---
731. (FEATURE) Astronomical Pattern Mining - Rule Search Script update for Maitreya Swiss Ephemeris text client maitreya8t - 4 December 2019

1.python-src/MaitreyaEncHoro_RuleSearch.py has been changed for updated chart summary table of maitreya swiss ephemeris text client which
truncates the names of signs and planets to first 3 alphabets.

2.New function prune_rule() has been defined to prune the class association rules to be matched from python-src/MinedClassAssociationRules.txt(which contains already learnt rules from SequenceMining.py for various weather datasets)
3.SearchOption=2 has been updated for redefined signs-planets defaultdict to populate truncated names.
4.Encoded chart concatenation has been changed for lookup of truncated names
5.Maitreya Text Client commandline has been changed to 64-bit deb package install /usr/bin/maitreya8t - text client is not built from Maitreya Dreams source.
6.An example celestial 2-body pattern "Mercury,Jupiter" is searched on 26/12/2019 when a rare 7 planet conjunction of Sun,Moon,Mercury,Jupiter,Saturn,Nodal Points,Pluto (8 planets including nodal axis) occurs. This 7+1 body planetary system in Sagittarius-Gemini is significant in the context of heightened correlation of weather events that might befall (seismic,oceanic and atmospheric).
7.logs for class association rule match are in python-src/testlogs/MaitreyaEncHoro_RuleSearch.maitreya8t.log.4December2019

---
732. (FEATURE and THEORY) Merit of Large Scale Visuals - Tensor Rank intrinsic merit of EventNet of a video - 7 December 2019 - related to 588

1.This commit upgrades ImageGraph_Keras_Theano.py to Python 3.6 from Python 2.7 and computes Tensor Rank of EventNet of a video by tensorly python library (tensorly requires Python 3) - autopep8 and 2to3 tools have been used for upgrade. Python 3.6 is many orders of magnitude faster than Python 2.7 and EventNet Tensor Product and Tensor Rank is computationally intensive which needs faster Python.
2.Following dependent files have been upgraded to Python 3.6 by two tools - autopep8 and 2to3 - which correct formatting issues to comply with PEP8 and convert a file written in python 2.7 to python 3.6 (Python 2.7 versions have been committed having .2.7 suffix):
    2.1 RecursiveGlossOverlap_Classifier.py
    2.2 WordNetPath.py
    2.3 ImageGraph_Keras_Theano.py
3.Note: all python 2.7 source files in NeuronRain can be upgraded to Python 3.6 in two simple steps for better performance:
    3.1 Execute autopep8 on the file for PEP8
    3.2 Execute 2to3-2.7 on PEP8 formatted file
4.ImageGraph_Keras_Theano.py - videograph_eventnet_tensor_product() has been modified for preprocessing (slicing 100*100 square) the tensor product of video EventNet and tensorly non_negative_parafac() is invoked which decomposes the video EventNet tensor.
5.logs have been committed to testlogs/ImageGraph_Keras_Theano.log.TensorRank1Decomposition.7December2019
6.Video EventNet Tensor is printed for debug purposes
7.Rank-1 tensor factors from non_negative_parafac() of Video EventNet Tensor are also printed

---
733. (FEATURE and THEORY) Computational Geometric Factorization - Quadcore Python 3.6 Upgrade Benchmarks - 11 December 2019 - related to all sections of Factorization
---
1.This commit upgrades Spark Python Computational Geometric Factorization implementation DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py from Python 2.7 to Python 3.6 for performance and availing the better range() utility in Python 3.x which breaches the maximum range barrier limitation in Python 2.7. Python 3.6 range() is a rewrite and supports huge integers (of

hundreds of digits).
2.Following Spark Python environment variables have been set to Python 3.6m binary for PySpark upgrade to Python 3.6:
      SPARK_PYTHON=/usr/bin/python3.6m
      SPARK_DRIVER_PYTHON=/usr/bin/python3.6m
3.Large Integer 999999999999999999 (60 bits) has been factorized which is by far the largest integer factorized in Spark BSP-NC-PRAM model. Larger integers throw a Spark Python error in PySpark context.py (File "spark-2.4.3-bin-hadoop2.7/python/lib/pyspark.zip/pyspark/context.py", line 501, in parallelize, OverflowError: Python int too large to convert to C ssize_t):
      if isinstance(c, xrange):
            size = len(c)
because of len() failure for larger integers
[
Error can also be produced from commandline:
>>> x=range(9999999999999999999)
>>> len(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: Python int too large to convert to C ssize_t
>>> x=range(999999999999999999)
>>> len(x)
999999999999999999
]
4.Previous error requires Spark rewrite of context.py parallelize() to:
      range().stop - range().start
replacing len() specific to Python 3.x range()
5.Python 2.7 Spark Factorization implementation has been upgraded to Python 3.6 by autopep8 followed by 2to3-2.7 utilities. Python 2.7 version has been marked as DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py.2.7
6.Spark Python version is printed by sys.version:
      Spark Python version: 3.6.8 (default, Oct 7 2019, 12:59:55)
      [GCC 8.3.0]
7.First few factors of 999999999999999999 are printed to testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.Python3.6Upgrade.11December2019. Timestamp of each factor is printed.
8.Ray shooting queries excluding Hardy-Ramanujan and some imports have been commented.

---
734. (FEATURE) Merit of Large Scale Visuals - Video EventNet Tensor Decomposition to Rank-1 Tensors on Python 3.7.5 - slicing - 12 December 2019

1.ImageNet_Keras_Theano.py videograph_eventnet_tensor_product() has been changed to accept side of square slice per frame as an argument which is by default 100*100 pixels.
2.Square slicing per frame is necessary when elements of video tensor product matrix are of unequal rectangular dimensions and tensorly stipulates equal dimensions.
3.Usual videos have frames of equal dimensions but the example montage video footage in NeuronRain has unequal frame sizes and minimum frame sides have to be found which are then used to slice each frame.
4.Since a video can have arbitrarily large number of frames in millions, finding the minimum sides of the frame rectangles could be daunting.
5.Instead of finding minimum frame sides, square slicing has been made default and configurable by user in function argument "squareslice" which is by default 100.
6.Python 3.6.8 for TensorLy has been upgraded to Python 3.7.5 which is the fastest python distribution.

---

735. (THEORY and FEATURE) Complement Diophantines - Least Squares Vandermonde Polynomial Fit Learning - related to all sections on Complement Diophantines - Python 3.7.5 upgrade - 14 December 2019

---

1. complement.py complement diophantine learner has been updated to learn a degree 5 interpolated polynomial for a complementary set (non-squares)
2. polyfit() from numeric python has been invoked on a complementary set of non-squares (https://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html) which is a least squares solution solver for Vandermonde coefficient matrix
3. polyfit() least square interpolation is an alternative to Lagrangian polynomial interpolation (https://en.wikipedia.org/wiki/Lagrange_polynomial) mentioned in arXiv version (https://arxiv.org/abs/1106.4102v1, https://scholar.google.com/citations?user=eLZY7CIAAAAJ&hl=en).
4. complement.py has been upgraded to Python 3.7.5 from Python 2.7 by autopep8 and 2to3 utilities. Python 2.7 version has been suffixed as .2.7
5. polyfit() least square interpolation is guaranteed to be unique - https://en.wikipedia.org/wiki/Polynomial_interpolation
6. logs for polyfit() are committed to testlogs/complement.PolyFitPython3.7.5.14December2019
7. Least Squares and Lagrangian interpolations are total functions while Lagrange Four Square Theorem complement map is a partial function (not all domain tuples are mapped to a point in complementary set)

---

736. (THEORY and FEATURE) Computational Geometric PRAM-NC-BSP Factorization - Python 3.7.5 upgrade - Spark Quadcore benchmarks - 512 bit integer - related to 477,481 and all other sections on Factorization - 18 December 2019

1.Computational Geometric Factorization Python-Spark implementation has been upgraded to Python 3.7.5 and following local tile search algorithm mentioned in 477 and 481 has been implemented which obviates parallel sorting and completely removes the maximum integer barrier altogether:

```
while(iterations <= O((logN)^k))
{
      *) Assign N/(logN)^k tiles to N/(logN)^k PRAMs in parallel (O(1) parallel time because each interval tile in a global shared memory array can be accessed by PRAM id as index)
      *) Binary Search tile in each PRAM for factors (O(logN) parallel time which can be reduced to O(logN/logp) by parallel binary search from Snir's Theorem)
}
```

2.Number of PRAMs (multicores) is 4 and constant k in (logN)^k has been set to 50 which, though not commensurate, is hardcoded for demonstrative reasons. Previous algorithm is work optimal NC because PRAM time * iterations = Sequential time
3.Spark context parallelize() has been refactored to following loop which circumvents the Spark context ssize_t exception in pyspark context.py parallelize() - maximum value of len() is limited to 64 bits per slice by suitable choice of exponent k:

```
        normal_order_n = int(math.pow(math.log(n, 2),50))
        tiles_start = 1
        tiles_end = int(n/normal_order_n)
        for x in range(normal_order_n):
            print("tiles_start:", tiles_start)
            print("tiles_end:", tiles_end)
            tiles = range(tiles_start, tiles_end)
            print(("len(tiles):", len(tiles)))
            spcon.parallelize(tiles).foreach(
```

```
                tilesearch_nonpersistent)
            tiles_start = tiles_end
            tiles_end += int(n/normal_order_n)
```
Previous loop of (logN)^k iterations slices the Python 3.7.5 range() into multiples of N/(logN)^k and individually parallelizes each slice to RDDs.
4.Because Python 3.7.5 range() is unlimited, maximum number factorizable is arbitrary for all practical purposes - previous benchmark for 60 bits has been bettered and a huge integer of 512 bits which is ecommerce-grade - 99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999 - has been factorized to print all factors on single node cluster of 4 multicores. This is the largest integer factorized thus far. Smoothness of this integer (high omega and large number of small prime factors) causes easy factorization.
5.logs at
testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.Python3.
7.5Upgrade.512bits.18December2019 print a truncated list of factors of
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999 with
in few seconds:
```
=================================================
('Factor is = ', 3, '(at ', 'Wed, 18 Dec 2019 05:28:26 GMT', ')')
=================================================
=================================================
('Factor is = ', 9, '(at ', 'Wed, 18 Dec 2019 05:28:26 GMT', ')')
=================================================
=================================================
('Factor is = ', 11, '(at ', 'Wed, 18 Dec 2019 05:28:26 GMT', ')')
=================================================
=================================================
('Factor is = ', 23, '(at ', 'Wed, 18 Dec 2019 05:28:26 GMT', ')')
=================================================
=================================================
('Factor is = ', 33, '(at ', 'Wed, 18 Dec 2019 05:28:26 GMT', ')')
=================================================
=================================================
('Factor is = ', 69, '(at ', 'Wed, 18 Dec 2019 05:28:26 GMT', ')')
=================================================
=================================================
('Factor is = ', 207, '(at ', 'Wed, 18 Dec 2019 05:28:27 GMT', ')')
=================================================
=================================================
('Factor is = ', 239, '(at ', 'Wed, 18 Dec 2019 05:28:27 GMT', ')')
=================================================
=================================================
('Factor is = ', 253, '(at ', 'Wed, 18 Dec 2019 05:28:27 GMT', ')')
=================================================
=================================================
('Factor is = ', 463, '(at ', 'Wed, 18 Dec 2019 05:28:28 GMT', ')')
=================================================
=================================================
('Factor is = ', 759, '(at ', 'Wed, 18 Dec 2019 05:28:28 GMT', ')')
=================================================
=================================================
('Factor is = ', 2277, '(at ', 'Wed, 18 Dec 2019 05:28:32 GMT', ')')
=================================================
=================================================
('Factor is = ', 4167, '(at ', 'Wed, 18 Dec 2019 05:28:37 GMT', ')')
=================================================
=================================================
('Factor is = ', 4649, '(at ', 'Wed, 18 Dec 2019 05:28:39 GMT', ')')
=================================================
=================================================
('Factor is = ', 5237, '(at ', 'Wed, 18 Dec 2019 05:28:40 GMT', ')')
=================================================
```

```
================================================
('Factor is = ', 5497, '(at ', 'Wed, 18 Dec 2019 05:28:41 GMT', ')')
================================================
================================================
('Factor is = ', 7887, '(at ', 'Wed, 18 Dec 2019 05:28:47 GMT', ')')
================================================
================================================
('Factor is = ', 8349, '(at ', 'Wed, 18 Dec 2019 05:28:48 GMT', ')')
================================================
================================================
('Factor is = ', 13947, '(at ', 'Wed, 18 Dec 2019 05:29:02 GMT', ')')
================================================
================================================
('Factor is = ', 16491, '(at ', 'Wed, 18 Dec 2019 05:29:09 GMT', ')')
================================================
================================================
('Factor is = ', 21649, '(at ', 'Wed, 18 Dec 2019 05:29:22 GMT', ')')
================================================
================================================
('Factor is = ', 23661, '(at ', 'Wed, 18 Dec 2019 05:29:28 GMT', ')')
================================================
[...truncated...]

--------------------------------------------------------------------------------
------------------------------------------------------------
737. (THEORY and FEATURE) Spark NC-PRAM-BSP Computational Geometric
Factorization quadcore Python 3.7.5 benchmarks - 1000+ bits non-smooth random
integer of mixed digits - 22 December 2019 - related to all sections on
Computational Geometric Factorization
--------------------------------------------------------------------------------
------------------------------------------------------------
1.DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been
changed to increase k to 100 which factorizes a difficult 1000 bits integer of
mixed digits (low omega and less number of prime factors) which is by far the
largest integer factorized in NeuronRain -
99320930203909302903909209302309029309029039203819821892891217288172871728718728
17281728718728172871872817872817318361763793902930911288218943488989389829311298
29189987676767675656565454553437878787878767675665656454545344343338787387287382
78372243439848343884384983984989483849839483984983984983984977 - in
O((logN)^100) Parallel RAM time requiring O(N/(logN)^100) PRAM processors -
first few factors are printed below within 20 seconds and 73 seconds
respectively which includes Spark-Python-Py4j-Java 8 bidirectional marshalling
and VM bytecode interpretation overhead (Cythonized C++ DLL version might be
still faster):

spark-submit DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py
99320930203909302903909209302309029309029039203819821892891217288172871728718728
17281728718728172871872817872817318361763793902930911288218943488989389829311298
29189987676767675656565454553437878787878767675665656454545344343338787387287382
78372243439848343884384983984989483849839483984983984983984977
19/12/22 13:10:08 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
19/12/22 13:10:09 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
('Spark Python version:', '3.7.5 (default, Nov  7 2019, 10:50:52) \n[GCC
8.3.0]')
('factors of ',
99320930203909302903909209302309029309029039203819821892891217288172871728718728
17281728718728172871872817872817318361763793902930911288218943488989389829311298
29189987676767675656565454553437878787878767675665656454545344343338787387287382
78372243439848343884384983984989483849839483984983984983984977, '(',
1003.212454334434, ' bits integer) are:')
================================================
('Factor is = ', 249, '(at ', 'Sun, 22 Dec 2019 07:40:29 GMT', ')')
```

```
================================================
================================================
('Factor is = ', 4731, '(at ', 'Sun, 22 Dec 2019 07:41:22 GMT', ')')
================================================
[...truncated...]
```

2.Previous complexity bounds imply O(1000^100) parallel time units (polylog depth) and O(2^1000/1000^100) processors ~= O(2^1000/2^997) = O(8) PRAMS (multicores) which is a meagre NC parallelism required for factorizing such a huge number - seemingly prohibitive O(N/(logN)^k)) PRAMs in definition of NC might not be a roadblock after all.
3.Recursion limit has been increased to 3000 because of stack overflow in recursive binary search and ray shooting queries have been commented.
4.Spark factorization logs (truncated) have been uploaded at testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.Python3.7.5Upgrade.1000bits.22December2019
5.Change in the constant k depending on the input simulates non-uniform NC circuit for factorization theoretically (family of circuits depending on input)

References:

737.6.Input Size and PRAMs - n and N defined - Geomblog - [Suresh Venkatasubramanian] - http://blog.geomblog.org/2008/05/p-vs-nc-part-i-preliminaries.html:
"...It will be important to pay attention to input size. As always, the input size can either be described by n, the cardinality of the input (the quantity counting the number of "things"), or by N, the bitsize (the quantity measuring the number of bits needed to write everything down). When we talk about polylogarithmic time and polynomial number of processors, unless specified otherwise, we will be referring to polynomials of n AND N. In the non-uniform setting, we'll assume that we're presented with a PRAM that for fixed n, N, runs in time t(n, N) with p(n, N) processors...." - In Computational Geometric Factorization NC definition, O((logX)^k) is a polynomial t(N) and O(X/(logX)^k) PRAMs is a polynomial p(n,N) where N=logX and n=X.
737.7 Definition of PRAM - Definition 2.2.1 - [RAYMOND GREENLAW,H. JAMES HOOVER,WALTER L. RUZZO - OXFORD UNIVERSITY PRESS 1995] - https://homes.cs.washington.edu/~ruzzo/papers/limits.pdf
737.8 Poster: Easy PRAM-based High-performance Parallel Programming with ICE - [Ghanim-Rajeev-Vishkin] - https://www.researchgate.net/publication/310824244_POSTER_Easy_PRAM-based_High-Performance_Parallel_Programming_with_ICE - input size to Pointer jumping algorithm

---

738. (THEORY and FEATURE) Computational Geometric NC-PRAM-BSP Factorization - Python 3.7.5 Single node cluster quadcore benchmarks - 1067 bits integer - some factcheck references on NC-PRAM equivalence and input size - 24 December 2019,31 December 2019,2 January 2020 - related to all sections on Factorization

---

1.In continuation of previous benchmarks on 1000 bits integer another hard non-smooth random 1067 bits integer of randomly mixed digits - 939483849839493438483989489384983948938843984983984983984938948348938499384928830928380289389283982983928398249748378478378738478738748729839289389289382983982939283982983928398249748378478378738478738748729839289389289382983982939283982983928398249748378478378738478738748729839289389289382983982939283982983928398249748378478378738478134938483349938497 9 - has been factorized on single node cluster of quadcores and first few factors - mostly primes - are printed from 18th second onwards:

```
================================================
('Factor is = ', 11, '(at ', 'Tue, 24 Dec 2019 08:21:47 GMT', ')')
================================================
================================================
('Factor is = ', 17, '(at ', 'Tue, 24 Dec 2019 08:21:47 GMT', ')')
```

```
================================================
================================================
('Factor is = ', 47, '(at ', 'Tue, 24 Dec 2019 08:21:49 GMT', ')')
================================================
================================================
('Factor is = ', 51, '(at ', 'Tue, 24 Dec 2019 08:21:49 GMT', ')')
================================================
================================================
('Factor is = ', 141, '(at ', 'Tue, 24 Dec 2019 08:21:53 GMT', ')')
================================================
================================================
('Factor is = ', 187, '(at ', 'Tue, 24 Dec 2019 08:21:55 GMT', ')')
================================================
================================================
('Factor is = ', 289, '(at ', 'Tue, 24 Dec 2019 08:22:00 GMT', ')')
================================================
================================================
('Factor is = ', 311, '(at ', 'Tue, 24 Dec 2019 08:22:01 GMT', ')')
================================================
================================================
('Factor is = ', 517, '(at ', 'Tue, 24 Dec 2019 08:22:11 GMT', ')')
================================================
================================================
('Factor is = ', 561, '(at ', 'Tue, 24 Dec 2019 08:22:13 GMT', ')')
================================================
================================================
('Factor is = ', 799, '(at ', 'Tue, 24 Dec 2019 08:22:24 GMT', ')')
================================================
================================================
('Factor is = ', 867, '(at ', 'Tue, 24 Dec 2019 08:22:27 GMT', ')')
================================================
================================================
('Factor is = ', 3179, '(at ', 'Tue, 24 Dec 2019 08:24:16 GMT', ')')
================================================
================================================
('Factor is = ', 3421, '(at ', 'Tue, 24 Dec 2019 08:24:27 GMT', ')')
================================================
================================================
('Factor is = ', 5287, '(at ', 'Tue, 24 Dec 2019 08:25:54 GMT', ')')
================================================
================================================
('Factor is = ', 8789, '(at ', 'Tue, 24 Dec 2019 08:28:37 GMT', ')')
================================================
================================================
('Factor is = ', 9537, '(at ', 'Tue, 24 Dec 2019 08:29:12 GMT', ')')
================================================
================================================
('Factor is = ', 10263, '(at ', 'Tue, 24 Dec 2019 08:29:46 GMT', ')')
================================================
================================================
('Factor is = ', 13583, '(at ', 'Tue, 24 Dec 2019 08:32:20 GMT', ')')
================================================
[....truncated....]
2.float() cast has been changed to Decimal() for highest precision and remove
overflow errors for tile segments and interval midpoint computations
3.Recursion limit has been increased to 10000
4.Exponentiation by math.pow() for normal order computation in local search
optimization has been replaced by faster ** exponentiation operator of Python.
5.Exponent k in local search loop - (logN)^k (iterations) * N/(logN)^k (PRAMs) -
has been parametrized in SearchTiles_and_Factorize() function which simulates
non-uniform NC circuit of (logN)^k depth and N/(logN)^k circuit elements.

Optimization:
-------------
```

Finding all factors by searching through every rasterized tile segment for hyperbola xy=N could be optimized by stopping after first factor p is found and re-rasterizing a new hyperbolic arc for xy=N/p. Following loop formalizes it (reduction in cost might depend on which is easier - parallel planar point location or rasterization - this optimization creates an array of hyperbolic arc bows equal to number of factors):

```
      for_all_factors_of_N
      {
            Rasterize hyperbola xy=N
            PRAM-BSP-NC Planar Point locate and Search the rasterized hyperbolic planar straight line graph till first factor p is found
            N=N/p
      }
```

Previous loop is of approximate total parallel RAM time (excluding parallel rasterization which is increasingly easier after each iteration churning out a factor):

O((log p1)^k) + O((log N/p1)^k) + O((log N/[p1*p2])^k) + ... + O((log N/[p1*p2*...pn])^k) for first n factors of N={p1,p2,...,pn}

References:

738.6 Sorting,Searching,Merging - [Joseph JaJa] - https://people.ksp.sk/~ppershing/data/skola/JaJa.pdf - Chapter 4 - Pages 159-160 - Algorithm 4.3 - Simple Merge Sort - " ... input: an array X of order n where n=2^l ...", Corollary 4.2 - CREW PRAM model - "... Sorting a sequence of n elements can optimally be done in O(logn*loglogn) ... " - previous local search does away with parallel sorting

738.7 PARALLEL TRANSITIVE CLOSURE AND POINT LOCATION IN PLANAR STRUCTURES - [ROBERTO TAMASSIA AND JEFFREY S. VITTER] - http://www.ittc.ku.edu/~jsv/Papers/TaV91.transclosure.pdf - "... Abstract. Parallel algorithms for several graph and geometric problems are presented, including transitive closure and topological sorting in planar st-graphs, preprocessing planar subdivisions for point location queries, and construction of visibility representations and drawings of planar graphs. Most of these algorithms achieve optimal O(log n) running time using n/log n processors in the EREW PRAM model, n being the number of vertices...." - number of processors required by this planar point location algorithm is poly(n,logn) and time is poly(logn) by reference 737.6 and ANSV problem for mergesort in reference 34.15 - all PRAM models are mutually reducible to other - previous local search planar point location factorization requires O((logN)^k) parallel time and N/(logN)^k PRAM processors and thus work optimal - number of vertices in rasterized hyperbolic arc bow planar simple line graph is approximately 3*N + 1 if every juxtaposed tile segment is construed as a rectangular polygon face of dimensions 1 * segment_length; polygon arithmetic progression segment located in polylog time is binary searched.

738.8 An accurate algorithm for rasterizing algebraic curves - [Gabriel Taubin - IBM Watson Research Centre] - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.72.9588&rep=rep1&type=pdf - Rasterization defined - "... By rasterizing an algebraic curve we mean to determine which cells, or pixels, from a square mesh of cells in the plane, are cut by a curve represented as the set of zeros of a polynomial in two variables. By using a recursive space subdivision scheme, the problem is be reduced to testing whether the curve cuts a square or not ..." - rasterizing hyperbola which is polynomial in two variables thus is the problem of collating the pixel squares on 2-D grid cut by hyperbola to axis parallel pixel-array polygons.

738.9 Multidimensional planar point location and rasterized point location - https://pure.tue.nl/ws/portalfiles/portal/4238920/373169.pdf - [Two and Three-Dimensional Point Location in Rectangular Subdivisions - Mark de Berg, Marc van Kreveld, Jack Snoeyink] - "... We give an new type of stratified tree that emphasizes a tradeoff between space and query time. In two dimensions, we can preserve Muller's o((log log U)^2) query time using only O(n log log U) space or reduce space to linear and increase the query time to O((logU)<l/h» for any constant h. ..."

738.10 Bulk Synchronous Parallel (BSP) and Coarse Grained Multicomputer Planar Point Location - [Frank Dehne, Wolfgang Dittrich, David Hutchinson, and Anil Maheshwari] - http://www.sce.carleton.ca/~hutchins/papers/DDHM02_Bulk.pdf - "... . The PRAM assumption of a large shared random access memory with uniform cost access to every cell by each processor has the advantage of simplicity but has also prompted alternative proposals intended to be more representative of practical machines. These include the Bulk Synchronous Parallel (BSP) model of Valiant [45], the Coarse-Grained Multicomputer (CGM) of Dehne et al. [22], the LogP model of Culler et al. [19], and the Extended BSP (BSP*) model of Baumker et al. [10]...."
738.11 Rasterized point location - H. Müller. Rasterized point location. In Proceedings Workshop on Graph theoretic Concepts in Computer Science, 1985 pages 281–293. Trauner Verlag, Linz, Austria
738.12 Entropy based Planar Point Location Query Complexity - [Sunil Arya,Malamatos,Mount] - https://www.cs.umd.edu/~mount/Papers/ptloc-rand.pdf
738.13 Fast Queries in Planar Point Location - [Goodrich-Orletsky-RamaIyer] - http://www.cs.jhu.edu/~goodrich/cgc/pubs/adaptive.ps.gz
738.14 Rasterization in Graphics - approximating curves by line segments - Bresenham rasterization - illustrations - http://www.cs.cornell.edu/courses/cs4620/2013fa/lectures/09rasterization.pdf
738.15 Dynamic Parallel Planar Point Location and extensions to higher dimensions - [Ketan Mulmuley-Sandeep Sen] - https://link.springer.com/content/pdf/10.1007/BF02293052.pdf - Theorem 7 - "...The dynamic point-location data structure for an arrangement of lines can be updated in 0(log n) time on a CRCW PRAM model using n processors. The query time is O(log n) and the space complexity is O(n^2). ..." - Drafts on Computational Geometric factorization by planar point location discuss only static parallel planar point location which assume hyperbola has been fully rasterized before location - rasterization creates an axis parellel line arrangement. Optimization could be to lazy-rasterize hyperbolic arc segments on demand and dynamically query the factor points on CRCW PRAM.
738.16 Zone Theorem and Ray shooting queries for approximate factors intersecting rasterized hyperbolic segment arrangement - http://homepages.math.uic.edu/~jan/mcs481/zonetheorem.pdf - Set of faces in an arrangement of lines intersecting a query ray is a Zone of ray query. Zone complexity is the sum of number of vertices,edges and faces in zone of a query ray. Zone theorem states that Zone complexity of an arrangement of m lines and a query ray is O(m). Hyperbolic rasterization produces an arrangement of O(N) lines. But yet because of being axis parallel, every ray shooting query from origin (0,0) intersects only 3 faces of the rasterized arrangement at most (above, pixel array rectangle, below) and thus zone complexity of ray shooting is a constant 3.
738.17 Approximate Factors and Local search for exact factor point location - https://en.wikipedia.org/wiki/Local_search_(optimization) - Approximate factors found by number theoretic results are in close proximity to exact factors. Local search optimization finds exact solution to an optimization problem from candidate solutions and iteratively refines. Finding exact factors from approximate factors could be formulated as local search problem e.g Approximate factor ax for an integer N could be refined by incrementing (decrementing) along x-axis - a kind of hill-climbing on hyperbolic arc. Local search algorithms could have arbitrary, user-defined runtimes.

---
739. (THEORY and FEATURE) Intrinsic Merit - Unique Identification and its necessity in People Analytics (Drone Electronic Voting Machines), Online Citizen Science, Sports Analytics and Academic Rankings, BKS Conjecture, Consensus versus Majority - 29 December 2019,2 January 2020 - related to 730

(*) Drone Electronic Voting Machine on GPS navigation to a voter's residence optionally can have an one time password multifactor authentication before voting. Traditional OTPs are electronic prone to sabotage. Alternatively a non-digital OTP authentication is envisaged e.g serial number of a currency bill

(only one voter in nation at any point in time can possess a note having a unique serial number if there are negligible counterfeits).Drone EVM reads the unique serial number in note by OCR before voting. Encrypted vote to a candidate salted with
 encrypted OCR-ed serial number of the currency is appended to voted candidate's bucket. Voter can have multiple non-digital currencies (multiple serial number unique ids) which are chosen at random by voter for OCR. Currencies are anonymous and unpredictably flow in economic networks which further randomizes the non-digital OTPs (high entropy randomness extractor source).
(*) QR codes are unique id(s) - two dimensional quick response bar codes
(*) NeuronRain KingCobra implements a protocol buffer cryptocurrency which is serial numbered by Boost Universally Unique Identifier(based on Perfect Forward copyless move idiom of C++ - has parallels to Cloud Object move in Google Clouds - more on this in http://neuronrain-documentation.readthedocs.io/en/latest/). Exchequer in some countries is RFID tagged for tracking money trail and similar reasoning might apply for voter uniqueness - RFID tagged currency authentication at drone EVM.
(*) Genetic Haplogroups of populace have specific cluster of traits across ethnic groups which might be occasionally necessary to establish uniqueness (e.g India has prominence of R1a1a... haplogroup - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1380230/).
(*) Vandermonde interpolation polynomial is guaranteed to be unique for set of points to be interpolated. This set of points could be strings sourced from multiple identification traits of a person which can be Rabin fingerprinted by a polynomial over GF(2) modulo an irreducible polynomial over GF(2). This set of fingerprinted data points are then interpolated by a Vandermonde unique polynomial to ID a voter.
(*) Sports Analytics and Intrinsic Performance Rankings have been mentioned in GRAFIT course material as a BigData usecase (https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/LinuxKernelAndCloud/BigdataAnalyticsCloud_CourseNotes.txt) - e.g ATP,WTA,NBA,ICC. Academic Rankings (rankings of researchers and universities - USA today,Times Higher Education) could leverage from Sports Rankings e.g Number of experimentally verified papers of an author in relativity and quantum mechanics with no dependency on number of citations is a measure of intrinsic research quality
(*) Evidence usecase for BKS conjecture: Everyone agrees on denomination of a currency bill or coin - $100 is universally accepted consensually as equal to $100 discounting devaluation and other economic phenomena. If this fact is formulated as an LTF or PTF - value(x) = x - could evidence the truth of BKS conjecture implying consensus is stabler than majority function.

---

740. (THEORY and FEATURE) People Analytics - Drone Electronic Voting Machine - Python 3.7.5 upgrade and Paper ballot shuffle simulation, PRAM Breadth First and Depth First Search, Set Partition Analytics, Money Trail Economic EventNet Graph Random Walk, Expander Graphs, Cheeger's Constant, Random Closed Packing, Space Filling, Pseudorandomness Extractors and Generators, Birthday Paradox, Unique Identification, Ramsey coloring, Computational Chaos, Voting analytics - 7 January 2020,12 January 2020,13 January 2020 - related to 22,135,230,338,461,739

(*) Drone Electronic Voting Machine implementation in Streaming_SetPartitionAnalytics.py - electronic_voting_machine() - has been updated to simulate random shuffle of paper ballots in boxes to ensure disorder - votes appended to bucket per candidate are shuffled after each vote - every new vote is randomly inserted into obfuscated encrypted bucket of votes per candidate which simulates a shuffle (shuffle is essentially a permutation group Sn) while an exact shuffle has to enumerate the permutation group (O(n!)) and choose one element - every shuffle of a candidate bucket is also a reduction from space filling and random closed packing
(*) Drone EVM implementation has been upgraded to Python 3.7.5 by autopep8 and 2to3 utilities and 2.7 version has been separately committed with eponymous

suffix - random.randint() has been invoked to get an index to insert a new vote somewhere in the candidate bucket
(*) Traditional incremental counters coalesce votes of multiple voters into one integer which prohibits forensic examination of individual votes if there is a dispute. Votes per candidate bucket thus isolates each anonymous encrypted vote but subverts chronological ordering.
(*) Serial number unique id(s) of paper currencies were envisaged earlier as non-digital one time passwords for authenticating a voter. Rationale for choice of currencies is the unpredictable pseudorandom flow of paper currency unique id(s) in economy which creates a huge EventNet Money Trail Graph (edges are labelled by flow of uniquely ID-ed currencies between buyers and sellers) which is non-digital and thus insulated from electronic sabotage. Money Trail EventNet Graph is a high entropy randomness source and could serve as a psueodrandomness generator and extractor e.g topological sort or random walk of a subgraph of money trail EventNet is a random concatenation of unique id(s)
(*) If Money Trail EventNet Graph is an edge expander, it could be a gadget for Pseudorandom Number Generator. Edge expander graphs are defined with Cheeger's constant for minimum high edge boundary per vertex subset:

minimum(|boundary(S)|/|S|), 1 < |S| < n/2

where boundary(S) of a subset S of vertices of G (V(G)) is:

boundary(S) = { (u,v) | u in S, v in V(G) \ S}

Random walks on an expander graph are conventionally applied to Pseudorandom Generators e.g. Margulis [Mar88] and Lubotzky, Phillips and Sarnak [LPS88]
(*) Locating an encrypted vote in a candidate bucket of votes (if bucket is a linked list) could be done in sublinear time by Parallel RAM Breadth First Search and Depth First Search of the linked list line graph - GRAFIT course material in
https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt mention usefulness of parallel RAM sublinear linked list search in the context of Survival Index Timeout OS Scheduler version of Process IDs Set Partition in OS Kernels
(*) Limitations of unique identification are best explained by birthday paradox - GRAFIT course material in
https://github.com/shrinivaasanka/Grafit/blob/master/course_material/ComputerScienceMiscellaneous/ComputerScienceMiscellaneous_CourseNotes.txt have a Q and A on birthday paradox probability of collision of unique IDs in a population.
(*) An expander graph simulation of Economic Network by Neuro Cryptocurrency in NeuronRain which is Boost UUID, Protocol Buffer, Perfect-Forward and Object Move based, theoretically could source pseudorandom bits.

Paper ballots and Drone EVM - comparison and contrast:

Previous concept implementation of Drone EVM in NeuronRain has separate buckets for candidates which are independently shuffled to simulate paper ballots whereas real world paper ballot votes are sealed and cast in single ballot box bucket without candidate delineation. Instead if Drone EVM is implemented such that all encrypted votes are in a single shuffled list (or) linked list bucket (which is how paper ballots work), reduction is necessary which shatters/classifies the shuffled list/linked list of votes to a set partition of buckets of per-candidate lists/linked lists of votes. Example:

every node of the list/linked list of anonymous votes has the fields <candidate_id>:<encrypted_vote>.

Usually this reduction happens post-poll in paper ballot counts. This sort of exact pseudorandom shuffle simulation of paper ballot box of assorted votes democratizes further and is a non-trivial monochromatic balls-bins problem in Drone EVM - every anonymous vote is colored by candidate index - single list (or) linked list line graph of votes of previous format has to be set/graph-partitioned to candidate buckets of multiple lists/line graphs (or) a multichromatic set has to be partitioned to monochromatic candidate bucket subsets - an O(N) labelled classifier. Sequence of N votes for k candidates is thus Ramsey k-colored and order emerges inevitably by Van Der Waerden Theorem - there exist numbers N,k,r such that if shuffled sequence of N votes are colored

by k candidates, there is a monchromatic arithmetic progression of length atleast r (in other words, r voters who had voted for same candidate are equally spaced out which presents an insight into voting pattern).

Conflict between pseudorandomness and arithmetic progressions in Ramsey coloring of votes - Orderly Disorder:

Previous simulation of pseudorandom paper ballot shuffle in Drone EVM gives rise to a multichromatic sequence of votes colored by candidate indices. But yet monochromatic arithmetic progression order emerges from pseudorandom disorder irrespective of number of colors (candidates) and voting chronology which is a contradiction. This "orderly disorder" oxymoron paradox should apply universally to any pseudorandomly shuffled multichromatic set not just limited to shuffled voting sequences. This limitation implies a prerequisite for indistinguishability of pseudorandom generators (PRG) from true randomness - PRGs should create pseudorandomly shuffled multicolored sequences of Ramsey number R(N,k,r) of high N,k and low r - Sequence of N elements, k-colored having an arithmetic progression of length atleast r - which lessens the probability of emergence of arithmetic progression periodicities. On a related note, Period Three Theorem of Chaos theory (www.its.caltech.edu/~matilde/LiYorke.pdf) implies sequences of periodicity 3 can have larger periodicities.

References:

740.1 Random Walks on Expander Graphs and Spectral Graph Theory - [Spielman] - http://www.cs.yale.edu/homes/spielman/561/2012/lect11-12.pdf - Section 11.5 - "...Our pseudo-random generator will use a random walk on an expander graph whose vertex set is {0, 1}^r. The expander graph will be d-regular, for some constant d. Let A be the adjacency matrix of the graph, and let its eigenvalues be d = μ1 > μ2 ≥ · · · ≥ μn. One property that we will require of the expander graph is that |μi|/d ≤ 1/10, (11.1) for all i. ... such graphs exist with d = 400. Degree 400 seems sort of big, but we will see that it is reasonable for our application...." - Degree 400 Money Trail EventNet could potentially be a high edge expander and random walks on it might be pseudorandom. Money Trail Graph could have Rich and Poor strata (and scalefree because select vertices could have huge neighbours) causing regularity to fluctuate and it suffices to find dense subgraphs of high regularity as candidates for expander graph random walks.
740.2 A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. Combinatorica, 8(3):261–277, 1988.
740.3 G. A. Margulis. Explicit group theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. Problems of Information Transmission, 24(1):39–46, July 1988.
740.4 Hardy-Ramanujan-Rademacher estimation of partition function,Bell numbers of set partitions - [Donald Knuth] - http://www.cs.utsa.edu/~wagner/knuth/fasc3b.pdf
740.5 Generalized Condorcet Jury Theorem for multipartisan election - [Christian List-Robert Goodin] - http://personal.lse.ac.uk/LIST/PDF-files/listgoodin.pdf - Appendix 1 and Proposition 1 - k-option Condorcet Jury Model - "...Let X1, X2, ..., Xk be the random variables whose values are the numbers of first choice votes (out of a total of n votes) cast for x1, x2, ..., xk, respectively. The joint distribution of X1, X2, ..., Xk is a multinomial distribution with the following probability function: P(X1=n1, X2=n2, ..., Xk=nk) = (n! / [n1! n2! ... nk!]) * (p1^n1 * p2^n2 * ... pk^nk) , where ∑i ni = n. ... " - Theoretical k-parts Set Partition Electronic Voting Machine in NeuronRain AstroInfer is exacly k-option plurality Condorcet Jury Model for k candidates multipartisan election and probability of a candidate win is derived as: "... For each i, the probability that xi will win under plurality voting is Pi := P(for all j≠i, Xi > Xj) = P(X1=n1, X2=n2, ..., Xk=nk :<n1, n2, ..., nk>ϵNi) = ∑<n1, n2, ..., nk>ϵNi (n! / n1! n2! ... nk!) * (p1^n1 p2^n2 ... pk^nk), where Ni := {<n1, n2, ..., nk> : (for all j, nj ≥ 0) & (∑jnj=n) & (for all j≠i, ni>nj)} (set of all k-tuples of votes for the k options for which option i is the plurality winner). Moreover, if, for all j≠i, pi>pj, then, for all j≠i, Pi>Pj. ...".

---
741. (THEORY and FEATURE) Computational Geometric NC-PRAM-BSP Factorization - Spark 2.4.3 + Python 3.7.5 benchmarks - 1213 and 1243 bits integers factorized, Hyperbolic Rasterization graphics example, Multidimensional hyperplanar point location, Integer Diophantines, Multiplicative partitions and sundry resolutions - related to all sections on factorization - 9 January 2020

(*) New function hyperbolic_arc_rasterization() has been defined which sequentially rasterizes the hyperbolic arc and creates a graphic image of how hyperbola looks like post-rasterization. A related schematic diagram for rasterization has already been committed to GRAFIT course material (https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/LinuxKernelAndCloud/BigdataAnalyticsCloud_CourseNotes.txt) on Geometric search of a point on hyperplanes separating halfspaces by multidimensional Hyperplanar point location which widens the scope of factorization to arbitrary algebraic curves (e.g diophantines) for integer solutions, not limited to hyperbolae - example: uvwxyz=N is a multiplicative partition problem solvable by rasterizing on Z^6 hyperplane point location - (u,v,w,x,y,z)
(*) Hyperbolic arc rasterized tile segments are plotted by matplotlib and example logs-images for a small integer are committed to testlogs/ - DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.Rasterization.9January2020.png and DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.Python3.7.5Upgrade.RasterizationGraphics.log.9January2020
(*) matplotlib rasterization graphics is enabled by a boolean flag passed on by commandline sys.argv arguments
(*) Depth of the non-uniform NC circuit which was hardcoded earlier has been parametrized as commandline argument
(*) Normal order loop has been changed because of overflow errors for large exponents
(*) Huge 1213 bits non-smooth hard integer of random digits
99999999999928918219821828128918928918281939787878773273872837129818280829189289
17871873871873237827846747545265277362838171291829819829818291892819829819289189
27182781782616271672617267167216271627617626172712676172617621726716271672
67162761726716271626715365365636635157612126716725162561526516256152613389132311
13283829232323343343412321283928938283298329l
has been factorized by a depth 58 NC circuit (PRAM time O((logN)^58) and first few factors are printed as ordered pairs (requires division because of or clause and division has been cast to Decimal() because of overflow errors) in testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.Python3.7.5Upgrade.1213bits_and_1243bits.9January2020 from 51st second onwards:
('factors of ',
99999999999928918219821828128918928918281939787878773273872837129818280829189289
17871873871873237827846747545265277362838171291829819829818291892819829819289189
27182781782616271672617267167216271627617626172712676172617621726716271672
67162761726716271626715365365636635157612126716725162561526516256152613389132311
13283829232323343343412321283928938283298329l, '(', 1212.5037546338863, ' bits integer) are:')
tiles_start: 1
tiles_end: 19622815
==========
('Factors are: (', 3, ',', Decimal('3.33333333330963940660727604E+364'), ') (at ', 'Thu, 09 Jan 2020 12:06:26 GMT', ')')
==========
==========
('Factors are: (', 7, ',', Decimal('1.428571428570413117426026116E+364'), ') (at ', 'Thu, 09 Jan 2020 12:06:26 GMT', ')')
==========
==========
('Factors are: (', 21, ',', Decimal('4.76190476190137705808675372OE+363'), ')

```
(at ', 'Thu, 09 Jan 2020 12:06:27 GMT', ')')
================================================

(*) Another large 1243 bits integer of all 9s (smooth, easy) has been factorized
and first few factors are printed from 18 seconds onwards:
('factors of ',
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999, '(', 1242.4011074878736,
' bits integer) are:')
tiles_start: 1
tiles_end: 488
================================================
('Factors are: (', 3, ',', Decimal('3.33333333333333333333333333E+373'), ') (at
', 'Thu, 09 Jan 2020 12:10:55 GMT', ')')
================================================
================================================
('Factors are: (', 9, ',', Decimal('1.11111111111111111111111111E+373'), ') (at
', 'Thu, 09 Jan 2020 12:10:56 GMT', ')')
================================================
================================================
('Factors are: (', 33, ',', Decimal('3.03030303030303030303030303E+372'), ')
(at ', 'Thu, 09 Jan 2020 12:10:57 GMT', ')')
================================================
================================================
('Factors are: (', 99, ',', Decimal('1.01010101010101010101010101E+372'), ')
(at ', 'Thu, 09 Jan 2020 12:11:02 GMT', ')')
================================================
================================================
('Factors are: (', 103, ',', Decimal('9.708737864077669902912621359E+371'), ')
(at ', 'Thu, 09 Jan 2020 12:11:02 GMT', ')')
================================================
================================================
('Factors are: (', 121, ',', Decimal('8.264462809917355371900826446E+371'), ')
(at ', 'Thu, 09 Jan 2020 12:11:03 GMT', ')')
================================================
================================================
('Factors are: (', 363, ',', Decimal('2.754820936639118457300275482E+371'), ')
(at ', 'Thu, 09 Jan 2020 12:11:20 GMT', ')')
================================================
tiles_start: 488
tiles_end: 976
================================================
('Factors are: (', 759, ',', Decimal('1.317523056653491436100131752E+371'), ')
(at ', 'Thu, 09 Jan 2020 12:11:47 GMT', ')')
================================================
tiles_start: 976
tiles_end: 1464
================================================
('Factors are: (', 1133, ',', Decimal('8.826125330979699911738746690E+370'), ')
(at ', 'Thu, 09 Jan 2020 12:12:13 GMT', ')')
================================================
tiles_start: 1464
tiles_end: 1952
tiles_start: 1952
tiles_end: 2440
================================================
('Factors are: (', 2277, ',', Decimal('4.391743522178304787000439174E+370'), ')
(at ', 'Thu, 09 Jan 2020 12:13:30 GMT', ')')
================================================
================================================
('Factors are: (', 2369, ',', Decimal('4.221190375685943436048965808E+370'), ')
```

```
(at ', 'Thu, 09 Jan 2020 12:13:37 GMT', ')')
================================================
tiles_start: 2440
tiles_end: 2928
================================================
('Factors are: (', 2783, ',', Decimal('3.59324469996406755300035932 4E+370'), ')
(at ', 'Thu, 09 Jan 2020 12:14:04 GMT', ')')
================================================

------------------------------------------------------------------------------
------------------------------------------------------------
742. (THEORY and FEATURE) Computational Geometric Factorization - Quadcore
Python 3.7.5 + Spark 2.4.3 benchmarks - 1024 bits integers - 11 January 2020 -
related to all sections on Factorization
------------------------------------------------------------------------------
------------------------------------------------------------
(*) Few 1024 bits hard random integers of mixed digits have been factorized and
logs have been captured in
testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.Python3.
7.5Upgrade.1024bits.11January2020 - all the integers are factorized from 28
seconds onwards - logs have been truncated and only first few factors are
printed. Integers factorized are:
      -
99999999029309203902939293232903920930293092039029302302382983927382382637627636
26362382392832893898198281928983927487847837479239289382983928938298398293829839
28938298398297327397298392893829382938283928938283982939283729392893829839288392
89382983928938298392893829839283982938239829389283982938298398293823
      -
99999999029309203902939293232903920930293092039029302302382983927382382637627636
26362382392832893898198281928983927487847837479239289382983928938298398293829839
28938298398297327397298392893829382938283928938288280837828729392893829839288392
89382983928938298392893829839283982938239829389283982938298398293823
      -
99999999029309203902939293232903920930293092039029302302382983927382382637627636
26362382392832893898198281928983927488383888888888883333383883892392899823829839
28938298398297327397298392893829382938283928938288280837828729392893829839288392
89382983928938298392893829839283982938239829389283982938298398293823
      -
99999999029309203902939293232903920930293092039029302302382983927382382637627636
26362382392832893898198281928983927488383888888888883333383883892392899823821111
11111111111111111111111111111111182938283928938288280837828729392893829839288392
89382983928938298392893829839283982938239829389283982938298398293823
      -
99999999029309203902939293232903920930293092039029388888999922222222188882777777
77777772277778938981982819289839274883838888888888833333838838923928998238211 11
11111111111111111111111111111111182938283928938288280837828729392893829839288392
89382983928938298392893829839283982938239829389283982938298398293833

(*) DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been
changed to print debug statements for verifying correctness of factorization -
or clause "if factorcandidate == number_to_factorize or xl*yl ==
number_to_factorize:" is true if point located in pixel arithmetic progression
array polygon of the rasterized hyperbola equals the number to factorize N in
two cases:
      (*) ordinate product xl*yl equals number to factorize N
            (and/or)
      (*) ordinate product (xl + intervalmidpoint)*yl equals number to factorize
N
(*) Truth values of both the above scenarios are printed and verified factor
ordered pairs are printed accordingly. Decimal division has been reverted
because of misleading mantissa-exponent notation

------------------------------------------------------------------------------
------------------------------------------------------------
```

743. (THEORY and FEATURE) Chaos Pseudorandom Generators - implementation update - Logistic and Lehmer-Palmore PRGs - 21 January 2020 - related to 318, 740 and all sections on Intrinsic Merit, Majority Voting, Ramsey coloring, Hardness Amplification, One-Way Functions, Chaos, Pseudorandomness and draft Randomized NC Chaos PRG defined in https://sites.google.com/site/kuja27/ChaoticPRG.pdf
---
(*) Logistic Equation Chaos PRG implementation in ChaosAttractor.py has been updated to also compute Lehmer-Palmore PRG by a function argument. An if clause has been introduced to choose between parabola attractor rx(1-x) and Lehmer-Palmore PRG (Bx mod M)
(*) ChaosAttractor.py Lorenz Attractor code has been refactored and made a function of parameters:
ChaosPRG(algorithm="Logistic",seqlen=100,radix=10,initialcondition=0.7,prime=104729) which respectively choose the algorithm ("Logistic" equation or "Lehmer-Palmore" prime modulus PRGs), length of pseudorandom sequence, radix (k for "Logistic" and B for "Lehmer-Palmore"), initial condition for sensitive dependence and a Large prime modulus
(*) R cor() correlation function has been invoked by R+rpy2 to print a correlation between pseudorandom sequence and a sample DJIA stock quotes dataset.
(*) ChaosPRG() function can be optionally invoked in NeuronRain wherever pseudorandomness is necessary - e.g Pseudorandom shuffle simulation of paper ballot in Drone Electronic Voting Machine, Pseudorandom Shuffle of 2-colored (0 and 1) leaves in a boolean majority voting circuit which gives rise to monochromatic arithmetic progression order from pseudorandom chaotic shuffled votes disorder by Ramsey theory (Van Der Waerden numbers), Economic merit - Models for Econometric Datasets
(*) Section 318 states a Majority version of XOR Hardness Amplification Lemma and points 318.15 to 318.19 define hardness of inverting boolean majority + voter SAT composition which might imply existence of an one way function (a #P-complete counting problem). Emergence of arithmetic progression order from binary colored voter leaves of Boolean Majority Circuit stipulates a prima facie indistinguishability condition between a good PRG and true randomness - Good pseudorandom shuffle must minimize probability of emergence of monochromatic arithmetic progressions in the leaves of boolean majority circuit.
(*) Probability of monochromatic arithmetic progressions in any pseudorandom bichromatic sequence of length N (including shuffled 2-colored voter leaves of boolean majority circuit) is inversely proportional to Van Der Waerden number VDW(N,2,r) - N elements in sequence(voters), 2 colors (candidates 0 and 1), arithmetic progression of length >= r (voters who chose same candidate are equally spread out in the shuffled bichromatic votes string).
(*) Chaos implies disorder from order (Period Three Theorem implies any sequence of period 3 can have larger periods while Feigenbaum sequences define self-referential recursive universality order in Chaos) and monochromatic arithmetic progressions in Pseudorandom multichromatic sequences imply order from disorder - these are two mutually conflicting trends in diametrically opposite directions.
(*) logs testlogs/ChaosAttractor.LehmerPalmorePRG.log.21January2020 and testlogs/ChaosAttractor.LogisticEquationPRG.log.21January2020 print a correlation between DJIA data and two chaotic PRGs - Logistic and Lehmer-Palmore. Opposing correlation signs between Logistic and Lehmer-Palmore PRGs and DJIA sample is an indicator of extent of chaos and pattern in economic data:
Logistic: correlation coefficient of pseudorandom and DJIA sequences is: -0.00173395205970038
Lehmer-Palmore: correlation coefficient of pseudorandom and DJIA sequences is: 0.00648048041159691
(*) Equilibrium point where the aforementioned Orderly Disorder (Ramsey) and Disorderly Order (Chaos) coincide might be of independent area of research.

References:

743.1 Chaos - [James Gleick] - Mandelbrot-Julia sets, Fractals, Koch curves, Lorenz attractors, Feigenbaum universality - Period Doubling, Yorke's Period

Three Theorem
743.2 Feigenbaum constants - First and Second -
https://en.wikipedia.org/wiki/Feigenbaum_constants

---
744. (THEORY and FEATURE) Chaotic Pseudorandom Generator - Python 3.7.5 upgrade, Computational Geometric Factorization - 30 January 2020 -
related to all sections on Pseudorandom Generators, Chaos and Factorization
---
1.This commit upgrades Chaos Pseudorandom Generator implementation to Python 3.7.5 by autopep8 followed by 2to3, which is the defacto NeuronRain recommended python version for performance reasons.
2.Both Logistic and Lehmer-Palmore PRGs have been tested on Python 3.7.5 + R-py2 combine and logs are committed to testlogs/ChaosAttractor.Python3.7.5Upgrade.30January2020
3.GRAFIT course material on Geometric search of BigData in https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/LinuxKernelAndCloud/BigdataAnalyticsCloud_CourseNotes.txt describe a Randomized NC factorization algorithm based on Parallel Chaotic Pseudorandom Generator and Central Limit Theorem based Space filling Linear Program mentioned in drafts - https://sites.google.com/site/kuja27/ChaoticPRG.pdf and https://sites.google.com/site/kuja27/Analysis%20of%20a%20Randomized%20Space%20Filling%20Algorithm%20and%20its%20Linear%20Program%20Formulation.pdf .
4.Exact PRAM-NC-BSP computational geometric factorization algorithm implemented on Spark in NeuronRain AstroInfer is the derandomized version of Randomized NC Bernoulli trials factorization algorithm in https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/LinuxKernelAndCloud/BigdataAnalyticsCloud_CourseNotes.txt which has the success probability of finding atleast one prime factor pr(m) = (1 - sqrt(2N)/(N*sqrt(2) - sqrt(2) - sqrt(N)))^(m-1)*(sqrt(2N)/(N*sqrt(2) - sqrt(2) - sqrt(N)) after m trials for square embedded hyperbolic arc. Expected number of trials E(m) before atleast one factor is found = sum_x=1-to-N(x*pr(x))
5.Thus NeuronRain Theory drafts define 3 different Nick's class Computational Geometric Integer factorization algorithms - Randomized NC, Quantum NC, Exact NC - which transition from a randomized, cohered quantum state to classical exact decohered state.

References:

744.1 Chaos - [James Gleick] - Pages 22,61,79 - Chaotic behaviour and non-linear modelling of epidemics - Period doubling in Measles,Rubella,Polio - Logistic equation
744.2 Simple mathematical models with very complicated dynamics - [Nature 261 459-67, 1976 - Robert May] - http://abel.harvard.edu/archive/118r_spring_05/docs/may.pdf - Logistic equation X(t+1) = a*X(t)*(1-X(t)) - Equations 3,4 - Figure 4 on Period doubling and Chaos after a > 3.57 - References 7,8,9,10,11,28 thereof on impact of Chaos on epidemiology,economics,social sciences.
744.3 CORD-19 - COVID-19 dataset - https://registry.opendata.aws/cord-19/, https://pages.semanticscholar.org/coronavirus-research
744.4 Foundations of Economic Analysis - [PA Samuelson] - Page 291 - Verhulst-Pearl-Reed Logistic Law - https://ia801603.us.archive.org/7/items/in.ernet.dli.2015.150369/2015.150369.Foundations-Of-Economic-Analysis.pdf
744.5 Game theoretic model of epidemics - https://www.elsevier.com/connect/using-game-theory-to-predict-peoples-behavior-in-an-epidemic - "...When there’s no epidemic, everyone behaves normally. In the event of an outbreak, changing behavior will be costly in some way – perhaps people stay home from work or avoid crowded environments, change their commute or buy preventative medication. But there’s also a payoff: people who change behavior are less likely to be infected. At a certain point in the epidemic, those people have an advantage in the population, and others begin to copy their

behavior...We applied game theory to these dynamics and produced a mathematical model that shows how people’s responses can affect the spread of a disease during an epidemic. We found that reducing the number of people an individual is in contact with, even by a small amount, can make a difference to the spread of disease..." - NeuronRain USBmd analyzer for malwares game theoretically (game between saboteur and sabotaged) is motivated on creating information or mining patterns in software viri and cybercrimes to counteract the spread, isolate infected systems, shutting down broadband networks and any instrument emitting RF(radio-frequency) and zero-in on origin - a PredPol. Translating this to epidemiology is obvious - information on prevention minimizes infection. But an apparent contradiction arises - Per 744.2 Verhulst model, if a > 3.57 when epidemic becomes chaotic to predict, rate of infection could outpace and be out of sync with information. Epidemics are also modelled by fitting to polynomials, exponential distribution, poisson distribution and logistic/linear regression - COVID19 exponential outbreak - https://www.nytimes.com/2020/03/13/science/coronavirus-math-mitigation-distancing.html.
744.6 Reproduction Number of COVID19 transmission - https://www.thelancet.com/journals/laninf/article/PIIS1473-3099(20)30144-4/fulltext - "... We modelled transmission as a geometric random walk process, and we used sequential Monte Carlo simulation to infer the transmission rate over time, as well as the resulting number of cases and the time-varying basic reproduction number (Rt), defined here as the mean number of secondary cases generated by a typical infectious individual on each day in a full susceptible population ..." - By modelling transmission as edges added/deleted in dynamic random graph between infected individuals, Reproduction Number is the average degree of Dynamic Infection Random graph.
744.7 Social Networks, Erdos-Renyi(ER) Random Graphs, Epidemics and SIR model, Small World, Clustering coefficient, Reproduction ratio - https://arxiv.org/pdf/1111.4875.pdf - Epidemic ER-SIR model applies as well to Cybercrimes - malwares,viri,bots.
744.8 Random Graphs, Quantum Chaos, Riemann Zeta Function, Graph representation of primes, Ihara Zeta Function, k-core, Expander graphs, Pseudorandomness - http://www.its.caltech.edu/~matilde/GraphsRandomChaosQuantum.pdf
744.9 Forest Fire Model or epidemic model of Social Networks - new node v connects to an ambassador node w, and links to x neighbours of w by forward burning and backward burning probabilities which is recursed - Data Mining - Section 9.2.2 - Page 560 - [Jiawei Han-Micheline Kamber] - this is typical of citation graphs: paper v citing w recursively consults subset of articles cited by w.

---
745. (THEORY and FEATURE) Compressed Sensing - Alphabet-Syllable vectorspace embedding - Python 3.7.5 upgrade - update - 4 February 2020
---
1. Function syllable_boundary_text_compression(self, text, vectorspace_embedding=True) in CompressedSensing python implementation has been updated to parametrically choose alphabet vectorspace embedding of words in text described in GRAFIT course material on Geometric Search - https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/LinuxKernelAndCloud/BigdataAnalyticsCloud_CourseNotes.txt - words of text are embedded in |A|^m vectorspace (|A| is the size of natural language alphabet and m is the longest word in text). For English, this is a space of 26^m.
2.Words are python tokenized to vectors and concatenated to an array creating a vectorspace embedding words in the text document.
3.Return value is a tuple which includes or excludes the word embedding of text depending on vectorspace_embedding boolean flag
4.CompressedSensing.py has been upgraded to Python 3.7.5 by autopep8 and 2to3 utilities. Python 2.7 version is separately committed.
5.An Example sentence is syllable hyphenated (PyHyphen), vowel stripped-compressed and embedded as:
##########

####################
Vowelless Syllable Vector Compression for text - This sentence is alphabet-syllable vector space embedded : (['This', 'sen', 'tence', 'al', 'pha', 'bet', '-', 'syl', 'la', 'ble', 'vec', 'tor', 'space', 'em', 'bed', 'ded'], 'Th_s-s_n-t_nc_-_l-ph_-b_t---syl-l_-bl_-v_c-t_r-sp_c_-_m-b_d-d_d')
Alphabet vectorspace embedding of text: [['T', 'h', 'i', 's'], ['s', 'e', 'n', 't', 'e', 'n', 'c', 'e'], ['i', 's'], ['a', 'l', 'p', 'h', 'a', 'b', 'e', 't', '-', 's', 'y', 'l', 'l', 'a', 'b', 'l', 'e'], ['v', 'e', 'c', 't', 'o', 'r'], ['s', 'p', 'a', 'c', 'e'], ['e', 'm', 'b', 'e', 'd', 'd', 'e', 'd'], []]
6.This kind of embedding is unconventional because it lifts an one dimensional text to an m-dimensional vectorspace nullifying the meaning of the word whereas traditional word embeddings reduce dimensionality and map words in higher dimensional space to real vectorspace of lower dimensions. Following the linguistic philosophy of "word is characterized by company it keeps" by [Firth] (Studies in Linguistic Analysis - http://cs.brown.edu/courses/csci2952d/readings/lecture1-firth.pdf), alphabet embedding above could be a framework for assessing colocated words by euclidean distance e.g "vector" and "space" are colocated in previous example and are closer in meaning (e.g WordNet distance) but have high euclidean distance. This kind of alphabet embedding is quite relevant to similar name clustering where same name is spelt differently ("What is in a name" is a non-trivial question to answer).

----
746. (THEORY and FEATURE) Social Network Analysis - People Analytics - Alphabet-Syllable vectorspace embedding - Python 3.7.5 upgrade - Name clustering - 4 February 2020

1.SocialNetworkAnalysis_PeopleAnalytics.py and its dependencies have been upgraded to Python 3.7.5 (CompressedSensing.py,TextCompression.py,WordSubstringProbabilities.py) by autopep8 and 2to3 utilities. Python 2.7 versions have been separately committed.
2.Example Professional Profile (of the author) has been updated in testlogs/CV.pdf and testlogs/CV.tex which is parsed for text and previously updated syllable_boundary_text_compression() function of CompressedSensing.py is invoked on profile text for alphabet vectorspace embedding of words.
3.Logs for this commit are in testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.Python3.7.5UpgradeWordEmbedding.4February2020 which demonstrate 3 alphabet vectorspace embeddings of same first name of the author spelt differently having low mutual euclidean distance.
4.Syllable embedding instead of alphabet embedding of words causes linguistic boundaries to vanish which has been mentioned in an earlier code commit - transliterating text of one script lacking compound alphabets to another script bestowed with compound alphabets helps in natural language text compression.
5.Match Rating algorithm NYSIIS (https://en.wikipedia.org/wiki/New_York_State_Identification_and_Intelligence_System) which is better than Soundex codes is an example of efficient Similar Name Search. Linguistically similar names are intersections of heterographs (different spellings and meanings) and synonyms (different spellings and same meaning) - Homophones - https://en.wikipedia.org/wiki/Homophone

----
----
747. (THEORY and FEATURE) Computational Geometric Spark NC-PRAM-BSP Factorization Benchmarks - 2014 bits, 2037 bits, 2050 bits integers - Quadcore Spark 2.4.3 + Python 3.7.5 - 5 February 2020

1.Computational Geometric Factorization Spark NC-PRAM-BSP implementation has been benchmarked on 3 2000+ bits integers both smooth and hard and factorization logs have been captured in testlogs/DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.Python3.

7.5Upgrade.2000plusbits.5February2020
2.Following are the 2000+ bits integers benchmarked:
      2.1 2014 bits smooth integer (heuristically integers having lot of 9s are easy to factorize) - all 9s -
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
9999999999999999999999999999999999999999999999
(first few factors are 3,9,13,33,... found from 18 seconds onwards)
      2.2 2037 bits less smooth integer - mixed digits,most 9s -
12891729999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999999999999999987108279
(first few factors are 3,9,11,27,33,... found from 18 seconds onwards)
      2.3 2050 bits hard integer (heuristically integers having lot of 7s are hard to factorize) - mixed digits,most 7s -
16333539509987274314748077777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777777777777777777
7777777777777777777777777777777777761444238267905034630297
(first few factors are 7,9,11,21,27,... found from 204 seconds onwards - unusually high duration compared to previous 2000+ bits integers implies hardness)
3. Previous 2000plus bits integers are the largest factorized by NeuronRain AstroInfer Spark NC-PRAM-BSP implementation.
4. Depth of the non-uniform circuits are 182,184 and 184 respectively for 2014 bits, 2037 bits and 2050 bits integers. PRAM time O(logN)^k) for 2050 bits factorization circuit of depth 184 therefore is O(2050^184) and number of PRAMs required are O(2^2050/2050^184) = O(2^2050/2^2025) = O(2^25) = 33554432 which is high number of multicores but plausible in a geographically distributed BSP cloud.
5. Increasing depth slows down the PRAM time but reduces the parallel RAMs. Notwithstanding this theoretically prohibitive parallelism factors of 2050 bits hard integer were found in quadcore in few minutes.

----
748. (THEORY and FEATURE) Leaky Bucket Timeseries Analyzer Implementation - Python 3.7.5 - 12 February 2020

1.This commit implements Leaky Bucket Timeseries Analyzer Algorithm mentioned in GRAFIT course material https://github.com/shrinivaasanka/Grafit/blob/master/course_material/NeuronRain/AdvancedComputerScienceAndMachineLearning/AdvancedComputerScienceAndMachineLearning.txt for generic timeseries analytics.
2.Streaming_LeakyBucket.py instantiates Streaming Abstract Generator buffer incoming stream (example Dictionary.txt file data source is analyzed as timeseries) for generic data source and data storage of specified buffer size.
3.leaky_bucket_timeseries_analyzer() iterates through buffer incoming stream and in a loop populates a deque object till it reaches maximum size . Elements

exceeding max size are overflown and printed. Deque simulates a leaky bucket - incoming elements fill the bucket from front end (insert() at index 0) and outgoing elements leak through other end.
4.Separate thread outgoing_stream_thread() independently scans the leaky bucket buffer and pops the elements from the deque. It busy waits if the leaky bucket deque is empty and continues to pop after buffer is refilled.
5.Synchronization is by a binary semaphore guarding the deque in main thread and outgoing stream thread.
6.Theoretical overtone of leaky bucket is its ability to flag outliers by overflows and peaks in timeseries are often the abnormalities in stream causing overflow. For example excessive network traffic corresponds to peaks in timeseries data and frequent overflows in leaky bucket as buffer fills faster than leakage.
7.Leaky Bucket Analyzer thus is a universal tool for mining any stream of text,audio,video,people data and most apt for wireless network stream, stream of celestial bodies configurations, climate change datastream (e.g http://climate.nasa.gov data stream on surface temperature, melting polar ice sheets, carbon emission, ocean currents and acidity), economic datasets having lot of outlier spikes.
8.Streaming_LeakyBucket.py has been implemented on Python 3.7.5 and Streaming_AbstractGenerator.py has been upgraded to Python 3.7.5
9.Logs for dictionary time series example are at testlogs/Streaming_LeakyBucket.log.12February2020

---

749. (THEORY and FEATURE) PAC Learning - Python 3.7.5 upgrade, Riemann Hypothesis, Patterns in Primes, class HNc and Algebraic Circuits - 21 February 2020, 22 February 2020 - related to 24,629 and all sections on Complement diophantines, Complementary Sets and Equations, Prime-Composite complementation and Patterns in Primes

---

1. PACLearning.py PAC learning implementation has been refactored and upgraded to Python 3.7.5. Binary encoded and bits mapped UTM Primes dataset has been PAC Learnt again for per-prime-bit-position boolean disjunctions of first 10000 primes (16 disjunctions for each bit of 16-bits prime integer). New refactored PACLearning() function accepts a generic datasource and a hypothesis dictionary as arguments.
2. Example logs for this PAC learning are at testlogs/PACLearning.PrimeBitsMappingConjunctions.Python3.7.5.21February2020 which are different from Python 2.7 approximate learning.
3. Prime pattern References in Section 24 (24.31, 24.32, 24.33) mention a prima facie pattern in juxtaposed primes especially their avoidance of same digit ending. PAC learnt hypothesis in testlogs/PACLearning.PrimeBitsMappingConjunctions.Python3.7.5.21February2020 show an intriguing avoidance pattern in 14th,15th and 16th bits for first 10000 primes though approximate.
4. Prime counting function (number of primes <= N) can be written in terms of non-trivial zeros of Riemann Zeta Function. Riemann Hypothesis (RH) conjectures all non-trivial zeros of Riemann Zeta Function are along critical line Re(z)=0.5. Riemann Zeta Function is just a rephrased version of Sieve of Eratosthenes and is expanded by Euler product.
5. From Section 629, Riemann Zeta Function has an algebraic computation tree circuit in HNc:
"... Riemann Zeta Function can be represented as a disjunctive clauses or non-uniform Hilbert Nullstellensatz algebraic circuit in
0-1-HNc (0 if s is a non-trivial zero else 1) of product polynomials having complex roots (non-trivial zeros):
RZF=product_of(1/[1-prime(k)^(-s)]) for all primes 2,3,5,7,..."
6.Approximate binary digit hypothesis disjunctions for primes dataset learnt by PAC and exact learning by Euler-Fourier polynomial imply pattern in spacings between consecutive primes and thus are indirect boolean prime counting functions. Both Euler-Fourier polynomial and PAC learnt disjunctive clauses are

mapping functions between integer n and i-th bit of n-th prime - f(n,i): n <=> i-th bit of n-th prime
7.Riemann Zeta Function has diophantine equation representation and thus in RE by MRDP theorem. Recent result of MIP*=RE implies a quantum multiparty interactive prover-classical verifier algorithm for diophantine and algebraic circuit versions of Riemann Zeta Function. It is notable and coincidental that distribution of Gaussian Unitary Ensemble eigenvalues in quantum world are identical to spacings of non-trivial zeros of RZF.

References:

749.1 Prime Obsession - Bernhard Riemann and the Greatest Unsolved Problem in Mathematics - [John Derbyshire] - Chapters 18,19,21 - Pages 292,299,309,328 - Montgomery-Odlyzko law, Euler product, Sieve of Eratosthenes, Function J(x) in terms of Prime counting function Pi(x) and Riemann Zeta Function in terms of J(x) - Equations 19.1,19.6,21.1 - " ... 1/s log(RZF(s)) = Integral(J(x) x^(-s-1) dx) ..."
749.2 MIP*=RE - https://arxiv.org/abs/2001.04383 - this result implies quantum computers could decide halting problem which is in RE. Diophantine representation of RZF is in Adleman-Manders class D and is in RE by MRDP theorem. MIP*=RE implies Riemann Zeta Function (RZF) and Riemann Hypothesis has a quantum verification algorithm which halts despite infinite number of non-trivial zeros of RZF. Similar to computerized proof of Four color theorem this could be a machine generated proof or disproof of Riemann Hypothesis - An example quantum algorithm for proving or disproving Riemann hypothesis in MIP*=RE:
- Multiparty interactive quantum prover finds all non-trivial zeros of RZF
- classical verifier checks real part of all non-trivial zeroes for Re(z)=0.5 and finds a counter-example for disproof or a lack of it for proof of Riemann Hypothesis.

[Intermediary sections 750-781 are scattered in other NeuronRain repositories - Acadpdrafts,USBmd,VIRGO,KingCobra,GRAFIT,Krishna_iResearch_DoxygenDocs]

---
782. (THEORY and FEATURE) Text Compression, Compressed Sensing, Syllable boundaries decompression implementation - Python 3.7.5 upgrade and refactoring - 28 February 2020, 29 February 2020

1.TextCompression.py vowelless text compression-decompression implementation has been upgraded to Python 3.7.5 by autopep8 and 2to3 utilities
2.Syllable boundaries text compression function from CompressedSensing.py has been imported to optionally choose between syllable boundary compression of text and earlier vowelless compression (which strips vowels across the board) by a boolean flag syllable_boundary
3.Earlier commented code for PyEnchant spelling suggestions has been uncommented to choose between PyEnchant maximum likelihood (prob() function) and Hidden Markov Model maximum likelihood (HiddenMarkovModel_MLE() function) estimators based on if suggestions is not-None and None respectively.
4.Both PyEnchant and HMM computed maximum likelihood words are printed to logs.
5.Logs at testlogs/TextCompression.log.28February2020 contain PyEnchant and Hidden Markov Model maximum likelihood estimations per decompressed string for both vowel compressed and syllable boundary compressed words
6.decompressedtext.txt contains the newly introduced approximate syllable boundary decompressed text:

ghts fer over with lus tins der ly2 on dead ster nal ve tting pro mer dial placed be form byss paw er ful ry mev ing grant ybody wn ters ician milk shy ser pant ged form ing cotch shed sh mania

for exact text:
Rama gets forever pleased with him who listens to or reads Ramayana daily. He is

indeed the eternal Vishnu, the Lord of preservation. Rama is the primordial Lord, clearly placed before the eyes the powerful Lord removing the sins and the great-armed, who has abode on waters of the ocean of milk Sesha the serpent-god forming his couch is said to be Lakshmana.

7. It is worth observing how the words "primordial", "removing","serpent" and "powerful" have been syllable decompressed - "pro mer dial", "ry mev ing", "ser pant", "paw er ful" - to homophones (phonetically equivalent but linguistically different)
8.Advantage of syllable boundaries is their language and script independence, dependence on phonetics alone, fine-graining of words to syllables for better accuracy in phonetic decompression.
9.Embedding a text on syllable vector space as opposed to traditional flat one-dimension thus blurs linguistic boundaries and useful for similar name clustering.

References:

782.1 Phonetic Match Rating Algorithm - for similar name clustering in airlines - https://ia801704.us.archive.org/2/items/accessingindivid00moor/accessingindivid00moor.pdf

---
783. (THEORY) Phonetic Syllable Boundaries Compression and Approximate Decompression - error bounds and success amplification - 1 March 2020 - related to all sections on Text Compression, Compressed Sensing and People Analytics - Similar name match
---
Approximate Syllable Boundaries Decompression in earlier example can be phonetically matched for further error correction and accuracy. Example decompression of "powerful" to "paw er ful" demonstrates approximate phonetic match which requires lookup of per word audio dictionary to spell out a matching word (e.g Interactive Voice Response). By finding the best phonetic match of "paw er ful" in a syllable audio dictionary (which is a map of collated-syllable-audio-of-words to words), "paw er ful" is further decoded to "powerful" thus amplifying success. Phonetic match therefore involves an audio waveform distance function of two words different from string edit distance text similarity (e.g waveforms of "paw er ful" and "powerful" are in proximity). This audio-to-words dictionary is static containing as many words in language and requires initialization only once. Another prerequisite is a syllables-to-audio dictionary for lookup of every syllable in approximately decompressed word and splice their audio waveforms (e.g waveforms of syllables "paw","er","ful" are lookedup and collated to obtain "paw er ful") which is then lookedup in audio-to-words dictionary (e.g waveform for "paw"+"er"+"ful" is lookedup to retrieve "powerful").

Number of vowels per english word (approximate) = length_of_the_word / 3
(because on the average every third letter of a word is a vowel)
Number of vowels per english word syllable (approximate) = 1
Number of syllables per english word = length_of_the_word / 3

Probability of erroneous decompression per english word syllable = 4/5 = choice of wrong vowel in a,e,i,o,u
Probability of wrong decompression of a word = (0.8)^(l/3) (l = length_of_the_word)
Probability of successful phonetic decompression increases therefore with increasing length of the word.

[Intermediary sections 784-791 are scattered in other NeuronRain repositories - Acadpdrafts,USBmd,VIRGO,KingCobra,GRAFIT,Krishna_iResearch_DoxygenDocs]

---
792. (FEATURE) Social Network Analytics - LinkedIn Connections Analysis, Name filter and Python 3.7.5 upgrades - 4 March 2020

1.SocialNetworkAnalysis_PeopleAnalytics.py - Connections in a Professional Network Profile have been analyzed by Recursive Lambda Function Growth (RLFG) after applying name filters to strip off Human names of connections and only extract their professional roles.
2.Name filter works by looking up words in profile connections text in a dictionary and filters Proper nouns (Human Names) - rationale is a language dictionary should not contain human names or contain least human names and is a short-cut stop-gap algorithm to recognize proper nouns while an artificial intelligence culture-neutral name parsing alternative might involve extensive training data and deep learning.
3.Dictionary.txt file already committed for CompressedSensing has been preferred to nondictionaryword() in RecursiveGlossOverlap_Classifier.py which repetitively invokes slower REST lookup - first word in each line of Dictionary.txt (Concise Oxford Dictionary) has been parsed to create an array of words
4.Quite a few dependent python source files have been upgraded to Python 3.7.5 including the very necessary RecursiveLambaFunctionGrowth.py implementation for performance by autopep8 and 2to3 utilities.
5.Dense subgraph extracted from definition graph of professional descriptions of connections is an indication of prominent designations/skills pattern.
6.Machine Translation has been commented because of slow REST calls to Google Translate.
7.Unfeasible Exception has been handled in korner_entropy() - maximal independent set computation.
8.Only a fraction of connections profile contents is analyzed because of intensive computation.
9.logs at testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.ConnectionsRLFGAnalysis.4March2020.gz contain exhaustive graph theoretic analysis of LinkedIn Connections of the author (K.Srinivasan - https://www.linkedin.com/in/srinivasan-kannan-608a671/) and to large extent core numbers reflect the designations/skills profile.
10.Of all the graph complexity measures, closeness centrality is more accurate than others - top ranked vertices which directly relate to designations are (and still there are Human names because dictionary attack is just an approximation):
[('system', 0.07260504201680672), ('computer_architecture', 0.05979238754325259), ('made', 0.05585003232062055), ('ecosystem', 0.05464895635673624), ('son', 0.05163025210084034), ('John', 0.050760079312623926), ('male', 0.050760079312623926), ('servicing', 0.049373191899710706), ('logic', 0.04934323243860651), ('considered', 0.048403361344537814), ('forming', 0.048403361344537814), ('components', 0.048403361344537814), ('interdependent', 0.048403361344537814), ('unified', 0.048403361344537814), ('body', 0.048403361344537814), ('whole', 0.048403361344537814), ('living', 0.048403361344537814), ('act', 0.04657368101879927), ('computer', 0.04537815126050421), ('permanent_wave', 0.044582043343653247), ('engagement', 0.04364876385336743), ('science', 0.042352941176470586), ('organization', 0.042352941176470586), ('software', 0.042352941176470586), ('structure', 0.042352941176470586), ('hardware', 0.042352941176470586), ('human', 0.04069952305246423), ('offspring', 0.04069952305246423), ('1215', 0.04015686274509804), ('sign', 0.04015686274509804), ('Henry', 0.04015686274509804), ('youngest', 0.04015686274509804), ('French', 0.04015686274509804), ('Magna', 0.04015686274509804), ('compelled', 0.04015686274509804), ('Richard', 0.04015686274509804), ('barons', 0.04015686274509804), ('1167-1216', 0.04015686274509804), ('1216', 0.04015686274509804), ('throne', 0.04015686274509804), ('brother', 0.04015686274509804), ('England', 0.04015686274509804), ('death', 0.04015686274509804), ('1199', 0.04015686274509804), ('II', 0.04015686274509804), ('lost', 0.04015686274509804), ('King', 0.04015686274509804), ('possessions',

0.04015686274509804), ('Carta', 0.04015686274509804), ('succeeded',
0.04015686274509804), ('group', 0.03997917751171265), ('organisms',
0.039705882352941174), ('physical', 0.039705882352941174), ('formed',
0.039705882352941174), ('interaction', 0.039705882352941174), ('community',
0.039705882352941174), ('environment', 0.039705882352941174), ('mating',
0.03928388746803069), ('animals', 0.03928388746803069), ('product',
0.038431372549019606), ('written', 0.03796078431372549), ('reasoning',
0.03682864450127877), ('strontium', 0.036580138128125744), ('writer',
0.035812060673325936), ('sharing', 0.03557202408522464), ('activities',
0.03557202408522464), ('infrastructure', 0.03529411764705882), ('machine',
0.034573829531812726), ('performing', 0.034573829531812726), ('automatically',
0.034573829531812726), ('calculations', 0.034573829531812726), ('hair',
0.03410975128306356), ('applying', 0.03410975128306356), ('heat',
0.03410975128306356), ('waves', 0.03410975128306356), ('series',
0.03410975128306356), ('chemicals', 0.03410975128306356), ('Das_Kapital',
0.03329893360853113), ('two', 0.031512605042016806), ('era',
0.031512605042016806), ('timbre', 0.030745098039215688), ('silver-white',
0.030732292917166865), ('element', 0.030732292917166865), ('yellowish',
0.030732292917166865), ('turns', 0.030732292917166865), ('celestite',
0.030732292917166865), ('metal', 0.030732292917166865), ('occurs',
0.030732292917166865), ('air', 0.030732292917166865), ('strontianite',
0.030732292917166865), ('soft', 0.030732292917166865), ('metallic',
0.030732292917166865), ('alkali', 0.030732292917166865), ('yellow',
0.030732292917166865), ('person', 0.029656862745098038), ('complex',
0.02956259426847662), ('summation', 0.029411764705882353), ('policy',
0.02920060331825038), ('produce', 0.028467683369644153), ('trade_name',
0.028467683369644153), ('mood', 0.027450980392156862), ('college',
0.026504394861392833), ('something', 0.02564917859035506), ('able',
0.02564917859035506), ('write', 0.02564917859035506), ('leader',
0.02464985994397759), ('working', 0.024546114742193172), ('former',
0.024434389140271493), ('1867', 0.02433383609854198), ('economic',
0.02433383609854198), ('Marx', 0.02433383609854198), ('theories',
0.02433383609854198), ('Karl', 0.02433383609854198), ('describing',
0.02433383609854198), ('book', 0.02433383609854198), ('large',
0.023529411764705882), ('booth', 0.023529411764705882), ('circumstances',
0.023291740938799762), ('someone', 0.023291740938799762), ('set',
0.023291740938799762), ('efforts', 0.023291740938799762), ('consequence',
0.023291740938799762), ('particular', 0.023291740938799762), ('contract',
0.022070223438212494), ('insurance', 0.022070223438212494), ('certificate',
0.022070223438212494), ('activity', 0.02137887413029728), ('serving',
0.02137887413029728), ('Indo-European', 0.021176470588235293), ('divided',
0.02100840336134454), ('geological', 0.02100840336134454), ('time',
0.02100840336134454), ('major', 0.02100840336134454), ('division',
0.02100840336134454), ('periods', 0.02100840336134454), ('usually',
0.02100840336134454), ('musical', 0.020227038183694528), ('sound',
0.020227038183694528), ('music', 0.020227038183694528), ('property',
0.020227038183694528), ('noise', 0.020227038183694528), ('distinctive',
0.020227038183694528), ('voice', 0.020227038183694528), ('a....]

---
793. (THEORY and FEATURE) GIS and Urban Sprawl Analytics - Image Segmentation and Face components statistics,Dual Graph,Four color theorem - 12 March 2020

1.ImageNet Keras-Theano + OpenCV + Python 3.7.5 implementation - ImageGraph_Keras_Theano.py - has been changed for computing statistics of image segmentation by watershed algorithm.
2.image_segmentation() function has been augmented to invoke cv2.connectedComponentsWithStatsWithAlgorithm(sure_fg,connectivity=8,ltype=2,ccltype=cv2.CCL_GRANA) which computes the connected face components of foreground image accompanied by statistics and algorithm parameters. 8-way connectivity of an image by GRANA connected components algorithm is computed.

cv2.connectedComponents() is limited to markers only.
3.Labels,Centroids and Statistics of each connected component of a segmented image are printed. Every component of segmented image is also a face in graph theoretic terms and a dual graph of adjacent faces can be drawn connecting centroids of faces.
4.By Four color theorem (a kind of Multiple Agent Resource Allocation), centroids of adjacent faces could be made color-repulsive - every face and thus its centroid can be colored by one of the four categories (Manufacturing-IT-ITES,Residential,Commercial,Greenery) and adjacent faces avoid same color. Dense subgraphs of Colored Weighted Dual graph (vertices are colored by one of the 4 categories and edges are labelled by distance between centroids of the faces) of segmented urban sprawl GIS offer insight into urbanization, expansion and density.
5.An example montage MP4 video (Example Video 1 - Erstwhile Research Scholar profile - 1995-2011 - academics and industry - of the author) has been benchmarked again along with a segmentation of Chennai Urban Sprawl GIS JPEG (SEDAC - http://sedac.ciesin.columbia.edu/mapping/popest/gpw-v4/).
6.Logs at testlogs/ImageGraph_Keras_Theano.log.12March2020 show two prominent segments of Chennai Urban Sprawl and their centroids.

[Intermediary sections 794-809 are scattered in other NeuronRain repositories - Acadpdrafts,USBmd,VIRGO,KingCobra,GRAFIT,Krishna_iResearch_DoxygenDocs]

---
810. (THEORY and FEATURE) GIS and Urban Sprawl Analytics - Delaunay Triangulation and Voronoi diagram of Segmented Image, Intrinsic Merit of Large Scale Visuals, Computational Geometry - Planar Point Location, Computational Geometric Factorization - 16 March 2020, 17 March 2020, 18 March 2020

1.ImageGraph_Keras_Theano.py image_segmentation() has been updated to approximately compute face graph of a segmented image by Voronoi diagram for segment centroids.
2.Centroids from connected components statistics are construed to be the points of a Voronoi diagram planar subdivision.
3.A planar rectangle of the size of image is instantiated by Subdiv2D() and centroids of image segment components are populated in planar subdivision.
4.Delaunay triangles and Voronoi faces tesselation of the planar subdivision are obtained by OpenCV API - getTriangleList() and getVoronoiFacetsList().
5.Voronoi diagram draws boundaries optimally in a way such that centroids are equidistant from them and thus approximates the face graph for the connected segment components of the image which is essentially necessary to simulate envy-free allocation of four categories/colors of an urban sprawl and a fair division MARA. An alternative to Voronoi diagram could be to minimize a linear program of average weighted distance between centroids of segment faces - optimal weights of edges between face centroids decide the dimensions of every segment category:
Minimize Sum(distance(Cx - Cy)) / |C| for set of centroids C={C1,C2,...,Ck}
6.For every facet in the Voronoi diagram which corresponds to a segment, a NetworkX vertex object is created by stringified concatentation of pixel ordinates delimited by "#".This uniquely identifies a vertex in facegraph. Edges are added for each face boundary in NetworkX Undirected graph. This NetworkX facegraph of an image is sufficient for graph theoretic analysis of any image.
7.Polygon is drawn for each face and NetworkX facegraph of an image in its entirety is written to a DOT file.
8.Two GIS images of Chennai Urban Sprawl have been segmented by watershed and their facegraphs have been plotted as PNG images. (There are graph theoretic alternatives to watershed viz., GRABCUT graph based image segmentation algorithm which is on similar line)
9.New video montage - Google Search of NeuronRain repositories and Google Scholar profile of author https://scholar.google.com/citations?

user=eLZY7CIAAAAJ&hl=en - has been recorded as MP4 and analyzed for EventNet Tensor Product Merit which is a good mix of visuals and text. Incidentally every frame of a video can also be segmented graph theoretically as FaceGraph as against ImageGraph analysis from ImageNet predictions.
10. Segment centroid Voronoi diagrams are as well irregular bounding boxes implementations for an image as opposed to rectangles.
11. Voronoi Diagrams of frames in video dynamically morph and create a stream of FaceGraph(s) - for a movie or YouTube video example, actors per scene/event (set of consecutive frames) could be located by segmentation and Voronoi tesselation of segment centroids. Every segmented frame has a Voronoi FaceGraph whose faces contain actor centroids. For consecutive frames per scene, actor centroids arbitrarily float (Centroid Tracking) resulting in a stream of Dynamic Voronoi FaceGraph(s). Intuitively, set of FaceGraphs per scene or event are isomorphic. Merit of Visuals has two facets - style and substance. Substance of visual is measurable by Tensor Rank connectedness of information while Style often is a visual genre stereotype and narrative USP of a creator e.g laidback, offbeat, actionpacked, VFX. Patterns in narrative graph theoretically reduces to frequent subgraph mining problem - multiple movies or videos (stream of Voronoi facegraphs of frames) of a creator are mined for frequent subgraphs in facegraphs.
12. Streaming Voronoi Tesselation of Videos is a Computational Geometric solution which involves all pervasive Planar Point Location - Voronoi Diagram FaceGraph is a Planar Subdivision of each Frame and Points to be located/tracked are Centroids of Faces/Segments(actors).
13. Rasterized Point Location in Computational Geometric Factorization could be expressed in terms of Voronoi Diagrams:
      13.1 Factors of an Integer (already found) are centroids of segments located on hyperbolic arc
      13.2 Voronoi diagrams subdivide 2D plane and find faces comprising every factor centroid
      13.3 Planar Point Location on Voronoi 2D planar subdivision locates the polygon face containing a factor point. Advantage of Rasterized hyperbolic arc: every polygon face is an axis-parallel arithmetic progression enabling binary search within faces.
14.Urban Sprawls are real counterparts of Social networks in virtual world - WWW, dynamic and undergo rapid expansion by preferential attachment - Good example of such an expansion is Chennai Urban Sprawl which is 8848 sqkm an eightfold expansion in 50 years from 1189 sqkms in 1970s. It is imperative that resource allocation is in tandem and keeps pace with urbanization and none of the four categories are found wanting. Graph theoretically, Subgraphs of FaceGraph of an Urban Sprawl by Voronoi Diagrams of segment centroids could be fairly allocated to categories by Multi Agent Graph Coloring. Expanding Urban sprawl grows the FaceGraph dynamically and new segments of various categories (and faces) are added in the periphery of the city. Multi Agent Four coloring of Urban sprawl FaceGraph therefore cannot be static and must be recomputed to catch up with this expanding dynamic facegraph. High Edge Expander facegraph of an urban sprawl has least Cheeger constant (low bottlenecks, high regularity and low eigenvalues) and high connectivity.

References:

810.1 Multi Agent Graph Coloring - [Blum-Rosenschein] - https://www.aaai.org/Papers/AAAI/2008/AAAI08-004.pdf - "...Multiple agents hold disjoint autonomous subgraphs of a global graph, and every color used by the agents in coloring the graph has associated cost. In this multiagent graph color- ing scenario, we seek a minimum legal coloring of the global graph’s vertices such that the coloring is also Pareto efficient, socially fair, and individual rational ... Every node corresponds to an agent, and each agent tries to determine its color so that neighbors do not have the same color." - Socially Fair Coloring of FaceGraphs of Urban Sprawls could involve Four colors (categories or agents) - Manufacturing-IT-ITES,Residential,Commercial,Greenery - which compete to grab their pie (subgraph) of Urban sprawl. Pareto optimal 4-coloring of Urban sprawl FaceGraph finds an efficient resource allocation.

---
811. (THEORY and FEATURE) Interview Algorithm - Spark Recursive Gloss Overlap Implementation - Python 3.7.5 upgrade and Quadcore Benchmarks - 20 March 2020

1.Spark Recursive Gloss Overlap MapReduce implementation in python-src/InterviewAlgorithm has been upgraded to Python 3.7.5 and benchmarked on Quadcore.
2.Earlier implementation on Python 2.7 and Dual core was slower because of local[2] which has been upgraded to local[4] for quadcores.
3.Memcache python client has been changed to pymemcache for Python 3.7.5 and pymemcache API changes have been incorporated.
4.Spark Python Recursive Gloss Overlap sources have been compiled to C by Cython3 (upgraded from Cython2) - following are commandline alternatives to setup.py and .so(s):
      cython3 <Python3.7.5_file.py> -o <Cython3_file.c>
      gcc -I<include_Python3.7_headers> <Cython3_file.c> -o <Cython3_executable> -lpython3.7m
5.Unicode encode() for Python 3 strings has been invoked wherever necessary
6.Pickle file is opened in 'a' and not 'ab' because of Python 3 bytes type conflict
7.Python 2.7 versions still depend on memcache python client, have .2.7 extension and Cython 2 .so(s) remain intact.
8.Replacement of Flat file pickle in MapReduce by memcache get/set was tried but the code for it has been left commented because of huge number of connection reset errors within memcache. This is despite lockdown of pages in memcache.conf and parallel connections set to 1024. In-memoy cache of Synsets in Spark mapreduce could reduce latency further and might be uncommented later.
9.2 sentences in small one liner file have been mapped to 2 textgraphs and timestamped textgraph plot images and logs are at InterviewAlgorithm/testlogs/

---
812. (THEORY and FEATURE) People Analytics - Human Resource Analytics - Recursive Lambda Function Growth Intrinsic Merit of Name-filtered Profile - corrections, Chaotic Hidden Markov Model of Tenures/Attrition, Martingales - related to 418,420,613 - 25 March 2020

812.1.Recursive Lamda Function Growth Intrinsic Merit of a profile text has been benchmarked again on Python 3.7.5 upgraded implementation because of some misleading graph cores created in Python 2.7 by proper nouns (cities,organization names,institution names). Exceptions for word hyphenator maximum word length have been handled. Logs are at python-src/testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.25March2020
812.2.An example profile text (testlogs/CV.tex) of the author has been analyzed and dense subgraph analysis of its textgraph has been executed. Most of the proper nouns have been filtered from profile text but for few ambiguities (e.g "Sun","Global" are in dictionary but they were old first names of organizations author worked for and "Green" is name of NeuronRain repositories, "Gold medalist" is mistook for "Golf medalist")
812.3.By and large top percentile core numbers of the profile textgraph reasonably reflect the stature of the profile and name filtering increases relevance of the dense subgraphs. Though anything can be written on a CV, an id verified profile content is a trustworthy measure of merit (Trusted Social Network Connections of professional profile in organizations worked vouch for authenticity despite being fame based citations) and facilitates autonomous recruitment if Merit Lowerbounds Fame:
==========
==========
Unsupervised Classification based on top percentile Core numbers of the definition graph(subgraph of WordNet)

```
================================================================================
============================
('This document belongs to class:', 'senior', ',core number=', 4)
('This document belongs to class:', 'architect', ',core number=', 4)
('This document belongs to class:', 'college', ',core number=', 4)
...
('This document belongs to class:', 'group', ',core number=', 2)
('This document belongs to class:', 'swarm', ',core number=', 2)
('This document belongs to class:', 'belongs', ',core number=', 2)
('This document belongs to class:', 'Green', ',core number=', 2)
('This document belongs to class:', 'golf', ',core number=', 2)
('This document belongs to class:', 'medalist', ',core number=', 2)
('This document belongs to class:', 'essential', ',core number=', 2)
('This document belongs to class:', 'kernel', ',core number=', 2)
...
('This document belongs to class:', 'technical_school', ',core number=', 2)
('This document belongs to class:', 'semantic', ',core number=', 2)
...
('This document belongs to class:', 'expert', ',core number=', 2)
('This document belongs to class:', 'adviser', ',core number=', 2)
```

812.4.Predictability of Tenure transitions of a profile can be effectively expressed as Chaotic Hidden Markov Model which is a combination of Verhulst Logistic and Hidden Markov Model. Verhulst-Pearl-Reed Logistic defines present observation X(n+1) in terms of previous X(n): X(n+1) = Lambda*X(n)*(1-X(n)). Hidden Markov Model on similar lines defines state transition probabilities and observations in each state. For this example profile, state transitions are organizations (including academic tenures and opensource initiative of the author) - in a gist left-to-right in chronological order (spanning 25 years - 1995-2020):

```
--------------------------------------------------------------------------------
------------------------------------------------------------
States: PSG - BaaN/SSA - Sun/Oracle - Krishna-iResearch - Verizon -
webMethods/SoftwareAG - CMI/IIT/IMSc - GlobalAnalytics/GAIN - ClockWork -
CloudEnablers - Unaffiliated private research
Observations: Could be any parameter - academic or work, opensource
contribution, tenure duration, city, remuneration, designation in each
organization, proven research, domain
--------------------------------------------------------------------------------
------------------------------------------------------------
```

812.5.Both Verhulst Law and Hidden Markov Model assume observation on present state depends only on observation on previous state and not cumulatively conditioned on all previous states or tenure switch is decided only by present independent of history (which would have otherwise been a Martingale sequence - E(Xi/X0,X1,X2,...,X(i-1)) = X(i-1) - expected observation in a state conditioned on all past states is the previous state observation - if each random variable Xi signifies one of the aforementioned observed parameters in an academic/work tenure state i). Chaotic Hidden Markov Model which is a merger based on this common fact underlying Chaos and HMM adds one more dimension to predictability of a profile or lack of it depending on value of Lambda.

References:

```
-----------
812.6 Randomized Algorithms - [Prabhakar Raghavan-Rajeev Motwani] - Page 85 -
Martingales

--------------------------------------------------------------------------------
------------------------------------------------------------
813. (FEATURE) Recursive Lambda Function Growth - corrections for machine
translation - 25 March 2020
--------------------------------------------------------------------------------
------------------------------------------------------------
1.RecursiveLambdaFunctionGrowth.py implementation has been changed to uncomment
```

machine_translation(). But yet because of REST ratelimit
English-Telegu translation is marred by "too many requests" exceptions.
Exception handling and error checks have been added in machine_translation() for NoneType and ratelimit exceptions. Alternative could be local lookup of English-Telegu dictionary from flat file.
2.Self loop edges have been removed because of exceptions in rich club coefficient computation for the textgraph.
3.A news article on Chennai Urban Sprawl expansion has been mapped to textgraph and merit measures are printed to logs testlogs/RecursiveLambdaFunctionGrowth.log.MachineTranslation.25March2020.
Telegu font decoding errors have created a Roman script graph. Proper noun filtering could further fine tune the objective meaning of the news article textgraph.

---
814. (THEORY and FEATURE) Computational Geometric Factorization and Set Partition to Lagrange Four Square Theorem Square Tile Cover Reduction - Randomization - 26 March 2020 - related to 651, 728 and all sections on Set partitions, Shell Turing Machines, Kernel Lifting and Factorization
---
1.DiscreteHyperbolicFactorizationUpperbound_TileSearch_Optimized.py has been changed to handle exceptions in Factor Spark Accumulator
object by stripping off zero-es in zero() function.
2.Streaming_SetPartitionAnalytics.py has been changed for subprocess invocation commandline of factorization. Error handling for size of
arrays has been done.
3.Hardcoded set has been removed and Set to be partitioned is instantiated by range() and maximum size of the set is sent by commandline sys.argv. Size of the set to partition is randomly chosen between 1 and maxsize and factorized to get the sides of the rectangle. Lagrangian Four Square Theorem Square Tile Cover reduction for the tile cover of factorized random set rectangle is printed.
4.This reduction has following phases:
	4.1 Partition a set of random size in SymPy - random histogram
	4.2 Factorize the size of the set of random size by Computational Geometry - sides of the rectangle
	4.3 Reduce each part of the random partition to sum of four squares by Lagrange Four Square Theorem - kernel lifting from 1 dimensional partition to 2 dimensional rectangle partition.
	4.4 Concatenate each of four square tiles per part of the random partition into single set of square tiles.
	4.5 Tile the rectangle by square tiles.
	4.6 Number of square tiles per tile cover of rectangle = 4 * number_of_parts_in_partition

---
815. (THEORY and FEATURE) Waveform Distance of Audios, Acoustic Distance of Strings, Name Similarity, Music Clustering - Python 3.7.5 -
27 March 2020,30 April 2020 - related to 783 and all sections on People Analytics, Compressed Sensing, Intrinsic Merit, Syllable embedding of strings, Hyphenation and Music Pattern Mining
---
1.NeuronRain AstroInfer already has a Music Recommender Systems implementation based on EMD similarities of Mel Frequency Cepstral Coefficients but not based on waveforms.
2.Usual Levenshtein Edit Distance is an alphabet based similarity measure while some problems in Name similarity require sophisticated acoustic matching algorithms (e.g NYSIIS, Soundex, Phonetic Match Rating)
3.This commit implements audio_distance() function for 2 audios (whether music or speech waveforms) which extracts waveform arrays of two audio(s) and compares them by Hausdorff distance and Earth Mover Distance which are more quantitative

distance measures than edit distance.
4.Such a generic waveform phonetic distance between audios of two text strings is an alternative to Levenshtein Edit Distance. But it requires two human voice audio recordings which spells out the text of two strings (as in IVRs)
5.Syllables are language and script independent phonemes and previous audio_distance() by waveform match theoretically captures syllable phonetics better than edit distance and suits name similarity. Audio waveform match might perform better by spelling out Syllable hyphenated texts than plain texts. Syllable boundary word hyphenation has been implemented earlier for People Analytics and requires syllable to human voice audio transforms.
6.AudioToBitMatrix.py has been upgraded to Python 3.7.5. Logs for the change are at
python-src/music_pattern_mining/testlogs/AudioToBitMatrix.log.AudioDistance.27March2020 which compare two similar and different pairs of audio clips of same duration.
7.Intrinsic merit in music is usually associated with creativity/originality of the composer where originality is defined as dissimilarity with works of other composers and similarity between works of self - waveform distance for music approximately captures how dissimilar a music work is from others and common pattern in compositions by self.

---
816. (THEORY and FEATURE) People Analytics - Human Resource Analytics - Chaotic Hidden Markov Model of Tenures/Attrition - implementation -
related to 812 - 31 March 2020

1.Chaotic Hidden Markov Model of Tenures/Attritions based on Verhulst actuarial logistic has been implemented in a new source file SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.py (Python 3.7.5) and SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.py.2.7 (Python 2.7)
2.Similar to other Hidden Markov Models for CRF and TextCompression, this implementation is a Python class accepting states(institutions and organizations),observations(including but not limited to domain-designations-remunerations-cities),emission probabilities(probability of an observation in a state),state transition probabilities(probability of switchovers between orgs) and Lambda (constant for Verhulst Logistic)
3.chaotic_HMM_viterbi() function computes the HMM Viterbi algorithm as usual but with a difference in following line of code:
(probability, state) = max((Viterbi[x-1]
[t]*self.Lambda*self.transition_probabilities[t][y]*(1-
self.transition_probabilities[t][y])*self.emission_probabilities[y]
[self.observations[x]], t) for t in self.states)
which computes the argmax of transition to a state and observation in transitioned state by Verhulst logistic format Lambda*x*(1-x)
4.Probabilities dictionaries have been hardcoded presently in __main__ for invoking Chaotic HMM Viterbi (Lambda=3.7).
5.Example profile of author (overall history of 25 years including academics and work) has been mapped to state transition and observation probabilities (in chronological order - ["Graduation", "AssociateSoftwareEngineer", "MemberTechStaff", "Founder-Architect", "SystemAnalyst", "Specialist", "PostGraduation-ResearchScholar", "Consultant-Architect", "Consultant", "Architect"]) and Viterbi computed is as below which accurately captures the career switches. Still this model considers parallel Tenures as sequential - FOSS initiative of author (Krishna iResearch) from 2003-present has duration of 17 years thus far. Non-major (below 18 years age) graduation in Hindi and schooling have been excluded which increase number of states and observations to 12 from 10 and history to 38 years:
==========
Chaotic HMM Viterbi computation
==========
[{'BaaN-SSAGlobal': 0.0,
      ...

```
  'PSGTech': 1.0,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.3330000000000001,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'SunMicrosystems-Oracle': 0.11088900000000004,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'Krishna-iResearch': 0.036926037000000016,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'Verizon': 0.012296370321000007,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'webMethods-SoftwareAG': 0.004094691316893003},
      ...
 {'BaaN-SSAGlobal': 0.0,
      ...
  'CMI-IIT-IMSc': 0.0013635322085253701,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'GlobalAnalytics': 0.0004540562254389483,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'Clockwork-PiQube': 0.0001512007230711698,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'CloudEnablers': 5.034984078269955e-05,
      ...
  'webMethods-SoftwareAG': 0.0}]

--------------------------------------------------------------------------------
------------------------------------------------------------
817. (THEORY and FEATURE) People Analytics - Talent Analytics - Chaotic Hidden
Markov Model of Parallel Tenures/Attritions and JSON persistence - Weighted
Automata model of Tenures - Tensor Decomposition of EventNet Logical Time -
related to 661,701,812 - 8 April 2020
--------------------------------------------------------------------------------
------------------------------------------------------------
1.SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.py has been updated to remove
hardcoded state transition and emission probabilities (from linkedin profile of
author) and to read them from a JSON persistence
SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.json by json.loads()
2.SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.json has been made
comprehensive to merge and serialize parallel academic and work tenures - there
are multiple parallel academic and work tenures in the example profile making it
increasingly difficult to fathom tenure durations because time is parallelized
amongst multiple tenures. Parallel tenures are metrics for ability of a profile
to multitask. Tensor decomposition of EventNet Causality Logical clock is worth
```

mentioning here which componentizes time - each linear component of time tensor could correspond to a parallel tenure empirically.
3.Parallel tenures are presently simulated by probability distributions - earlier sequential viterbi has been recomputed for parallel tenures which has multiple non-zero probability states per chronological layer of viterbi trellis and largest probability per layer points to the most prominent state at a time point. Optimal choice of probabilities in JSON are reflected in viterbi and following is an example for parallel tenure traversal from logs testlogs/SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.log.8April2020 simulating parallelism:

```
============================
Chaotic HMM Viterbi computation
============================
[{'BaaN-SSAGlobal': 0.0,
      ...
  'Schooling': 0.1,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'DBHPS': 0.036223000000000005,
  'PSGTech': 0.0333,
  'Schooling': 0.0333,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'DBHPS': 0.012062259000000004,
      ...
  'PSGTech': 0.013121057290000003,
  'Schooling': 0.012062259000000004,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.004369312077570002,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'SunMicrosystems-Oracle': 0.001454980921830811,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'Krishna-iResearch': 0.00048450864696966017,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'Verizon': 0.00016134137944089685,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'webMethods-SoftwareAG': 5.3726679353818657e-05},
 {'BaaN-SSAGlobal': 0.0,
  'CMI-IIT-IMSc': 1.7890984224821618e-05,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'GlobalAnalytics': 5.9576977468656e-06,
      ...
  'webMethods-SoftwareAG': 0.0},
```

```
 {'BaaN-SSAGlobal': 0.0,
      ...
  'Clockwork-PiQube': 1.983913349706245e-06,
      ...
  'webMethods-SoftwareAG': 0.0},
 {'BaaN-SSAGlobal': 0.0,
      ...
  'CloudEnablers': 6.606431454521795e-07,
      ...
  'webMethods-SoftwareAG': 0.0}]
```

4. Tenure transitions could also be formalized by weighted automata which has been already implemented in NeuronRain AstroInfer to measure complexity of music notes:
(tenure1, a, tenure2) - transition from state tenure1 to state tenure2 on symbol a of weight 1
(tenure2, 2b, tenure3) - transition from state tenure2 to state tenure3 on symbol b of weight 2
Symbols inducing tenure transitions could be any observation (remuneration,designation,location,...) augmented by weight which simulates importance of observation.
5. Complexity of Chaotic HMM and Weighted Automata for tenure transitions of a profile measures unpredictability and merit which is any of the following - parallel tenures, cyclical tenures, academic-work transitions, individual contributions - (corporate and personal initiatives(FOSS)),....

---
818. (THEORY and FEATURE) Chaotic Pseudorandom Generator (PRG) - Mandelbrot set, Undecidability of Learning Chaotic Bigdata - 13 April 2020 - related to 459,629,740,743,744 and all sections on Pseudorandomness, EVMs, Hardness amplification, Computational learning theory, Chaos
---
1.ChaosAttractor.py has been changed to create Mandelbrot set sequence by recursively iterating complex logistic map in a separate clause and function arg - "Mandelbrot" - to ChaosPRG():
Z <- Z^2 + C (for seed C)
2.From 629.6 earlier, Mandelbrot set is undecidable due to differing Hausdorff dimensions(1 and 2) which is crucial because Mandelbrot sets are universally found amongst most natural processes (biological, economic-markets, weather, geographic massive datasets) - Undecidability implies there is no halting Turing machine which recognizes language of Bigdata Mandelbrot sets dealing a lethal blow to BigData learnability.
3.Pseudorandom Mandelbrot sequence generated by previous map Z: Z^2 + C thus is undecidable and a distinguisher Turing machine A (Pr[A(x)=1]-Pr[A(ChaosPRG(seed))=1] >> epsilon where x is a perfect random string in {0,1}^n) which recognizes such a chaotic sequence does not exist too.
4.Example Mandelbrot sequences can be found in testlogs/ChaosAttractor.Mandelbrot.log.13April2020
5.Psuedorandom paper ballot shuffle simulation of Drone Electronic Voting Machine ballots by Chaotic Mandelbrot PRG thus nullifies distinguishability - but yet aforementioned Ramsey theoretic anomaly - "inevitable emergence of order from chaos" - arises: monochromatic arithmetic progressions are still possible in chaotically shuffled multichromatic sequence of votes(if votes are explicitly colored by candidate indices).

References:

818.1 Chaotic Cryptographic Hash Functions - Merkle-Damgard Construction and Chaotic Network automata - https://ui.adsabs.harvard.edu/abs/2017JPhCS.936a2058M/abstract - "Chaos theory has been used to develop several cryptographic methods relying on the pseudo-random properties extracted from simple nonlinear systems such as cellular

automata ... Following the Merkle and Damgård model of construction, a portion of the message is entered as the initial condition of the network automata, so that the rest parts of messages are iteratively entered to perturb the system ..."
818.2 Merkle-Damgard Construction - initialization vector (IV) and iterated hashing - https://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd_construction

---
819. (THEORY and FEATURE) Chaotic Pseudorandom Generator - 1D Binary Cellular Automaton - Multicolored Space Filling - implementation - 16 April 2020,17 April 2020 - related to 135, 752, 818 and all sections on Monochromatic and Multichromatic Space filling, Chaos, Computational Geometric Planar Point Location Search, Pseudorandomness, Linear Programs/CSP, LSH/Separate Chaining, Circle Packing in NeuronRain Unified Theory Drafts
---
1.New function grow_cellular_automaton() has been defined in ChaosAttractor.py which grows an one dimensional binary cellular automaton.
2.ChaosPRG() function accepts four algorithm options - "Logistic","Lehmer-Palmore","Mandelbrot" and "Cellular Automaton".
3.grow_cellular_automaton() function accepts a binary string of 0s and 1s as initialization vector and a random Lambda fraction.
4.Following simple growth rule has been implemented:
	4.1 Automaton is grown by scanning the two binary digit neighbours of a bit location for four possibilities - 00,10,01,11
	4.2 For each possible neighbourhood, bit location in next generation is updated depending on random value of lambda to 0 or 1
	4.3 This simulates Conway's Birth-Survival-Death Game of Life Cellular Automaton approximately and 2-dimensional tableau of binary strings grown in parallel thus captures space filling complexity of natural processes including random fluid flow, rainfall, randomly strewn solids, ...
	4.4 Cellular Automaton Space Filling Algorithm in 135 , a refinement of LP version in https://sites.google.com/site/kuja27/Analysis%20of%20a%20Randomized%20Space%20Filling%20Algorithm%20and%20its%20Linear%20Program%20Formulation.pdf?attredirects=0 describes a Parallel PRG based 2-dimensional Cellular Automaton which monochromatically fills a 2-D plane in parallel
	4.5 Each generation row of 1D binary Cellular Automaton is a Pseudorandom bit sequence
5.States (colors) of the squares in this implementation are binary - 1 or 0 which could be generalized to arbitrary coloring alphabet by defining/reading new growth rules within grow_cellular_automaton() making it a multichromatic space filling. An example rule for 2-dimensional multicolored cellular automata could be Four color theorem - cell and its 8 neighbours are 4-colorable facets; neighbouring faces avoid same color and for successive generations coloring violations might have to be readjusted - Example:

```
1 2 3
3 4 1
1 2 3
```

6.R correlations of the Cellular Automaton PRG to a stock quote dataset are printed in testlogs/ChaosAttractor.CellularAutomaton.log.16April2020.

---
820. (THEORY and FEATURE) People Analytics - LinkedIn Dataset Analysis - Tenure statistics - 17 April 2020 - related to 817

1.LinkedIn profile parser function parse_profile() in SocialNetworkAnalysis_PeopleAnalytics.py has been updated to compute following tenure statistics for each linkedin text profile for being compliant to Kaggle linkedin CSV dataset (https://www.kaggle.com/killbot/linkedin)
which has these fields. Ethnic and physical feature fields have been excluded:

```
            self.avg_n_pos_per_prev_tenure = 1
            self.avg_pos_len =
float(sum(self.timedeltas))/float(len(self.timedeltas))
            self.avg_prev_tenure_len =
float(sum(self.timedeltas[:-1]))/float(len(self.timedeltas[:-1]))
            self.n_prev_tenures = len(self.timedeltas)-1
            self.tenure_len = self.timedeltas[-1:]
            self.n_followers = self.parse_connections(profile_text)
```
2.Previous statistics compute average number of positions per previous tenure (which has been assumed to be 1), average length of positions(computed from tenure timedeltas histogram), average previous tenure length (computed by neglecting present tenure), number of previous tenures, tenure length (only present tenure), number of followers (connections)
3.Each of these tenure variables could be either analyzed as moving average timeseries (kind of martingale because it captures past history) or observations read from JSON in Weighted Automata - Chaotic Hidden Markov Model of Tenures (SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.py) - JSON file based on these variables could be dynamically written per linkedin profile and Viterbi is computed.
4.Logs for example linkedin profile of author (https://www.linkedin.com/comm/in/srinivasan-kannan-608a671) are at testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.17April2020
5.Video resumes could also be analytics datasources apart from usual social network data in PDF and TXT because of their greater authenticity (Alumni,Quora,Facebook,Twitter,Google,Repositories profile montage Video resumes of author in python-src/image_pattern_mining - ExampleVideo_1.mp4, ExampleVideo_4.mp4, ExampleVideo_2.mp4, ExampleVideo_Facebook_GRAFIT_29April2019.mp4, ExampleVideo_3.mp4, ExampleVideo_GoogleScholar_Search.mp4) .

---

865. (THEORY and FEATURE) NeuronRainApps - NeuronRain usecases - Drone Autonomous Online Shopping Delivery Implementation - MAVSDK + JMAVSIM simulation for PX4 - 1 September 2020 - related to 637,675,713

---

1. OnlineShoppingDelivery Python 3.7.5 implementation in NeuronRainApps/Drones has been changed to import MAVSDK Mission Plan classes and dronecode_sdk imports have been commented (Example: https://github.com/mavlink/MAVSDK-Python/blob/master/examples/mission.py).
2. Following GIS analytics variables are read by socket streaming datasource in Streaming Abstract Generator from a webserver listening on port 64001 (netcat -l 64001) and are parsed as key-value pairs (these variables have also been committed to python-src/NeuronRainApps/Drones/OnlineShoppingDelivery.GISvars.txt) delimited by end marker:

```
      longitude=47.397825620791895
      latitude=8.5450092830163281
      relative_altitude_m=1
      speed_m_s=1
      is_fly_through = True
      gimbal_pitch_deg = nan
      gimbal_yaw_deg = nan
      camera_action = none
      loiter_time_s = nan
      camera_photo_interval_s = nan
      end
```

3. In this commit only first four variables for navigation are being read from socket stream while rest are assumed to have default values.
4. Code changes for MissionPlan specific to MAVSDK mission-upload, start-mission and arm-takeoff-take_photo have been committed. To simulate a hamiltonian cycle, drone.mission.set_return_to_launch_after_mission(True) has been invoked.
5. awaits for mission progress and observe_is_in_air have been commented.

6. GIS navigation analytics variables read over socket stream are filtered for convex-hull boundaries and mission items are appended accordingly to Mission plan. Every mission item is a delivery point.
7. JMAVSIM and PX4 SITL Logs, Netcat logs for streaming analytics variables and MP4 video of the simulated flight of a drone have been committed as hereunder:

python-src/NeuronRainApps/Drones/testlogs/OnlineShoppingDelivery_Drone_JMAVSIM_Flight_Simulation.log1.1September2020

python-src/NeuronRainApps/Drones/testlogs/OnlineShoppingDelivery_Drone_JMAVSIM_Flight_Simulation.log2.1September2020

python-src/NeuronRainApps/Drones/testlogs/OnlineShoppingDelivery_PX4_Drone_JMAVSIM_Flight_Simulation2.mp4
8. Video
python-src/NeuronRainApps/Drones/testlogs/OnlineShoppingDelivery_PX4_Drone_JMAVSIM_Flight_Simulation2.mp4 shows the imaginary multirotor drone in JMAVSIM PX4 SITL making a vertical takeoff, drawing an aerial cycle based on longitude-latitude-altitude of mission item delivery points before comeback to origin by vertical landing.

---
867. (THEORY and FEATURE) OpenCV Image Contours and Segmentation for Drone Obstacle Avoidance and Motion Planning - related to 866 and all sections on Large Scale Visual Recognition, GIS analytics and Drone Navigation - 12 September 2020

1.This code change implements a new function image_segmentation_contours() in ImageGraph_Keras_Theano.py which is exactly similar to contouring code in handwriting_recognition() (DeepLearning_ConvolutionNetwork_BackPropagation.py) for topological handwriting recognition (Product Homotopy) but without distance computation for two imagefiles and finds contour interpolation polynomials for single image.
2.image_segmentation_contours() has been implemented as a subroutine which is invoked by image_segmentation() and could be used by any other NeuronRain computer vision and Large Scale Visual Recognition analytics code.
3.image_segmentation() earlier had only a computational geometric segmentation of an image for finding component segments and their centroids (GRANA), Delaunay Triangulation and Facegraphs.
4.A-Star motion planning implementation in NeuronRain (Section 866 of NeuronRain Theory Drafts) assumes a graph adjacency matrix and heuristic array signifying estimated cost of obstacle per vertex.
5.Contour detection for GIS imagery is a tool for obstacle avoidance in drone navigation and motion planning. For example, extracting contour points and its interpolated polynomials from a Road network GIS imagery of a terrain is useful for drawing a Contour graph vertices of which are contour points connected by contour polynomials. Adjacency matrix of such a contour graph could be used in A-Star implementation for Drone navigation.
6. It is worth observing that 3D-navigation of drones on the edges of road (rail or other transport) network graphs (assuming sufficient vehicle clearance heights greater than grade separators and interchanges for drone altitude) is always obstacle-free because of zero collision (on the contrary driverless vehicles on 2D road networks have non-trivial obstacle avoidance challenge - an optional obstacle avoidance could be to do a VerticalTakeOffLanding (VTOL) for circumventing every obstacle adding a third dimension to surface transport vehicle).
7. An example Road network map from Google Maps has been contour-recognized and following logs and image files illustrate the contour polynomials inferred from road network - by both image_segmentation() and image_segmentation_contours() :
testlogs/ImageGraph_Keras_Theano.log.ContourSegmentation.26September2020
testlogs/RemoteSensingGIS/Google_Maps_SouthernIndia_RoadMap-000.jpg
testlogs/RemoteSensingGIS/Google_Maps_SouthernIndia_RoadMap-

000_contours_segmented.png
testlogs/RemoteSensingGIS/Google_Maps_SouthernIndia_RoadMap-000_segmented.jpg
testlogs/RemoteSensingGIS/Google_Maps_SouthernIndia_RoadMap.pdf
testlogs/VoronoiFaceGraph_GoogleMaps_RoadMap_1_26September2020.png

---

868. (THEORY and FEATURE) OpenCV Image Contours and Segmentation for Medical Imageing - Magnetic Resonance Imageing (MRI and fMRI) - related to 656, 681, 867, 815 and all sections on Large Scale Visual Recognition, Medical Imageing and Intrinsic Merit of Music - 26 September 2020

1.Functional MRI datasets (OpenNeuro - https://openneuro.org/public/datasets - earlier OpenFMRI) contain huge volumes of experimental functional MRI data for analyzing connectomes in brain and impact of music on humans (e.g https://openneuro.org/datasets/ds000171/versions/00001)
2.Machine Learning has been extensively applied to analyze fMRI data - [Frontiers in Psychology - https://www.researchgate.net/publication/51563375_Functional_Data_Analysis_in_Brain_Imaging_Studies] - by means of which merit of music waveforms could be quantitatively compared for music clustering. NeuronRain already implements automata theoretic and mel-frequency merit measures for music.
3.This code change adds a clause for MRI in medical_imageing() in ImageGraph_Keras_Theano.py and invokes medical_segmentation_contours() to find contours of an example MRI image of brain for different activities.
4.Actual MRI and contour segmented images are at:

testlogs/ImageGraph_Keras_Theano.MedicalImageing.MRIContours.26September2020.png
testlogs/medical_imageing/MRI_fpsyg-01-00035-000.jpg
5.Logs for 867 - testlogs/ImageGraph_Keras_Theano.log.ContourSegmentation.26September2020 - have computation of contours for example MRI medical image (testlogs/medical_imageing/MRI_fpsyg-01-00035-000.jpg) as well.

---

869. (THEORY and FEATURE) Drone Obstacle Avoidance by 3D Navigation on Airspace defined by Road Geometry - related to 582, 867 and all sections on Motion Planning, GIS Analytics, Logical clocks, Large Scale Visuals Recognition, Drone Swarms - 2 October 2020, 6 October 2020

---

1. 3D Navigation of Drones along road network graph airspace (which is theoretically free of natural obstacles) mentioned in 867 requires a webservice which churns out array of GPS ordinates of a road geometry from origin to destination.
2. Google Roads webservice - https://developers.google.com/maps/documentation/roads/overview - returns array of snapped GPS ordinate points best fit to a road geometric course between origin and destination.
3. Previous array of GPS ordinates could be looped to create mission items which can be populated to a mission plan and uploaded which is theoretically an obstacle free mission sticking to airspace strip along the road geometry.
4. Obstacle avoidance in 3D (4D including time reckoning relativity for Drone Swarms) thus reduces to finding optimal altitude along the airspace on road geometry
5. Road geometry airspace poses mostly man-made obstacles (flyovers etc.,) and drone altitude must be recomputed for each obstacle (requires interchange/grade-separators structure recognition and their height inference by OpenCV contouring and sensors) or a maximum altitude for drone mission items is set which is greater than all possible man-made structures on a road strip.
6. Groups of Drones (Drone Swarms) and Obstacle avoidance amongst them has

increasing applications in combat and tactical ops.

References:

869.1 Google Roads Geometry REST API - https://developers.google.com/maps/documentation/roads/snap - returns an array of GPS points and place id(s) best fit to an approximate road trajectory between origin and destination
869.2 Lane detection for autonomous surface transport vehicles in OpenCV - https://campushippo.com/lessons/detect-highway-lane-lines-with-opencv-and-python-21438a3e2
869.3 Drone Swarms - DARPA - https://www.theverge.com/2019/8/9/20799148/darpa-drones-robots-swarm-military-test
869.4 SwarmLab - Obstacle Avoidance Simulation of Drone Swarms in MATLAB - Olfeti-Saber and Vaserhelyi Obstacle avoidance algorithms - https://arxiv.org/pdf/2005.02769.pdf
869.5 Decentralized 3D Collision Avoidance for Multiple UAVs in Outdoor Environment - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6308403/

---
872. (THEORY and FEATURE) Fibonaccian Search - alternative to binary search - related to all sections on Computational Geometric Planar Point Location Factorization, Geometric Search of BigData, Arithmetic Progressions and Ramsey coloring - 31 October 2020
---
1. Binary Search could be costly in some architectures because of division by 2. Fibonaccian search based on Fibonacci series avoids division and depends only on addition and subtraction. Note: Fibonaccian search is different from Fibonacci search to locate maximum of a unimodal function.
2. Binary search is central to Computational Geometric Factorization to locate a factor point within a segment/interval/pixel-array-polygon which is a sorted arithmetic progression of ordinate products.
3. Fibonaccian search optimization to replace Binary search in planar point location could significantly reduce constant involved in O(logN) search.
4. Both Binary search and Fibonaccian search are O(logN) while the search tree of the latter makes the difference - Left subtree of Fibonacci search is 1.618 (Golden ratio) times more skewed than Right subtree.
5. In terms of runtime efficiency Fibonaccian search lies between Binary Search and Interpolation search (which is O(loglogN) and assumes elements are within a known range)

References:

872.1 Fibonaccian search - Section 6.2.1 - Searching an ordered table - Page 418 - Algorithm F - The Art of Computer Programming - Volume 3 - [Donald Knuth]

---
873. (THEORY and FEATURE) Fibonaccian Search - Python 3.9.0 upgrade for Factorization - related to 872 - 5 November 2020
---
1. Fibonaccian search python implementation has been upgraded to Python 3.9.0 by autopep8 and 2to3-3.9 tools for potential invocation from Computational geometric factorization which is already Python 3 compatible.
2. Python 2.7 version of Fibonaccian search has been separately committed.
3. After this upgrade NeuronRain supports all three versions of Python 3 - Python 3.7.5, Python 3.8.5 and Python 3.9.0

---
874. (THEORY and FEATURE) People Analytics - People Profiles as Tensors - related to 752 and all sections on Embedding in vectorspace, Social Network Analytics, Psephology/Election analytics, Majority Voting, Intrinsic merit of People profiles - 13 November 2020

1.python-src/SocialNetworkAnalysis_PeopleAnalytics.py has been changed to optionally choose between REST API calls to PIPL.com and reading a json file of people profile from any datasource including PIPL.com by boolean parameter loadfromjson in function pipldotcom_analytics() which reads People data in JSON from testlogs/SocialNetworkAnalysis_PeopleAnalytics.json
2.JSON objects are multidimensional which could be construed as points on a vectorspace. TensorFlow supports reading JSON format of Tensors as Protobufs (Read JSON string and return a Protobuf Tensor - https://www.tensorflow.org/api_docs/python/tf/io/decode_json_example).
3.Mapping people profiles onto Tensor points on vectorspace opens up possibility of whole gamut of Computational geometric People analytics - for example closest pair and clusters of people could be found and profiles could be iteratively ranked to get similarities.
4.Clustering of people profiles is crucial in analyzing voting patterns.
5.On a related note, NeuronRain implements a Set partition Electronic Voting Machine and every candidate bucket is a cluster of voters - Distance similarity (or Swing between consecutive polls) between any two Set Partition EVMs is measurable by Earth mover distance (Wasserstein distance).

---
875. (THEORY and FEATURE) People Analytics - Alphabet Vectorspace Embedding Distance implementation - related to 752,874 and all sections on Embedding in vectorspace, Compressed Sensing, Social Network Analytics, Meaning Representation, Intrinsic Merit of People Profiles/HR Analytics - 15 November 2020

---
1. CompressedSensing.py has been changed to implement a new function alphabet_vectorspace_embedding_distance(self, embedding1, embedding2) which finds the euclidean distance between two string vector points (1-dimensional Tensor) embedded in alphabet space.
2. It uses the ord() builtin to compute ordinal values of the ascii or unicode strings.
3. alphabet_vectorspace_embedding_distance() is invoked from python-src/SocialNetworkAnalysis_PeopleAnalytics.py to compare two different name spellings in English of a Sanskrit name.
4. Strings of different lengths are left justified and padded by "#" fill before distance computation.
5. Distance computed is normalized - divided by length of the strings compared.
6. Logs python-src/testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.15November2020 show distance similarities amongst different name spellings in English of Sanskrit name which can be contrasted against Phonetic match rating and NYSIS Flight manifest name similarity algorithms as below:
embedding1: ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 's', '#', '#', '#']
embedding2: ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 'a', 's', 'a', 'n']
Alphabet Vectorspace Embedding Distance between ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 's', '#', '#', '#'] and ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 'a', 's', 'a', 'n'] : 127.251719045363
Normalized Alphabet Vectorspace Embedding Distance between ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 's', '#', '#', '#'] and ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 'a', 's', 'a', 'n'] : 10.604309920446918
leftjustified: ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 's', '#']
embedding1: ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 's', '#']
embedding2: ['S', 'r', 'i', 'n', 'i', 'v', 'a', 's', 'a', 'n']
Alphabet Vectorspace Embedding Distance between ['S', 'h', 'r', 'i', 'n', 'i',

'v', 'a', 's', '#'] and ['S', 'r', 'i', 'n', 'i', 'v', 'a', 's', 'a', 'n'] : 84.34453153583817
Normalized Alphabet Vectorspace Embedding Distance between ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 's', '#'] and ['S', 'r', 'i', 'n', 'i', 'v', 'a', 's', 'a', 'n'] : 8.434453153583817
leftjustified: ['S', 'r', 'i', 'n', 'i', 'v', 'a', 's', 'a', 'n', '#', '#']
embedding1: ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 'a', 's', 'a', 'n']
embedding2: ['S', 'r', 'i', 'n', 'i', 'v', 'a', 's', 'a', 'n', '#', '#']
Alphabet Vectorspace Embedding Distance between ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 'a', 's', 'a', 'n'] and ['S', 'r', 'i', 'n', 'i', 'v', 'a', 's', 'a', 'n', '#', '#'] : 103.24243313676794
Normalized Alphabet Vectorspace Embedding Distance between ['S', 'h', 'r', 'i', 'n', 'i', 'v', 'a', 'a', 's', 'a', 'n'] and ['S', 'r', 'i', 'n', 'i', 'v', 'a', 's', 'a', 'n', '#', '#'] : 8.603536094730662
7.Invocation of Recursive Lambda Function Growth Meaning Representation of a Professional Profile and Email context name parsing have been uncommented. Because of unfiltered proper nouns of organizations, few lambda functions are misleading.
8.Two phonetically different name spellings "Rasa" and "Raja" of same name (in Tamizh and Sanskrit respectively) are compared for euclidean distance similarity.
9.Syllable Hyphenation of name strings adds one more dimension and would create a 2-dimensional tensor embedding and distance between two 2D tensors could be more accurate e.g:
"Srinivasan" - [[S,r,i,n,i],[v,a,s,a,n]]
"Shrinivas" - [[S,h,r,i,n,i],[v,a,s]]
"Shrinivaasan" - [[S,h,r,i,n,i],[v,a,a,s,a,n]]
10.For arbitrary dimensional people data as JSON tensors which include name,work experience,connections,academics etc., distance computation between Person1 and Person2 requires expanding dimensions of smaller tensor and computing the norm of Person1 - Person2 (Stackoverflow question - https://stackoverflow.com/questions/55196194/how-to-calculate-the-euclidean-distance-between-a-2-d-tensor-and-a-3-d-tensor)

---
876. (THEORY and FEATURE) People Analytics - People profiles as Tensors - Chaotic Hidden Markov Model of Tenure Transitions - TensorFlow integration for conversion of People profile data to Tensors - 17 November 2020 - related to 365,400,572,875 and all sections on Intrinsic merit of People, BKS Conjecture, Condorcet Jury Theorem, Margulis-Ruzzo Threshold, Berry-Esseen Central Limit Theorem and Stability of Interview formalized by LTF/TQBF/PTF

1.This commit invokes TensorFlow convert_to_tensor() function for converting any People profile data encoded as any Python object including JSON to a Tensor.
2.python-src/SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.py which models tenure transitions of People profiles by Chaotic Hidden Markov Model Viterbi Trellis,now imports TensorFlow and TensorFlow I/O.
3.SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.json Tenure Transition is read and the fields are converted to Tensors by convert_to_json() TensorFlow API e.g tenure organizations - academic and work - are encoded as TF tensor below:
piplprofiletensor - states: tf.Tensor(
[b'Schooling' b'DBHPS' b'PSGTech' b'BaaN-SSAGlobal'
b'SunMicrosystems-Oracle' b'Krishna-iResearch' b'Verizon'
b'webMethods-SoftwareAG' b'CMI-IIT-IMSc' b'GlobalAnalytics'
b'Clockwork-PiQube' b'CloudEnablers'], shape=(12,), dtype=string)
4.TensorFlow function decode_as_json() might also be invoked but there are occassional JSON parsing errors.
5.Objective of NeuronRain People Analytics is to automate, do away with or minimize human errors in talent recruitment through machine learning models. Stability and Noise Sensitivity criterion of an interview TQBF/LTF/PTF derived earlier theoretically stipulates when a talent recruitment system by Question-Answering is better than majority voting and thus has enormous HR analytics

ramifications. Goodness of majority voting - when group decision converges to good outcome for p-biased voters - is limited by Margulis-Ruzzo Threshold/Condorcet-Black-Ladha Jury Theorems for p-biased boolean functions. Combining the two, Stability criterion for interview TQBF (which is in IP=PSPACE) outclassing majority decision and Margulis-Ruzzo/CJT threshold for majority decision converging to 100% are thus inversely related - if Questions-Answers are improperly weighted (a <= 1.2056205488/n) for every possible LTF, Interview process is subverted and becomes unstable making BKS Conjecture False.

---

877. (FEATURE) Merit of Large Scale Visuals (LSVR) - TensorFlow Keras Backend - 18 November 2020 - related to 868 and all sections on Video Merit, Large Scale Visual Recognition

---

This commit changes KERAS_BACKEND to TensorFlow for Video predictions by Keras and ImageNet. By this both Theano and TensorFlow are supported by NeuronRain Large Scale Visual Analytics. An example GoogleScholar and Google Search for query "NeuronRain Srinivasan Kannan" is analyzed by Keras-TensorFlow combine. TensorFlow has inbuilt support for tensors and Logical Clock EventNet Tensors, Video EventNet Tensors, People Profile Tensors and any multidimensional data could be translated to TensorFlow tf.Tensor() object. TensorFlow supports Keras models.

---

878. (FEATURE) EventNet Python Parser - Events as Tensorflow Tensors - 20 November 2020 - related to 701 and all sections on EventNet Logical Clock, Video EventNet Tensor Products algorithm for Merit of Large Scale Visuals and Category Theory

---

This commit refactors EventNet/EventNet.py to define EventNet_Tensor_Parser() function which parses the EventNet Vertices and Edges text files and instantiates TensorFlow Tensor objects for each intraevent conversations (edges amongst actors in an event) and a holistic EventNet TensorFlow Tensor for interevent causality. EventNet abstraction helps analyzing Videos - each Frame is an Event, actor objects within a Frame event category interact by morphisms (intraevent) and Causality between Frames (interevent) is defined by edges or Functors between two Frame Event categories. More than anything else, EventNet conceptualizes a multidimensional logical time Tensor and postulates that time tensor could be decomposed to components. EventNet example has been redrawn by GraphViz and testlogs/EventNet.Tensor.log.20November2020 capture the EventNet Tensor instantiated as TensorFlow tf.Tensor() objects.

1123. (THEORY and FEATURE) People Analytics - Alphabet Vectorspace Embedding - Earth Mover Distance between String Syllable Tensors - related to 752,874,875 and all sections on Embedding in vectorspace, Compressed Sensing, Social Network Analytics, Meaning Representation, Intrinsic Merit of People Profiles/HR Analytics, Set Partition and Histogram analytics - 27 November 2020

1.This commit augments CompressedSensing.py - alphabet_vectospace_embedding_distance() function to optionally compute syllable vector earth mover distance between two phonetic syllable hyphenated string 2D tensors (encoded by ordinal integers) by syllable_tensor_distance boolean flag
2.Lot of options were tried for computing distance norm between two 2D tensors having arbitrary number of rows and columns of unequal lengths - TensorFlow, sklearn, SciPy directed_hausdorff, wasserstein_distance and cdist, OpenCV2 EMD - and OpenCV2 EMD has been chosen because of its implicit dimension expansion, support for unequal rows and columns between two 2D tensors (which is very much

necessary for comparing two strings of unequal number of syllables of arbitray length) and ability to measure syllables as histograms - syllables partition a string to histogram buckets and distance between two strings is the amount of work necessary to move one string syllable histogram to another (2D analogue of Edit Distance) which is captured by Earth Mover Distance of OpenCV.
3.Function tosig() translates a syllable hyphenation of string to OpenCV.EMD() format.
4.Logs at
python-src/testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.27November2020
compute EMD between following string syllable hyphenations:
 csensing.alphabet_vectorspace_embedding_distance([['n','o'], ['m','i'], ['n','a','l']], [['s','e'], ['m','i'],['n','a','l']],True)
 csensing.alphabet_vectorspace_embedding_distance([['t','e','r'], ['m','i'], ['n','a','l']], [['s','e'], ['m','i'],['n','a','l']],True)
 csensing.alphabet_vectorspace_embedding_distance([['S','h','r','i'],['n','i'], ['v','a','a'],['s','a','n']],[['S','r','i'],['n','i'], ['v','a','s','a','n']],True)
5.Following are the respective EMDs computed for string syllable hyphenations(encoded by ordinals):
ordembedding1: [[110, 111], [109, 105], [110, 97, 108]]
ordembedding2: [[115, 101], [109, 105], [110, 97, 108]]
Earth Mover Distance between two String syllable tensors: (0.006666666828095913, None, array([[110.,   0.,   0.,   0.,   0.,   0.,   0.],
       [5., 101.,   0.,   0.,   0.,   0.,   0.],
       [0.,   0., 109.,   0.,   0.,   0.,   0.],
       [0.,   0.,   0., 105.,   0.,   0.,   0.],
       [0.,   0.,   0.,   0., 110.,   0.,   0.],
       [0.,   0.,   0.,   0.,   0., 97.,   0.],
       [0.,   0.,   0.,   0.,   0.,   0., 108.]], dtype=float32))
ordembedding1: [[116, 101, 114], [109, 105], [110, 97, 108]]
ordembedding2: [[115, 101], [109, 105], [110, 97, 108]]
Earth Mover Distance between two String syllable tensors: (0.0, None, array([[115.,   0.,   0.,   0.,   0.,   0.,   0.],
       [0., 101.,   0.,   0.,   0.,   0.,   0.],
       [0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [0.,   0., 109.,   0.,   0.,   0.,   0.],
       [0.,   0.,   0., 105.,   0.,   0.,   0.],
       [0.,   0.,   0.,   0., 110.,   0.,   0.],
       [0.,   0.,   0.,   0.,   0., 97.,   0.],
       [0.,   0.,   0.,   0.,   0.,   0., 108.]], dtype=float32))
ordembedding1: [[83, 104, 114, 105], [110, 105], [118, 97, 97], [115, 97, 110]]
ordembedding2: [[83, 114, 105], [110, 105], [118, 97, 115, 97, 110]]
Earth Mover Distance between two String syllable tensors: (0.3320339024066925, None, array([[83.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [0., 104.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [0.,   9., 105.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 104.],
       [0.,   0.,   0., 110.,   0.,   0.,   0.,   0.,   0.,   0.],
       [0.,   0.,   0.,   0., 105.,   0.,   0.,   0.,   0.,   0.],
       [0.,   0.,   0.,   0.,   0., 118.,   0.,   0.,   0.,   0.],
       [0.,   0.,   0.,   0.,   0.,   0., 97.,   0.,   0.,   0.],
       [0.,   0.,   0.,   0.,   0.,   0.,   0., 97.,   0.,   0.],
       [0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [0.,   0.,   0.,   0.,   0.,   0.,   0., 11.,   0.,   0.],
       [0.,   0.,   0.,   0.,   0.,   0.,   0.,   7., 97.,   6.]],
      dtype=float32))

1124. (THEORY and FEATURE) Graph Search and Analytics - NetworkX Graph Edit Distance - related to 753, 829, 1123 and all sections on
TextGraph algorithms for Meaning Representation, Social Network Analytics, Distance and Similarity measures for Text/Audio/Video/People,
Frequent Subgraph mining, Inexact Graph Matching, Program Analysis - 9 December

2020

1.GraphMining_GSpan Python implementation has been upgraded to Python 3.7 by autopep8 and 2to3-3.7 utilities.
2.New function GraphEditDistance() has been defined in GSpan class which takes two NetworkX Graph objects as arguments and computes
 exact Graph Edit Distance (NP-Hard) by graph_edit_distance() and approximations by optimize_edit_paths() and optimize_graph_edit_distance()
NetworkX APIs.
3.Logs in
python-src/testlogs/GraphMining_GSpan.GraphEditDistance.log.9December2020
capture the GSpan Graph mining for some example TextGraphs (Recursive Gloss Overlap algorithm) and Graph Edit Distance for two example numeric labelled graphs.
4.GraphEditDistance() could be invoked from any Graph analytics implementation in NeuronRain for Similar Graph clustering viz., Social network graphs, FaceGraphs (Delaunay triangles and Voronoi facets) of Segmented images(Pattern recognition, GIS, Urban Sprawls), TextGraphs, Economic networks, Program Slice Graphs, FTrace kernel callgraphs, Userspace Valgrind/KCacheGrind/Callgrind callgraphs, Control Flow Graphs et al.
5.Graph Edit Distance for Audios (Speech) assumes a hypothetical AudioNet similar to ImageNet dataset or Speech-to-Text engine for eliciting graph structure in speech though Music as a graph is vague to define, but NeuronRain music pattern mining learns a weighted automata from musical clips which are directed graphs of state transitions labelled by heptatonic western classical notes (CDEFGABC).
6.Classical Indian music has 72 fundamental notes patterns and many derivatives from them.

References:

1124.1 An Exact Graph Edit Distance algorithm for Pattern Recognition Problems - https://hal.archives-ouvertes.fr/hal-01168816

1125. (THEORY and FEATURE) Software Analytics - Cyclomatic Complexity Program Analyzer - Graph Edit Distance - related to 581,1124 and all
sections on Program Analysis, Software Analytics, Graph Edit Distance, Inexact Graph Matching - 15 December 2020

---
1. Cyclomatic Complexity Program Analyzer Spark implementation
python-src/software_analytics/CyclomaticComplexitySparkMapReducer.py has been refactored and new function cyclomatic_complexity() has been defined for Cyclomatic Analysis of FTrace callgraph by GraphFrames.
2. Function graph_mining() now invokes GraphEditDistance() from GraphMining_GSpan for each pair of callgraph NetworkX objects instantiated from DOT files in addition to GSPAN algorithm for frequent subgraph mining.
3. Finding code similarity by Graph Edit Distance between callgraphs is dynamic program analysis (Inexact matching) which is more comprehensive profiling than Graph Isomorphism (Exact Matching)

1126. (THEORY and FEATURE) People Analytics - Word Sense Disambiguation changed to NeuronRain implementation - related to 876,1123 and
all sections on Embedding in vectorspace, Compressed Sensing, Social Network Analytics, Meaning Representation, Intrinsic Merit of People Profiles/HR Analytics, Set Partition and Histogram analytics - 16 December 2020

---
1. Recursive Lambda Function Growth Meaning Representation was inferring some

misleading lambda functions related to the proper noun
dictionary-filtered profile contents - Word "Architect" was incorrectly
disambiguated to civil engineering and produced lemmas - "slab",
"home_plate", "base_ball" et al.
2. This was perhaps because of deficiencies of NLTK lesk implementation.
Changing to PyWSD significantly slows down and there are WordNet
errors within PyWSD. This has necessitated reverting to NeuronRain Lesk Word
Sense Disambiguation function best_matching_synset() and
both NLTK and PyWSD disambiguation have been disabled.
3. NeuronRain WSD best_matching_synset() has extracted quite accurate synset
lemmas which have been used to create TextGraph and Graph Tensor Neural Network
Lambda functions are grown from Random walks and Cycles on this new TextGraph -
TextGraph for best_matching_synset() is much different from those of NLTK/PyWSD
Lesk functions.
4. New TextGraph has been written to DOT file RecursiveLambdaFunctionGrowth.dot
5. Logs testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.16December2020 print
following Dense subgraph classes (which give high weightage to durations) and
intrinsic merit parameters while Betweenness,Closeness,Degree,PageRank
centralities rank the profile highly in class "Software":
...
==========
Unsupervised Classification based on top percentile Core numbers of the
definition graph(subgraph of WordNet)
==========
('This document belongs to class:', 'April', ',core number=', 6)
('This document belongs to class:', 'March', ',core number=', 6)
('This document belongs to class:', 'May', ',core number=', 6)
('This document belongs to class:', 'month', ',core number=', 6)
('This document belongs to class:', 'preceding', ',core number=', 6)
('This document belongs to class:', 'following', ',core number=', 6)
('This document belongs to class:', 'architect', ',core number=', 4)
('This document belongs to class:', 'search', ',core number=', 4)
('This document belongs to class:', 'Hindi', ',core number=', 4)
('This document belongs to class:', 'institute', ',core number=', 4)
('This document belongs to class:', 'education', ',core number=', 4)
('This document belongs to class:', 'experience', ',core number=', 4)
('This document belongs to class:', 'software', ',core number=', 4)
('This document belongs to class:', 'science', ',core number=', 4)
('This document belongs to class:', 'activities', ',core number=', 4)
('This document belongs to class:', 'Tamil', ',core number=', 4)
('This document belongs to class:', 'skill', ',core number=', 4)
('This document belongs to class:', 'knowledge', ',core number=', 4)
('This document belongs to class:', 'something', ',core number=', 4)
('This document belongs to class:', 'India', ',core number=', 4)
('This document belongs to class:', 'written', ',core number=', 4)
('This document belongs to class:', 'language', ',core number=', 4)
('This document belongs to class:', 'spoken', ',core number=', 4)
('This document belongs to class:', 'someone', ',core number=', 4)
...
intrinsic_merit_dict: {'graph_tensor_neuron_network_intrinsic_merit':
1011.984847256428,
'maximum_per_random_walk_graph_tensor_neuron_network_intrinsic_merit':
('(experience(content,(education(skill,(cognition(ability,(cognition(ability,
(psychological_feature(cognition,(activity(event,(act(education,
(skill(education,(act(activity,(psychological_feature(institute,
(abstraction(event,(organization(group,(social_group(institute,
(organization(association,(abstraction(association,(group(science,
(abstraction(knowledge_domain,(cognition(abstraction,
(psychological_feature(content,
(discipline(social_group,psychological_feature))))))))))))))))))))))))))))
))))))', 16.89555722055722), 'korner_entropy': 0.7184074850146093, 'density':
0.007855421686746989, 'bose_einstein_intrinsic_fitness': 0.8612343424751857,

'recursive_gloss_overlap_intrinsic_merit': 856492032.0, 'empath_sentiment': (0.0, 0.0, 1.8548942433184686e-296)}

1127. (THEORY and FEATURE) People Analytics - HR and Talent Analytics - Domain Specific Dictionary - LinkedIn Dataset analytics in Spark 3.0.1+ Hadoop 3.2 - related to 613,727,812,876 and all sections on People Analytics, Intrinsic Merit Versus Fame Equilibrium, Novelty Detection, BKS Conjecture, Stability of Interview LTF/PTF - 19 December 2020

1.NeuronRain People Analytics is mainly about automating the process of recruiting talent by minimizing or obviating human error and intervention.
2.Theoretically if intrinsic merit lowerbounds fame, automatic recruitment is feasible which has been analyzed in earlier relevant sections. In this commit, profile parser function has been updated to include a domain specific dictionary argument which contains fields specific to a work domain - IT, Manufacturing, Academics etc.,
3.For Information Technology domain, example dictionary has two fields (values) - "name" (Information Technology) and "opensource_sloc" (Number of Source Lines of Code contributed to OpenSource products). Domain of IT is quite different from other industries in the sense work experience and technical wherewithal is quantifiable by lines of code and arbitrary number of clones can be derived from digital content which is impossible in other disciplines.
4.Technical Talent in Information Technology domain is measurable quantitatively by number of lines of code written which is used in various opensource codebase analyzers (e.g OpenHub COCOMO analyzer) and software complexity metrics (e.g Function Point Analysis).
5.Total experience of a profile is weighted by opensource_sloc value (which is at present just a linear addition function but could be any other) for least energy intrinsic merit of a profile which is the single largest crucial parameter more important than mere tenure duration for automatic recruitment in Information Technology domain and more intrinsic than Majority Voting (which suffers from limitations of Condorcet Jury Theorem) and manual processes of Interview (which is subject to BKS Conjecture and human error).
6.If Novelty Detection measures (e.g Patents) and SLOC measures per proprietary and opensource products of organizations are accumulated and weighted it could further augment the previous merit statistic.
7.Spark Version has been upgraded to 3.0.1 (based on Hadoop 3.2) for LinkedIn Dataset Analytics.

1128. (THEORY and FEATURE) People Analytics - HR and Talent Analytics - Chaotic Hidden Markov Model - Career Transition Score of a Profile - Weighted Automaton of Career Transition - related to 753, 1127 and all sections on People Analytics, Intrinsic Merit Versus Fame Equilibrium, BKS Conjecture, Stability of Interview LTF/PTF - 23 December 2020
---
1. This commit implements an important and most fundamental merit measure for People profiles - Career Transition Score - which quantifies and grades the career tenure switchovers of a Professional profile by traversing designations/paygrades in a Career Transition Weighted Automaton State Machine.
2. Section 753 attributes following qualities to career transition ranked by incentivizing choices made:
" ...
(*) transition from industry to academics is considered nobler than industry-to-industry.
(*) transition from academics to industry is majority choice
(*) transition from low remuneration to high is a majority choice
(*) transition from high remuneration to low is rare
(*) transition from employment to entrepreneur is rare and considered less

risk-averse
      (*) transition from low designation to high is majority choice
      ... "
3.SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.py defines a new function career_transition_analytics() which traverses the Observations
of the Chaotic Hidden Markov Model,classifies them into states and looksup the state transitions in a predefined Weighted Automaton defined by a dictionary:
      self.career_statemachine={'academic-academic':2,'work-work':1,'startup-startup':2,'academic-work':1,'work-academic':2,'academic-start up':3,'startup-academic':3,'work-startup':3,'startup-work':1,'academic-business':5,'business-business':6}
4.Previous career state machine presently has 4 classes of designations for states - academic, work, startup (which includes own initiatives of a profile) and business (corporate designations) - and transitions among them are weighted by integers in ascending order of importance and risk aversion. E.g Transition work-work is weighted as 1 which is considered secure while business-business is weighted as 6 because of high risk.
5.Designations (Observations in HMM) are classified into 4 classes above - academic,startup,business,work - and weighted by increased paygrades/power/independence/authority/experience/totempole:

self.designations["academic"]={"Schooling":1,"Graduation":2,"PostGraduation":3,"PostGraduation-ResearchScholar":4}
        self.designations["startup"]={"Founder-Architect":7}
        self.designations["business"]={'CEO':10,'CTO':9,'CFO':8}
        self.designations["work"]={"AssociateSoftwareEngineer":1, "MemberTechStaff":2, "SystemAnalyst":3, "Specialist":4, "Consultant-Architect":6, "Consultant":5, "Architect":6}
6.More predefined designations, classes of designations and state transitions could be added to previous dictionaries which might be optionally read from a persisted NoSQL datastore.
7.Observations of Career Chaotic Hidden Markov Model are traversed and weights per state transition and corresponding observation are summed to get a Career Transition Score of a profile which intrinsically rates merit of a profile based on predefined weights.
8.Industry has been divided into startup (for MSMEs, private initiatives) and business (for management profiles of Corporates, Bellwether, Blue Chips, Big Fishes) for weighting the risk.

1130. (THEORY and FEATURE) People Analytics - HR and Talent Analytics - Chaotic Hidden Markov Model - Career Polynomial of a Profile and
Inner Product Space Distance Similarity between 2 people profiles - related to 753,1127,1128,1129 and all sections on People Analytics, Intrinsic Merit Versus Fame Equilibrium, BKS Conjecture, Stability of Interview LTF/PTF - 30 December 2020

1.This commit implements 3 new functions in python-src/SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.py - career_polynomial_inner_product_distance(),normalize(),second() - and polynomial fit of weighted HMM observations (designations) in career_transition_analytics()
2.career_polynomial_inner_product_distance() takes two arrays of points on a polynomial (of same size sampled on similar time points which could be numeric values for designations,remunerations or durations), polyfit()-s the points to two polynomials and computes inner product of the two polynomials which is the clustering similarity measure for two people profiles.
3.second() is a utility function for getting the second element in an array of ordered tuples. Usual cast of array of ordered tuple to dict() and invoking values() on them returns unique values and not a multiset of repetitive values which is redressed by second().
4.Utility Function normalize() divides a numeric array by maximum element which might be necessary for comparing trends in remunerations,designations and

durations of two talent profiles.
5.Apart from these, weighted observations (designations) within Chaotic HMM are as well fitted to a polynomial in career_transition_analytics()
6.All polynomials are of degree 5 and logs testlogs/SocialNetworkAnalysis_PeopleAnalytics_ChaoticHMM.log.30December2020 demonstrate the inner product distance measures and fitted career polynomials through few example inputs.

1132. (THEORY and FEATURE) People Analytics - Alphabet Vectorspace Embedding - Earth Mover Distance between String Syllable Tensors - related to 752,874,875,1123 and all sections on Embedding in vectorspace, Compressed Sensing, Social Network Analytics, Meaning Representation, Intrinsic Merit of People Profiles/HR Analytics, Set Partition and Histogram analytics - 5 February 2021
---
1. SocialNetworkAnalysis_PeopleAnalytics.py implementation has been updated to compare Earth Mover Distance and Wagner-Fischer Edit distance.
2. Few debug print()s have been added to print EMD and Edit distance measures for 2 strings. EMD computes the Earth Mover Distance between
 histograms of 2 strings.
3. Each string histogram has single alphabet buckets which reduces Edit Distance to Earth Mover Distance and subquadratic Earth Mover Distance computation (e.g LC-ACT linear complexity approximation of EMD has high accuracy close to 100% depending on iterations) implies SETH is False.

1133. (THEORY and FEATURE) Recursive Lambda Function Growth - Transformer Attention Model - related to 697,1131 and all sections on Intrinsic Merit of Texts, Embedding in vectorspace, Machine Translation, Named Entity Recognition and Meaning Representation - 5 February 2021

1.Transformers attention model implementation in NeuronRain could be used for Named Entity Recognition and Machine Translation.
2.Attention of a word vertex wi to word vertex wj in text graph from Recursive Gloss Overlap and Recursive Lambda Function Growth algorithms is heuristically inverse-proportional to degree of the vertex wj - extent to which meaning of the word vertex wj depends on wi.
3.Transformer attention model learns Query weights (Q), Key weights (K), Value weights (V) for embedding xi for word i defined by:
      Qi = xi * qi
      Ki = xi * ki
      Vi = xi * vi
      Attention(Q,K,V) = softmax(Q * K^T/sqrt(d^k))*V, d^k = dimension of Keys vector
      a(i,j) = attention of word i to word j = qi * kj
      softmax(xi) = e^xi/Sigma(e^xj) which normalizes sum to 1
4.Every word vertex of the text graph could be embedded in a vectorspace by word2vec and from previous transformer attention model:
      attention of word vertex wi to word vertex wj in text graph = a(wi,wj) = proportional to (1/degree(wj)) = qi * kj
5.New function rlfg_transformers_attention_model() has been implemented in RecursiveLambdaFunctionGrowth.py which computes degree of every word vertex in a text definition graph, derives degree attention matrix from them and learns Query weights, Key weights and Value weights from degree based attentions of the word vertices of text graph by Gradient Descent.
6.Product of learnt query and key weights yields the exact attention values between word vertices wi and wj.
7.Transformer Gradient Descent implementation in NeuronRain GRAFIT has been imported and invoked.

8.Logs for 1132 and 1133 in xz format have been committed:
- python-src/testlogs/SocialNetworkAnalysis_PeopleAnalytics.log.5February2021.xz
- python-src/testlogs/SocialNetworkAnalysis_PeopleAnalytics.log2.5February2021.xz
9.Debug statements and correction to for loops have been introduced in GRAFIT/course_material/NeuronRain/LinuxKernelAndCloud/code/Transformers_PerceptronAndGradient.py

---
1134. (THEORY and FEATURE) Named Entity Recognition - Transformers Attention Model - related to 697,1131,1133 and all sections on Named Entity Recognition - 9 February 2021,11 February 2021
---
1. NamedEntityRecognition_HMMViterbi_CRF.py has been updated to invoke rlfg_transformers_attention_model() from Recursive Lambda Function Growth python implementation for the textgraph of the observation text based on degree attention per word vertex defined earlier.
2. Perceptron learnt Query,Key and Value weights are printed in Recursive Lambda Function Growth rlfg_transformers_attention_model() routine.
3. Adjacency matrix for textgraph is instantiated by Numeric Python ones() function initialized to all 1s and cast to python list.
4. Textgraph edges (wi,wj) between word vertex wi to word vertex wj of maximum query,key,value weights could be high contributors to meaning of word vertex wj.
5. Test Logs for this code change are:

python-src/testlogs/NamedEntityRecognition_HMMViterbi_CRF.log.11February2021

python-src/testlogs/NamedEntityRecognition_HMMViterbi_CRF.log.9February2021
(merged and compressed in python-src/testlogs/NamedEntityRecognition_HMMViterbi_CRF.log.11February2021.xz)

1135. (THEORY and FEATURE) SETH is 100% False - Draft Proof - related to 752 and all sections on Earth Mover Distance, Edit Distance, BigData and String analytics, String complexity measures, Histogram analytics, Merit of Large Scale Visuals - 11 March 2021,12 March 2021

1135.1.LC-ACT (Algorithm 3) in IBM Research paper on LC-RWMD has 2 nested loops (for and argmin) for variables p (size of histogram1) and q (size of histogram2) as quoted below:

```
Algorithm 3 Approximate Computation of ICT
1: function ACT(p, q, C, k)
2: t = 0 ◃ initialize transportation cost t
3: for i = 1 . . . , hp do ◃ iterate the indices of p
4:    s = argmink (Ci,[1...hq]) ◃ find top-k smallest
5:    l = 1 ◃ initialize l
6:    while l < k do
7:          r = min(pi , qs[l]) ◃ size of max. transfer
8:          pi = pi − r ◃ move r units of pi to qs[l]
9:          t = t + r · Ci,j ◃ update cost
10:         l = l + 1 ◃ increment l
11:   end while
12:   if pi != 0 then ◃ if pi still has some mass
13:         t = t + pi · Ci,s[k] ◃ move the rest to qs[k]
14:   end if
15: end for
16: return t ◃ return transportation cost t
17: end function
```

1135.2.Bottom left constant size quadrant rectangle of Edit distance memoization

table hence computes edit distances of constant size substring histograms of 1-alphabet buckets. Example edit distance table from https://web.stanford.edu/class/cs124/lec/med.pdf is quoted below:

```
      N 9 8 9 10 11 12 11 10 9 8
      O 8 7 8 9 10 11 10 9 8 9
      I 7 6 7 8 9 10 9 8 9 10
      T 6 5 6 7 8 9 8 9 10 11
      N 5 4 5 6 7 8 9 10 11 10
      E 4 3 4 5 6 7 8 9 10 9
      T 3 4 5 6 7 8 7 8 9 8
      N 2 3 4 5 6 7 8 7 8 7
      I 1 2 3 4 5 6 7 6 7 8
      # 0 1 2 3 4 5 6 7 8 9
        # E X E C U T I O N
```

1135.3 In previous example, constant quadrant could be the bottom left 4*4 square fragment of the table computed by LC-ACT in linear time while the rest of the table adheres to conventional quadratic recurrence:

```
      T 3 4 5 6
        N 2 3 4 5
        I 1 2 3 4
        # 0 1 2 3
          # E X E
```

1135.4.Algorithm 3 for LC-ACT has histogram sizes p,q and iterations k as arguments. Therefore Algorithm 3 runs in predefined constant linear time for constant size quadrant of edit distance table - Iterations variable k could be appropriately predetermined for constants p and q of histogram sizes - For 100% accuracy iterations depend on string histogram sizes p and q. Each pi and qs have sizes <= 1 because string histograms have buckets of 1 alphabet.

1135.5.Thus a fraction of edit distance dynamic programming is performed in linear time. This reduces the exponent of quadratic time edit distance computation from 2 to 1.99999... or less asymptotically depending on constant size quadrant which implies edit distance is subquadratic and SETH is false.

1135.6.Constant size quadrant fraction of edit distance dynamic programming table of sides x and y has maximum distance computation d(x,y) in top right corner of the constant quadrant for 2 string histograms of lengths x and y equivalent to edit distance between 2 strings of lengths x and y - this assumes LC-ACT distances for string histograms are directly proportional to integral edit distances between strings of those histograms. In previous example fragment d(#INT,#EXE) = 6 for x=4 and y=4.

1135.7.Thus x and y are the maximum constant values for 2 nested loops of string histograms of size p and q (x=p, y=q).

1135.8.Constant number of iterations k in LC-ACT could be found once in initialization phase for constant values p=x and q=y and used as constant upperbound for all computations of distances within the constant sized quadrant rectangle.

1135.9.LC-ACT depends only on lengths p,q of string histograms and iterations k.

1135.10.Edit distance table grows from bottom left to top right. Therefore LC-ACT optimization works only for a constant sized rectangle in bottom left of the table, where constant length substring distances are computed.

1135.11 For string histograms, each bucket has only one alphabet and number of iterations (k) required is only 1 from LC-ACT algorithm 3 earlier (lines 7 to 10) because r <= 1 which is a major runtime reduction - at most one alphabet could be moved from bin of p to bin of q.

1135.12 Complexity of data parallel LC-ACT implementation (Table 3 of 1135.14) is O(vhm + nhk) where v = size of vocabulary, h = average size of the string histograms, m = dimension of the coordinates, n = number of string histograms, k = iterations.

1135.13 For Edit distance - Earth mover distance reduction, n = 1 (for distance between pair of string histograms), m = 1 (because each bin has only one alphabet) and k = 1 (because only one alphabet is moved) which implies the Complexity of LC-ACT edit distance computation is O(vh + h) which is linear in average size of the string histograms (or) linear in average length of the strings if size of the vocabulary v is constant (ASCII or Unicode) and thus subquadratic implying SETH is false and previous constant quadrant optimization

could be unnecessary on the average.

References:

1135.14 LC-RWMD and LC-ACT - Algorithm 3 - http://proceedings.mlr.press/v97/atasu19a/atasu19a.pdf
1135.15 Edit Distance Dynamic Program Memoization - https://web.stanford.edu/class/cs124/lec/med.pdf

---

1136. (THEORY and FEATURE) SETH is 100% False - Draft Proof - alternative Edit distance-Earth mover distance reduction - related to 752,1135 and all sections on Earth Mover Distance, Edit Distance, BigData and String analytics, String complexity measures, Histogram analytics, Merit of Large Scale Visuals, Embedding of Strings in Syllable vector space - 24 March 2021, 11 April 2021

1136.1 As alternative to 1 alphabets per bin, every String histogram has buckets containing ASCII or Unicode numeric values of alphabet as numeric weights. This makes string histograms more similar to traditional histograms having bins of numeric weights e.g. Image Pixel Histograms of Large Scale Visuals.
1136.2 LC-ACT is thus readily applicable to string histograms of numeric weights. Direct proportionality assumption between edit distance values and earth mover distance values stated earlier in 1135 has to be proved.
1136.3 For same strings both edit distance and earth mover distance are zero. For dissimilar strings edit distance inserts alphabets, deletes alphabets and updates alphabets while previous numeric weights string histogram for earth mover distance depends on r and C(i,j) - r and C(i,j) increase with widening numeric weights (differing alphabets) - Edit distance counts only insertions, deletions and substitutions while EMD generalizes edit distance.
1136.4 Because there is not much literature on relation between edit distance and earth mover distance (except L1-embeddings which exist for all classes of distance metrics) following equation in 1136.5 has been contrived for proving proportionality between former and latter inspired by edit distance dynamic programming recurrence:
Edit distance = minimum(d(i-1,j-1),d(i-1,j) + delete(ai),d(i,j-1) + insert(bj),d(i-1,j-1) + substitute(ai,bj))
1136.5 Earth Mover Distance = w1*Sum(unicode values of inserted alphabets) - w2*Sum(unicode values of deleted alphabets) + w3*Sum(difference between unicode values of substituted alphabets)
1136.6 Previous linear program sums unicode values of inserted alphabets in string histogram bins, subtracts unicode values of deleted alphabets in string histogram bins, sums the difference between unicode values of substituted alphabets in string histogram bins by assigning appropriate weights which is same as lines 7-10 of LC-ACT data parallel algorithm earlier.
1136.7 Instead of 1s for counting insertions, deletions and substitutions, unicode value weights of the string histogram bins are used which maps edit distance to earth mover distance - iterated summation of r (=difference between weights of bins in 2 histograms) * C(i,j) (=cost).
1136.8 Replacing unicode weights by 1s reverts EMD to Edit distance. Number of iterations of LC-ACT is upperbounded by total number of alphabet encodings e.g. 256 for ASCII, 512 for Unicode - for numeric string histograms which is again constant and thus linear time and accuracy is 100%.
1136.9 An example computation of LC-ACT algorithm - lines 7-10:
For pi = 3 (unicode weight) and qs = 1 (unicode weight), r = min(3,1) = 1. If C(i,j) is 1 uniformly throughout:
pi - r = 3 - 1 = 2 (iteration 1), t = 1
pi - r = 2 - 1 = 1 (iteration 2), t = 2
1136.10 Edit distance to Earth mover distance reduction could be useful as well for (*) computing distance between syllable hyphenated natural language strings in which each string is a histogram of bins containing syllables (*) and stream of Survival Index WCET OS Scheduler Histograms in which timeslice bins are process id strings.

References:

1136.11 High Dimensional EMD - [Andoni-Indyk-Krauthgamer] - http://www.cs.columbia.edu/~andoni/papers/emd-hd.pdf - "...Since many metrics (e.g., variants of edit distance [MS00, CM07, OR05, CK06]) can be embedded with bounded distortion into L1 with small integer coordinates, our result implies that EMD over s-subsets of such metrics can be embedded into L1 as well...."
1136.12 Linear sketch for EMD - https://people.cs.umass.edu/~mcgregor/papers/13-approx1.pdf - "...Example 1. Suppose A = {2, 3, 10} and B = {3, 4, 8} and note that EMD(A, B) = 4. Then x = (0, 1, 2, 2, 2, 2, 2, 2, 2) and y = (0, 0, 1, 2, 2, 2, 2, 3, 3) and |x − y|1 = 4 as required ..."
1136.13 Embeddings of non-norms (Earth Mover Distance and Edit Distance) to norms(L1),Distortion,Non-embeddability - https://simons.berkeley.edu/sites/default/files/docs/516/andonislides2.pdf

---
1137. (THEORY and FEATURE) Intrinsic Merit Versus Majority Voting, P=BPP ("Maybe" tends to "Exists") in the context of Condorcet Jury Theorem and its derivatives, Margulis-Russo Threshold for boolean majority, Histogram analytics, Set Partition Theoretical Electronic Voting Machines, Edit distance to Earth Mover Distance reduction, Boolean function composition, KRW Conjecture, Communication complexity, Hardness amplification, Bounded Rationality, Gibbard-Satterthwaite Theorem - related to 400,572,688,1136 and all sections on complexity theoretic aspects of Majority Voting and Intrinsic merit - 31 March 2021

Condorcet Jury Theorem and its later refinements formalize goodness of majority voting function allowing voter error. Bounded probabilistic polynomial time Turing Machine (BPP) accepts YES instances (x in L) with probability > 2/3 and rejects NO instances with probability < 1/3 (x not in L). Bounded error majority voting Turing machine with p-bias < 1/2 (voter has bounded rationality and errs < 50% while choosing a candidate) converges to 1 for infinite electorate by CJT and its variants implying P=BPP for infinite majority voting - BPP majority voting Turing machine elects a good candidate xi from input vector x of candidates by gathering votes vector v from infinite number of voters with probability ~1 if xi in L (Good) or rejects with probability ~0 otherwise by Condorcet Jury Theorem. In the context of intrinsic merit of an entity, infinite voters having p-bias < 1/2 (low error) increasingly reach consensus on perceived merit - intrinsic merit and perceived merit coincide. Previous BPP majority voting Turing machine is depth-1 majority and does not compose boolean functions. Depth-2 majority function composes majority and individual voter SATs (boolean for binary or non-boolean for multipartisan) of arbitrary complexity class. For example, Majority * BPP denotes a composition of boolean majority with Voter functions of BPP where every voter function is a BPP machine which decides correctly with > 2/3 probability (p-bias < 1/2) implying outcome of voting tends to 100% correctness for infinite number of voters (but polynomial e.g number of voters x^1000000000000 is polynomial though huge). BPP is low for itself => BPP^BPP = BPP and if Majority * BPP could be alternatively formalized as oracle machine BPP^BPP then depth-2 majority is in BPP and by CJT, voting tends to 100% goodness for infinite number of voters and BPP is derandomized to P. Previous Majority * BPP boolean composition is in KRW conjecture - Communication complexity regime. P=BPP does not imply P=NP and thus hardness amplification (and separation of P from NP) is not contradicted though converse P=NP implies P=BPP. As an aside, Edit distance to Earth mover distance reduction earlier applies to theoretical set partition voting machine histograms defined and implemented in NeuronRain if votes in histogram bins are encoded as strings. Gibbard-Satterthwaite theorem implies no voting system for more than 2 candidates is manipulation free thus implying non-boolean majority functions (boolean majority is restricted to 2 candidates) are always in Probabilistic complexity classes.

References:

1137.1 Election Manipulation and Integer Factorization - [Edith Hemaspaandra , Lane A. Hemaspaandra , and Curtis Menton] - https://drops.dagstuhl.de/opus/volltexte/2013/3949/pdf/37.pdf - "...If integer factoring is hard, then for election manipulation, election bribery, and some types of election control, there are election systems for which recognizing which instances can be successfully manipulated is in polynomial time but producing the successful manipulations cannot be done in polynomial time...."

1138. (THEORY and FEATURE) Recursive Lambda Function Growth Textgraph as EventNet Causality Graphical Event Model - implementation -
12 April 2021 - related to 762 and all sections on Intrinsic Merit of Text and Videos, EventNet, Graphical Event Models
---
1.New python source file RecursiveLambdaFunctionGrowth_EventNetGEM.py derived from RecursiveLambdaFunctionGrowth.py implements
function rlfg_eventnet_graphical_event_model() which takes a Recursive Lambda Function Growth WordNet textgraph as argument and computes all pairs shortest paths between word vertices from textgraph.
2.All pairs shortest paths in the textgraph are filtered for words of verb part-of-speech and top core number words of dense subgraph unsupervised classifier - rationale being paths between word vertices containing lot of verb vertices are approximate chronology and causality indicators of actors and actions while words of top core numbers are keywords of the text - "who does what to whom".
3.This EventNet causality model for texts (mostly news articles) is inspired by EventNet Tensor product algorithm implementation for Merit of Large Scale Visuals which determines causality between frames of video by ImageNet.
4.Textgraph of an example news article is filtered for verb paths - some exceprts from
testlogs/RecursiveLambdaFunctionGrowth_EventNetGEM.log.12April2021:
['government', 'unit', 'density', 'size', 'add', 'make', 'propose', 'proposal', 'intention', 'usually', 'area', 'urban', 'city', 'Chennai', 'Tamil', 'Dravidian']
['government', 'unit', 'density', 'size', 'add', 'make', 'propose', 'proposal', 'intention', 'usually', 'area', 'urban', 'city', 'Chennai', 'Tamil', 'language']
['influence', 'extend', 'expand']
5.Existing Graphical Event Models viz, PGEM extract graph dependency from timeseries of events by probabilistic maximum likelihood parent search while previous algorithm is graph theoretic. Start and/or end vertices of paths are (actors) nouns while most intermediary nodes are (actions) verbs.

1139. (THEORY and FEATURE) Recursive Lambda Function Growth EventNet GEM - CoronaVirus 2019 GEM usecase - 5,12 May 2021 - related to 752, 1138
and all sections of Intrinsic Merit, EventNet Causal and Graphical Event Models
---
1.RecursiveLambdaFunctionGrowth_EventNetGEM.py is executed on a text file containing abstract of a research article on COVID19
(Is the 2019 novel coronavirus related to a spike of cosmic rays? - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7529058/) and a
textgraph created by it is analyzed for random walk causality (having noun-verb-noun patterns fitting actor-action-actor cause-effect
template). [As an astronomical aside 2 peak waves of COVID19 tantalizingly coincided with 2 notable celestial conjunctions of planets in December 2019 and February 2021 apart from Solar eclipses in 2019 and 2020 (which are known for malevolent radiations) and an abnormal comet passing tangential to earth (Borisov 2019)]
2.Following are some of the notable random walks in textgraph EventNet GEM

showing actor-action causalities from python-src/testlogs/RecursiveLambdaFunctionGrowth_EventNetGEM.log.5May2021.xz:

```
...
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended'],
['determined', 'magnetic', 'sunspot', 'field', 'space', 'cosmic', 'extended']
...
```

3.Nested loop for verb filtering has been optimized.
4.Recursive Lambda Function Growth EventNet GEM textgraph is particularly useful when two different seemingly unrelated research articles are collated (their texts are merged) and textgraph from merged text is analyzed for actor-action-actor causality random walks. This could prove hidden connections between hitherto unrelated fields of research.

1142. (THEORY and FEATURE) SETH is 100% False - Draft Proof - Edit distance-Earth mover distance reduction and order preservation - illustration - related to 752,1135,1136 and all sections on Earth Mover Distance, Edit Distance, BigData and String analytics, String complexity measures, Histogram analytics and Set partitions, Merit of Large Scale Visuals, Embedding of Strings in Syllable vector space - 26 May 2021

1142.1 Earth mover distance - Edit distance reduction preserves order of bins in histograms because histograms are usually set partitions of
a numeric range on x-axis (or y-axis) and thus order of alphabets in a string and order of single alphabet buckets in string histograms are same.
1142.2 Single alphabet buckets of string histogram therefore are order preserving and Earth Mover distance and Edit distance are equivalent and directly proportional (Insertion of an alphabet = Insertion of new single alphabet bucket, Deletion of an alphabet = Removal of single alphabet bucket, Update of an alphabet = Update of single alphabet bucket)
1142.3 For string histograms of numeric ASCII or Unicode values for each 1-alphabet bucket, the order preservation is obvious - each bucket has size equal to ASCII or Unicode value of the alphabet.
1142.4 Following illustration depicts Earth mover distance to Edit distance reduction between two string histograms for strings "acdfp" and "qdpfg" (each bin has size equal to ASCII or Unicode values of the alphabet):

```
                  String histogram1 ("acdfp")   ===== morphed to =====>String
histogram2 ("qdpfg")

                  a ##############                          q #################

                  c ######                                  d ########
                  d ########                                p #####
```

```
                f ##########                                    f #######
                p ####                                          g #########
```

1142.5 Earth Mover Distance (EMD) - Edit Distance (ED) reduction is immediate for numeric string histograms because transportation of bin weights in LC-ACT - Algorithm 3, lines 7-10 - (which are differences in ASCII or Unicode values of the bins) changes ASCII or Unicode values between strings morphing one string to another (Example iterations in 1136.9) - direct proportionality is by linear program earlier.
1142.6 Previous illustration envisions an alternative spectacle to view a string - every string is a numeric set partition of sum of ASCII or Unicde values of constituent alphabets and thus edit distance between strings is equivalent to distance between numeric ASCII or Unicode set partitions which unusually relates String complexity and Theory of partitions.

1143. (THEORY and FEATURE) Recursive Lambda Function Growth EventNet GEM - 2 different research articles collated - related to 752, 1138,
1139 and all sections of Intrinsic Merit, EventNet Causal and Graphical Event Models - 8 June 2021

1.RecursiveLambdaFunctionGrowth_EventNetGEM.py has been executed on merged abstracts of 2 different research articles on Collatz conjecture and Pseudorandom number generators (but both are indirectly related to Chaotic maps and Fractals) - https://www.math.auckland.ac.nz/~king/745/collatz.pdf and https://www.researchgate.net/publication/332583614_Pseudo-random_number_generators_based_on_the_Collatz_conjecture - which reasonably extracts the crux of the two articles and finds random walks connecting the abstracts.
2.There are some irrelevant word vertices which could be because of deficient word sense ambiguation (Primitive Lesk and PyWSD) and non-scientific nature of WordNet which is more linguistic and might therefore necessitate a Concept Ontology specific to Theoretical and Experimental Science.
3.Cause-effect random walks extracted from the previous 2 articles on Collatz conjecture and Pseudorandom number generators have prominent word vertices "simulation", "graph", "mathematics" which convey the common thread connecting the two articles. Full text merge of the 2 articles might increase the accuracy but cause slowdown.
#/
```
**************************************************************************
**********************
#* UMB - Universal Modified  Bus Driver - simple USB driver for debugging
#* This program is free software: you can redistribute it and/or modify
#* it under the terms of the GNU General Public License as published by
#* the Free Software Foundation, either version 3 of the License, or
#* (at your option) any later version.
#*
#* This program is distributed in the hope that it will be useful,
#* but WITHOUT ANY WARRANTY; without even the implied warranty of
#* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#* GNU General Public License for more details.
#*
#* You should have received a copy of the GNU General Public License
#* along with this program.  If not, see <http://www.gnu.org/licenses/>.
#*
#-------------------------------------------------------------------------
-------------------------
#Copyleft (Copyright+):
#Srinivasan Kannan (alias) Ka.Shrinivaasan (alias) Shrinivas Kannan
#Ph: 9791499106, 9003082186
#Krishna iResearch Open Source Products Profiles:
#http://sourceforge.net/users/ka_shrinivaasan,
```

#https://github.com/shrinivaasanka,
#https://www.openhub.net/accounts/ka_shrinivaasan
#Personal website(research): https://sites.google.com/site/kuja27/
#emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
#kashrinivaasan@live.com
#---

#***
************************/

USBmd driver is an experimental modified version of already existing USB driver
in linux.

Purpose of this modified version is for doing more sophisticated debugging of
USB endpoints and devices and as
USB packet sniffer. Technical Necessity for this was created due to prolonged
data theft, id spoofing and cybercrime that has been happening
in author's personal electronic devices for years that resulted in a Cybercrime
Police Complaint also few years ago.

There were also such incidents while developing open source code (some code
commits have description of these mysterious occurrences). There is no
comprehensive USB debugger available on linux to sift bad traffic though there
are strong evidences of such cybercrime and datatheft through other sources.
Author is inclined to believe that such recurring events of datatheft that
defies all logic can have no other intent but to cause malafide theft or loss of
private data and an act of defamation among other things.

This is also done as a technical learning exercise to analyze USB Hosts, packets
and USB's interaction,if any, with wireless devices including
mobiles, wireless LANs(radiotap) etc.,

In the longterm USBmd might have to be integrated into VIRGO. As VIRGO would
would have the synergy of AstroInfer machine learning
codebase for "learning" from datasets, this USBmd driver can have the added
ability of analyzing large USB traffic (as a dataset)
using some decision making algorithms and evolve as an anti-cybercrime, anti-
plagiarism and anti-theft tool to single out
"malevolent" traffic that would save individuals and organisations from the
travails of tampering and loss of sensitive confidential data.

The pattern mining of numeric dataset designed for AstroInfer can apply here
also since USB bitstream can be analyzed using algorithms for
numerical dataset mining. Also Discrete Fourier Transform used for analyzing
data for frequencies (periodicities if any) can be used for
USB data , for example USB wireless traffic.

===
new UMB driver bind - 27 Feb 2014 (for Bus id 7)
===
Following example commandlines install umb.ko module, unbind the existing option
driver from bus-device id and bind the umb.ko to that bus id:

```
sudo insmod umb.ko
echo -n "7-1:1.0" > /sys/bus/usb/drivers/option/unbind
echo -n "7-1:1.0" > /sys/bus/usb/drivers/umb/bind
```

===
Commits as on 29 July 2014
===
Driver has been ported and built on 3.15.5 kernel. Also a driver build script
has been committed.

```
------------------------------------------------------
USBmd version 14.9.9 has been release tagged on 9 September 2014
------------------------------------------------------
------------------------------------------------------
USBmd version 15.1.8 has been release tagged on 8 January 2015
------------------------------------------------------

http://sourceforge.net/p/usb-md/code-0/HEAD/tree/Adding%20new%20vendor%20and
%20product%20IDs%20to%20an%20existing%20USB%20driver%20on%20Linux.html has steps
to add new vendor-id.

------------------------------------------------------------------------------
USB debug messages from "cat /sys/kernel/debug/usb/devices" for UMB bound above:
------------------------------------------------------------------------------

T:  Bus=07 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#= 12 Spd=12   MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=12d1 ProdID=140b Rev= 0.00
S:  Manufacturer=HUAÿWEI TECHNOLOGIES
S:  Product=HUAWEI Mobile
S:  SerialNumber=ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
C:* #Ifs= 4 Cfg#= 1 Atr=a0 MxPwr=500mA
I:* If#= 0 Alt= 0 #EPs= 3 Cls=ff(vend.) Sub=ff Prot=ff Driver=umb
E:  Ad=81(I) Atr=03(Int.) MxPS=  16 Ivl=128ms
E:  Ad=82(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=02(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
I:* If#= 1 Alt= 0 #EPs= 2 Cls=ff(vend.) Sub=ff Prot=ff Driver=option
E:  Ad=84(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=04(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
I:* If#= 2 Alt= 0 #EPs= 2 Cls=ff(vend.) Sub=ff Prot=ff Driver=option
E:  Ad=86(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=06(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
I:* If#= 3 Alt= 0 #EPs= 2 Cls=08(stor.) Sub=06 Prot=50 Driver=usb-storage
E:  Ad=87(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=08(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms

------------------------------------------------------------------------
usbmon, libpcap tcpdump and wireshark (or vusb-analyzer) debugging
------------------------------------------------------------------------
*mount -t debugfs none_debugs /sys/kernel/debug
*modprobe usbmon
*ls /sys/kernel/debug/usb/usbmon/

0s  0u      1s  1t      1u  2s      2t  2u      3s  3t      3u  4s      4t  4u
      5s  5t      5u  6s      6t  6u      7s  7t      7u  8s      8t  8u

*cat /sys/kernel/debug/usb/usbmon/8t > usbmon.mon (any of the above usbmon debug
logs)
*vusb-analyzer usbmon.mon

ef728540 3811287714 S Ci:001:00 s a3 00 0000 0001 0004 4 <
ef728540 3811287743 C Ci:001:00 0 4 = 00010000
ef728540 3811287752 S Ci:001:00 s a3 00 0000 0002 0004 4 <
ef728540 3811287763 C Ci:001:00 0 4 = 00010000
f50f6540 3811287770 S Ii:001:01 -115 2 <
f50f6540 3811287853 C Ii:001:01 -2 0
f5390540 3814543695 S Ci:001:00 s a3 00 0000 0001 0004 4 <
f5390540 3814543715 C Ci:001:00 0 4 = 00010000
f5390540 3814543756 S Ci:001:00 s a3 00 0000 0002 0004 4 <
f5390540 3814543767 C Ci:001:00 0 4 = 00010000
f50f6540 3814543805 S Ii:001:01 -115 2 <

*modprobe usbmon
*ls /dev/usbmon[1-8]
```

```
*tcpdump -i usbmon1 -w usbmon.pcap
tcpdump: listening on usbmon1, link-type USB_LINUX_MMAPPED (USB with padded
Linux header), capture size 65535 bytes
^C86 packets captured
86 packets received by filter

*wireshark usbmon.pcap (loads on wireshark)

------------------------------------------------------------
Dynamic Debug - dev_dbg() and dev_vdbg()
------------------------------------------------------------

USB Debugging References:
-------------------------
- Texas Instruments -
http://elinux.org/images/1/17/USB_Debugging_and_Profiling_Techniques.pdf

------------------------------------------------------------
NeuronRain version 15.6.15 release tagged
------------------------------------------------------------

------------------------------------------------------------
Commits as on 11 July 2015
------------------------------------------------------------
usbmd kernel module has been ported to Linux Kernel 4.0.5

------------------------------------------------------------
Commits as on 26 November 2015
------------------------------------------------------------
- Updated USB-md driver with a lookup of VIRGO kernel_analytics config variable
exported by kernel_analytics module in umb_read() as default.
- New header file umb.h has been added that externs the VIRGO kernel_analytics
config array variables
- Module.symvers has been imported from VIRGO kernel_analytics and clean target
has been commented in build script after initial build as make clean removes
Module.symvers.
- kern.log with umb_read() and umb_write() have been added with following
commandlines:
      - cat /dev/umb0 - invokes umb_read() but there are kernel panics sometimes
      - cat <file> > /dev/umb0 - invokes umb_write()
  where umb0 is usb-md device name registered with /sys/bus/usb as below:
      - insmod umb.ko
      - echo -n "7-1:1.0" > /sys/bus/usb/drivers/option/unbind
      - echo -n "7-1:1.0" > /sys/bus/usb/drivers/umb/bind
- Updated build generated sources and object files have been added

------------------------------------------------------------
Commits as on 27 November 2015
------------------------------------------------------------
New folder usb_wwan_modified has been added that contains the USB serial, option
and wireless USB modem WWAN drivers from kernel mainline
instrumented with lot of printk()s so that log messages are written to kern.log.
Though dev_dbg dynamic debugging can be used by writing to
/sys/kernel/debug/<...>/dynamic_debug
printk()s are sufficient for now. This traces through the USB connect and data
transfer code:
        - probe
        - buffer is copied from userspace to kernelspace
        - URB is allocated in kernel
        - buffer is memcopied to URB
        - usb send/receive bulk pipe calls
        - usb_fill_bulk_urb
Almost all buffers like in and out buffers in URBs, portdata, interfacedata,
serial_data, serial_port_data are printed to kern.log. This log is
```

analyzable by AsFer machine learning code for USB debugging similar to usbmon logs.

These are initial commits only and usb-serial.c, usb_wwan.c, option.c and serial.h might be significantly altered going forward.

Commits as on 30 November 2015

Added usb.h from kernel mainline, instrumented with printk() to print transfer_buffer in usb_fill_[control/bulk/interrupt]_urb() functions. kern.log for this has been added in usb_wwan_modified/testlogs.

Commits as on 1 December 2015

- new kernel function print_buffer() has been added in usb.h that prints contents of char buffer in hex
- Above print_buffer() is invoked to print transfer_buffer in usb_wwan.c, usb-serial.c, option.c
- kern.log with print_buffer() output has been added - This dumps similar to wireshark, usbmon and other usb analyzers.

Commits as on 2 December 2015

- changed print_buffer() printk() to print a delimiter in each byte for AsFer Machine Learning code processing
- add a parser script for kern.log to print print_buffer() lines
- parsed kern.log with print_buffer() lines has been added
- Added an Apache Spark MapReduce python script to compute byte frequency in parsed print_buffer() kern.log

---
(ONGOING) NeuronRain USBmd Debug and Malafide Traffic Analytics
---
As mentioned in commit notes above, USB incoming and outgoing data transfer_buffer are dumped byte-by-byte. Given this data various analytics can be performed most of which are already implemented in AsFer codebase:
- frequency of bytes
- most frequent sequence of bytes
- bayesian and decision tree inference
- deep learning
- perceptrons
- streaming algorithms for USB data stream
and so on.

Commits as on 3 December 2015

- Apache Spark script for analyzing the USBWWAN byte stream logs has been updated with byte counts map-reduce functions from print_buffer() logs and temp DataFrame Table creation with SparkSQL.
- logs for the script have been added in usb_wwan_modified/python-src/testlogs/Spark_USBWWANLogMapReduceParser.out.3December2015
- kern.log parser shellscript has been updated

AsFer commits for USBmd as on 4 December 2015

All the Streaming_<>.py Streaming Algorithm implementations in AsFer/python-src/ have been updated with:

- hashlib ripemd160 hash MD algorithm for hash functions and return hexdigest()
- USBWWAN byte stream data from USBmd print_buffer() logs in usb-md/usb_wwan_modified/testlogs/ has been added as a Data Storage and Data Source
- logs for the above have been added to asfer/python-src/testlogs/
- Streaming Abstract Generator has been updated with USB stream data iterable and parametrized for data source and storage
- Some corrections to the asfer/python-src/Streaming_<> scripts

---
Commits as on 7 December 2015

- added Spark Mapreduce and DataFrame log for USBWWAN byte stream
- added a parsed kern.log with only bytes from USBWWAN stream
- Added dict() and sort() for query results and printed cardinality of the stream data set which is the size of the dict.
An example log has been added which prints the cardinality as ~250. In contrast, LogLog and HyperLogLog counter estimations
approximate the cardinality to 140 and 110 respectively

AsFer commits for USBmd as on 11 December 2015 - USBWWAN stream data backend in MongoDB

 Dependency Injection code commits for MongoDB backend - With this MongoDB is also a storage backend for AsFer algorithms similar to MySQL:
- Abstract_DBBackend.py has been updated for both MySQL and MongoDB injections
- MongoDB configuration and backend connect/query code has been added. Backend is either populated by Robomongo or pymongo reading from the Streaming Abstract Generator iterable framework.
- With this AsFer supports both SQL(MySQL) and NoSQL(file,hive,hbase,cassandra backends in Streaming Abstract Generator).
- log with a simple NoSQL table with StreamingData.txt and USBWWAN data has been added to testlogs/.
- MongoDB configuration has a database(asfer-database) and a collection(asfer-collection).
- MongoDB_DBBackend @provides pymongo.collection.Collection which is @inject-ed to Abstract_DBBackend

---
Commits as on 10 January 2016

NeuronRain USBmd enterprise version 2016.1.10 released.

Commits - 4 August 2016

1.New build script for drivers/usb top level folder has been added.
2.Copyleft notices updated
3.print_buffer() in usb.h has been #ifdef-ed based on a build time flag to suppress the buffer bytes dump preferentially so that
kern.log is not flooded.
4.Flag PRINT_BUFFER has to be defined with #define somewhere within KBuild makefiles or externally.
5..ko files rebuilt
6. Miscellaneous code changes to suppress kbuild warnings - cast etc.,
7. PRINT_BUFFER block changed to print the bytes in single line for each buffer

```
--------------------------------------------------------------------------------
--------------------------------
Commits - 13 July 2017 - usb-storage driver last sector access slab out of
bounds error in 64-bit - committed for analysis
- this error was frequently witnessed in VIRGO 32-bit stability issues and
panics - ISRA looks like a GCC
optimization of a function invocation (Interprocedural Scalar Replacement of
Aggregates)
--------------------------------------------------------------------------------
--------------------------------
Jul 13 15:03:36 localhost kernel: [ 9837.497280]
==================================================================
Jul 13 15:03:36 localhost kernel: [ 9837.499787]
==================================================================
Jul 13 15:03:36 localhost kernel: [ 9837.499822] BUG: KASAN: slab-out-of-bounds
in last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage] at addr
ffff88007cdaa758
Jul 13 15:03:36 localhost kernel: [ 9837.499831] Read of size 8 by task usb-
storage/6243
Jul 13 15:03:36 localhost kernel: [ 9837.499844] CPU: 0 PID: 6243 Comm: usb-
storage Tainted: G    B          4.10.3 #18
Jul 13 15:03:36 localhost kernel: [ 9837.499849] Hardware name: Dell Inc.
Inspiron 1545                   /0J037P, BIOS A14 12/07/2009
Jul 13 15:03:36 localhost kernel: [ 9837.499851] Call Trace:
Jul 13 15:03:36 localhost kernel: [ 9837.499863]  dump_stack+0x63/0x8b
Jul 13 15:03:36 localhost kernel: [ 9837.499870]  kasan_object_err+0x21/0x70
Jul 13 15:03:36 localhost kernel: [ 9837.499877]
kasan_report.part.1+0x219/0x4f0
Jul 13 15:03:36 localhost kernel: [ 9837.499893]  ?
last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.499899]  kasan_report+0x25/0x30
Jul 13 15:03:36 localhost kernel: [ 9837.499906]  __asan_load8+0x5e/0x70
Jul 13 15:03:36 localhost kernel: [ 9837.499922]
last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.499938]
usb_stor_invoke_transport+0x1a1/0x960 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.499946]  ?
migrate_swap_stop+0x2e0/0x2e0
Jul 13 15:03:36 localhost kernel: [ 9837.499963]  ?
usb_stor_port_reset+0xb0/0xb0 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.499973]  ?
wait_for_completion_interruptible+0x1a7/0x260
Jul 13 15:03:36 localhost kernel: [ 9837.499981]  ?
wait_for_completion_killable+0x2a0/0x2a0
Jul 13 15:03:36 localhost kernel: [ 9837.499989]  ?
raise_softirq_irqoff+0xba/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.499995]  ? wake_up_q+0x80/0x80
Jul 13 15:03:36 localhost kernel: [ 9837.500011]
usb_stor_transparent_scsi_command+0xe/0x10 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
usb_stor_control_thread+0x344/0x510 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ?
usb_stor_disconnect+0x120/0x120 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ?
default_wake_function+0x2f/0x40
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ? __wake_up_common+0x78/0xc0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kthread+0x178/0x1d0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ?
usb_stor_disconnect+0x120/0x120 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ?
kthread_create_on_node+0xd0/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ret_from_fork+0x2c/0x40
Jul 13 15:03:36 localhost kernel: [ 9837.500017] Object at ffff88007cdaa668, in
cache kmalloc-192 size: 192
```

```
Jul 13 15:03:36 localhost kernel: [ 9837.500017] Allocated:
Jul 13 15:03:36 localhost kernel: [ 9837.500017] PID = 6277
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  save_stack_trace+0x1b/0x20
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  save_stack+0x46/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kasan_kmalloc+0xad/0xe0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
kmem_cache_alloc_trace+0xef/0x210
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kernfs_fop_open+0x14b/0x540
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  do_dentry_open+0x39a/0x560
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  vfs_open+0x84/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  path_openat+0x4ab/0x1e10
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  do_filp_open+0x122/0x1c0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  do_sys_open+0x17c/0x2c0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  compat_SyS_open+0x1b/0x20
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  do_fast_syscall_32+0x188/0x300
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
entry_SYSENTER_compat+0x4c/0x5b
Jul 13 15:03:36 localhost kernel: [ 9837.500017] Freed:
Jul 13 15:03:36 localhost kernel: [ 9837.500017] PID = 6277
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  save_stack_trace+0x1b/0x20
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  save_stack+0x46/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kasan_slab_free+0x71/0xb0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kfree+0x9e/0x1e0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kernfs_fop_release+0x87/0xa0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  __fput+0x177/0x350
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ____fput+0xe/0x10
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  task_work_run+0xa0/0xc0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
exit_to_usermode_loop+0xc5/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  do_fast_syscall_32+0x2ef/0x300
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
entry_SYSENTER_compat+0x4c/0x5b
Jul 13 15:03:36 localhost kernel: [ 9837.500017] Memory state around the buggy
address:
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ffff88007cdaa600: fc fc fc fc
fc fc fc fc fc fc fc fc fc fb fb fb
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ffff88007cdaa680: fb fb fb fb
fb fb fb fb fb fb fb fb fb fb fb fb
Jul 13 15:03:36 localhost kernel: [ 9837.500017] >ffff88007cdaa700: fb fb fb fb
fb fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
^
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ffff88007cdaa780: fc fc fc fc
fc fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ffff88007cdaa800: fc fc fc fc
fc fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
==================================================================
Jul 13 15:03:37 localhost kernel: [ 9837.668157]
==================================================================
Jul 13 15:03:37 localhost kernel: [ 9837.668191] BUG: KASAN: slab-out-of-bounds
in last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage] at addr
ffff88007cdaa758
Jul 13 15:03:37 localhost kernel: [ 9837.668200] Read of size 8 by task usb-
storage/6243
Jul 13 15:03:37 localhost kernel: [ 9837.668213] CPU: 1 PID: 6243 Comm: usb-
storage Tainted: G    B           4.10.3 #18
Jul 13 15:03:37 localhost kernel: [ 9837.668218] Hardware name: Dell Inc.
Inspiron 1545                   /0J037P, BIOS A14 12/07/2009
Jul 13 15:03:37 localhost kernel: [ 9837.668220] Call Trace:
Jul 13 15:03:37 localhost kernel: [ 9837.668233]  dump_stack+0x63/0x8b
Jul 13 15:03:37 localhost kernel: [ 9837.668240]  kasan_object_err+0x21/0x70
Jul 13 15:03:37 localhost kernel: [ 9837.668247]
kasan_report.part.1+0x219/0x4f0
```

```
Jul 13 15:03:37 localhost kernel: [ 9837.668263]  ?
last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668269]  kasan_report+0x25/0x30
Jul 13 15:03:37 localhost kernel: [ 9837.668277]  __asan_load8+0x5e/0x70
Jul 13 15:03:37 localhost kernel: [ 9837.668292]
last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668308]
usb_stor_invoke_transport+0x1a1/0x960 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668316]  ?
migrate_swap_stop+0x2e0/0x2e0
Jul 13 15:03:37 localhost kernel: [ 9837.668332]  ?
usb_stor_port_reset+0xb0/0xb0 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668343]  ?
wait_for_completion_interruptible+0x1a7/0x260
Jul 13 15:03:37 localhost kernel: [ 9837.668351]  ?
wait_for_completion_killable+0x2a0/0x2a0
Jul 13 15:03:37 localhost kernel: [ 9837.668360]  ?
raise_softirq_irqoff+0xba/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668366]  ? wake_up_q+0x80/0x80
Jul 13 15:03:37 localhost kernel: [ 9837.668382]
usb_stor_transparent_scsi_command+0xe/0x10 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668398]
usb_stor_control_thread+0x344/0x510 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668415]  ?
usb_stor_disconnect+0x120/0x120 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668422]  ?
default_wake_function+0x2f/0x40
Jul 13 15:03:37 localhost kernel: [ 9837.668430]  ? __wake_up_common+0x78/0xc0
Jul 13 15:03:37 localhost kernel: [ 9837.668436]  kthread+0x178/0x1d0
Jul 13 15:03:37 localhost kernel: [ 9837.668454]  ?
usb_stor_disconnect+0x120/0x120 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668460]  ?
kthread_create_on_node+0xd0/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668466]  ret_from_fork+0x2c/0x40
Jul 13 15:03:37 localhost kernel: [ 9837.668472] Object at ffff88007cdaa668, in
cache kmalloc-192 size: 192
Jul 13 15:03:37 localhost kernel: [ 9837.668478] Allocated:
Jul 13 15:03:37 localhost kernel: [ 9837.668483] PID = 6277
Jul 13 15:03:37 localhost kernel: [ 9837.668494]  save_stack_trace+0x1b/0x20
Jul 13 15:03:37 localhost kernel: [ 9837.668500]  save_stack+0x46/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668506]  kasan_kmalloc+0xad/0xe0
Jul 13 15:03:37 localhost kernel: [ 9837.668513]
kmem_cache_alloc_trace+0xef/0x210
Jul 13 15:03:37 localhost kernel: [ 9837.668520]  kernfs_fop_open+0x14b/0x540
Jul 13 15:03:37 localhost kernel: [ 9837.668527]  do_dentry_open+0x39a/0x560
Jul 13 15:03:37 localhost kernel: [ 9837.668532]  vfs_open+0x84/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668538]  path_openat+0x4ab/0x1e10
Jul 13 15:03:37 localhost kernel: [ 9837.668544]  do_filp_open+0x122/0x1c0
Jul 13 15:03:37 localhost kernel: [ 9837.668549]  do_sys_open+0x17c/0x2c0
Jul 13 15:03:37 localhost kernel: [ 9837.668554]  compat_SyS_open+0x1b/0x20
Jul 13 15:03:37 localhost kernel: [ 9837.668561]  do_fast_syscall_32+0x188/0x300
Jul 13 15:03:37 localhost kernel: [ 9837.668568]
entry_SYSENTER_compat+0x4c/0x5b
Jul 13 15:03:37 localhost kernel: [ 9837.668570] Freed:
Jul 13 15:03:37 localhost kernel: [ 9837.668575] PID = 6277
Jul 13 15:03:37 localhost kernel: [ 9837.668583]  save_stack_trace+0x1b/0x20
Jul 13 15:03:37 localhost kernel: [ 9837.668589]  save_stack+0x46/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668594]  kasan_slab_free+0x71/0xb0
Jul 13 15:03:37 localhost kernel: [ 9837.668599]  kfree+0x9e/0x1e0
Jul 13 15:03:37 localhost kernel: [ 9837.668605]  kernfs_fop_release+0x87/0xa0
Jul 13 15:03:37 localhost kernel: [ 9837.668611]  __fput+0x177/0x350
Jul 13 15:03:37 localhost kernel: [ 9837.668616]  ____fput+0xe/0x10
Jul 13 15:03:37 localhost kernel: [ 9837.668623]  task_work_run+0xa0/0xc0
Jul 13 15:03:37 localhost kernel: [ 9837.668629]
```

```
exit_to_usermode_loop+0xc5/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668635]  do_fast_syscall_32+0x2ef/0x300
Jul 13 15:03:37 localhost kernel: [ 9837.668642]
entry_SYSENTER_compat+0x4c/0x5b
Jul 13 15:03:37 localhost kernel: [ 9837.668644] Memory state around the buggy
address:
Jul 13 15:03:37 localhost kernel: [ 9837.668655]  ffff88007cdaa600: fc fc fc fc
fc fc fc fc fc fc fc fc fc fb fb fb
Jul 13 15:03:37 localhost kernel: [ 9837.668664]  ffff88007cdaa680: fb fb fb fb
fb fb fb fb fb fb fb fb fb fb fb fb
Jul 13 15:03:37 localhost kernel: [ 9837.668674] >ffff88007cdaa700: fb fb fb fb
fb fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:37 localhost kernel: [ 9837.668680]
^
Jul 13 15:03:37 localhost kernel: [ 9837.668689]  ffff88007cdaa780: fc fc fc fc
fc fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:37 localhost kernel: [ 9837.668698]  ffff88007cdaa800: fc fc fc fc
fc fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:37 localhost kernel: [ 9837.668704]
==================================================================
Jul 13 15:03:37 localhost NetworkManager[745]: <info>  [1499938417.1889]
address 192.168.1.100

------------------------------------------------------------------------------
-------------------------------------------------
Commits - 13 August 2017 - Suspicious use-after-free error flagged by Kernel
Address Sanitizer - committed for analysis
This error precedes last_sector_hacks ISRA error above in USB storage driver.
------------------------------------------------------------------------------
-------------------------------------------------
Aug 13 14:53:17 localhost kernel: [   47.797146] BUG: KASAN: use-after-free in
sr_probe+0x7e0/0xb20 at addr ffff88000009637e
Aug 13 14:53:17 localhost kernel: [   47.797146] Read of size 1 by task kworker/
u4:1/37
Aug 13 14:53:17 localhost kernel: [   47.797146] page:ffffea0000002580 count:0
mapcount:0 mapping:          (null) index:0x0
Aug 13 14:53:17 localhost kernel: [   47.797146] flags: 0x0()
Aug 13 14:53:17 localhost kernel: [   47.797146] raw: 0000000000000000
0000000000000000 0000000000000000 00000000ffffffff
Aug 13 14:53:17 localhost kernel: [   47.797146] raw: ffffea00000025a0
ffffea00000025a0 0000000000000000 0000000000000000
Aug 13 14:53:17 localhost kernel: [   47.797146] page dumped because: kasan: bad
access detected
Aug 13 14:53:17 localhost kernel: [   47.797146] CPU: 1 PID: 37 Comm:
kworker/u4:1 Tainted: G    B           4.10.3 #18
Aug 13 14:53:17 localhost kernel: [   47.797146] Hardware name: Dell Inc.
Inspiron 1545                   /0J037P, BIOS A14 12/07/2009
Aug 13 14:53:17 localhost kernel: [   47.797146] Workqueue: events_unbound
async_run_entry_fn
Aug 13 14:53:17 localhost kernel: [   47.797146] Call Trace:
Aug 13 14:53:17 localhost kernel: [   47.797146]  dump_stack+0x63/0x8b
Aug 13 14:53:17 localhost kernel: [   47.797146]
kasan_report.part.1+0x4bc/0x4f0
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? sr_probe+0x7e0/0xb20
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? scsi_mode_select+0x370/0x370
Aug 13 14:53:17 localhost kernel: [   47.797146]  kasan_report+0x25/0x30
Aug 13 14:53:17 localhost kernel: [   47.797146]  __asan_load1+0x47/0x50
Aug 13 14:53:17 localhost kernel: [   47.797146]  sr_probe+0x7e0/0xb20
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
kernfs_next_descendant_post+0x93/0xf0
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? sr_block_ioctl+0xe0/0xe0
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
sysfs_do_create_link_sd.isra.2+0x7c/0xc0
Aug 13 14:53:17 localhost kernel: [   47.797146]
```

```
driver_probe_device+0x40b/0x670
Aug 13 14:53:17 localhost kernel: [   47.797146]
__device_attach_driver+0xd9/0x160
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? __driver_attach+0x120/0x120
Aug 13 14:53:17 localhost kernel: [   47.797146]  bus_for_each_drv+0x107/0x180
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? bus_rescan_devices+0x20/0x20
Aug 13 14:53:17 localhost kernel: [   47.797146]  __device_attach+0x17e/0x200
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? device_bind_driver+0x80/0x80
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
kobject_uevent_env+0x1ec/0x7f0
Aug 13 14:53:17 localhost kernel: [   47.797146]  device_initial_probe+0x13/0x20
Aug 13 14:53:17 localhost kernel: [   47.797146]  bus_probe_device+0xfe/0x120
Aug 13 14:53:17 localhost kernel: [   47.797146]  device_add+0x5f1/0x9f0
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
device_private_init+0xc0/0xc0
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
scsi_dh_add_device+0xd4/0x130
Aug 13 14:53:17 localhost kernel: [   47.797146]  scsi_sysfs_add_sdev+0xd1/0x350
Aug 13 14:53:17 localhost kernel: [   47.797146]  do_scan_async+0xfd/0x230
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? scsi_scan_host+0x250/0x250
Aug 13 14:53:17 localhost kernel: [   47.797146]  async_run_entry_fn+0x84/0x270
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
pwq_dec_nr_in_flight+0x8c/0x110
Aug 13 14:53:17 localhost kernel: [   47.797146]  process_one_work+0x2c6/0x7d0
Aug 13 14:53:17 localhost kernel: [   47.797146]  worker_thread+0x90/0x850
Aug 13 14:53:17 localhost kernel: [   47.797146]  kthread+0x178/0x1d0

------------------------------------------------------------------------------
------------------------------------------------------
(FEATURE-DONE) Spark Cloud Analytics for Linux Kernel 4.10.3 64 bit with Kernel
Address Sanitizer debug logging enabled - 15 August 2017
- Commits 1
------------------------------------------------------------------------------
------------------------------------------------------
(*) Upgraded Spark version to 2.1.0 on Hadoop 2.7
(*) Changed to SparkContext text file instead of reading the input kernel log in
python I/O
(*) Added flatMap to front of MapReduce chain of transformations for tokenizer
(*) Changed the input kernel log to 64bit 4.10.3 Kernel Address Sanitizer
enabled kern.log which prints lot of debugging information on
memory accesses especially for USBWWAN and USB Storage drivers.
(*) This is an alternative to traditional promiscuous USB Analyzers like
WireShark to get kernel stack traces for USB and WLAN operations.
(*) Particularly useful in malware related untoward memory access and traffic
analysis
(*) Unifies Kernel Address Sanitizer, USB storage/WLAN driver and Spark Cloud
for analytics
(*) Logs for this have been committed to testlogs/ and python-src/testlogs

------------------------------------------------------------------------------
------------------------------------------------------
(FEATURE-DONE) Spark Cloud Analytics for Linux Kernel 4.10.3 64 bit with Kernel
Address Sanitizer debug logging enabled - 15 August 2017
- Commits 2
------------------------------------------------------------------------------
------------------------------------------------------
(*) Added a substring match filter to RDD map/reduce transformations chain
(*) Presently hardcoded as "+0x" which extracts all kernel functions invoked
from Kernel Address Sanitizer kern.log and their frequencies

Previous profiling prints following top kernel function invocations:
(u'last_sector_hacks.isra.1.part.2+0xc9/0x1d0', 159),
 (u'usb_stor_disconnect+0x120/0x120', 106),
 (u'save_stack+0x46/0xd0', 106),
```

```
 (u'save_stack_trace+0x1b/0x20', 106),
 (u'entry_SYSENTER_compat+0x4c/0x5b', 85),
 (u'kthread+0x178/0x1d0', 74),
```
implying heavy dependence on last_sector_hacks.isra gcc optimization. Discussion on https://groups.google.com/forum/#!topic/linux.kernel/IYBXrW7K2Vc shows it to be an old kernel bug.

USBWWAN Kernel Log Spark Analyzer Update - Refactoring to a new python function - 18 June 2018

---

1. Spark Log Analyzer Spark_USBWWANLogMapReduceParser.py has been changed to modularize the pattern extraction
by defining a new function accepting kern.log file, pattern and filter and also creates Spark DataFrame SQL
table and queries it.
2. This is similar to NeuronRain AsFer log_mapreducer()

(FEATURE) USBWWAN analytics - USBmon and FTrace logs analysis - 15 November 2018

---

1. Logs Analysis for 2 standard kernel tracing facilities have been included - USBmon and FTrace. USBmon is the
kernel debugfs tracing facility and FTrace is the Kernel functions tracing utility accessible from user space. (Kernel Address Sanitizer - KASAN - is only enabled in kernelspace via KBuild config and kernel build transparent to userspace)
2. USBmon traces are enabled by debugfs in /sys/kernel/debug/usb/usbmon and can be loaded in wireshark in libpcap format or usbmon pseudodevices can be viewed in tcpdump:

```
  467  ls /sys/kernel/debug/
  468  modprobe usbmon
  472  dumpcap -D
  474  ls /dev/usbmon0
  475  ls -lrt /dev/usbmon*
  487  tcpdump -i usbmon1
  488  tcpdump -i usbmon2
  489  tcpdump -i usbmon0
  490  tcpdump -i usbmon3
  491  tcpdump -i usbmon4
  520  cat /sys/kernel/debug/usb/usbmon/1t 2>&1 > usbmon.mon
```

3. FTrace for function graph analysis are enabled by (Kernel.org FTrace Documentation: https://www.kernel.org/doc/Documentation/trace/ftrace.txt):

```
  536  ls /sys/kernel/debug/tracing/current_tracer
  537  echo nop > /sys/kernel/debug/tracing/current_tracer
  538  echo 0 > /sys/kernel/debug/tracing/tracing_on
  539  echo $$ > /sys/kernel/debug/tracing/set_ftrace_pid
  541  echo function > /sys/kernel/debug/tracing/current_tracer
  545  echo 1 > /sys/kernel/debug/tracing/tracing_on
  557  ls -lrt /sys/kernel/debug/tracing/trace
  561  cat /sys/kernel/debug/tracing/set_graph_function
  562  cat /sys/kernel/debug/tracing/trace_options
  563  echo funcgraph-duration > /sys/kernel/debug/tracing/trace_options
  566  cat /sys/kernel/debug/tracing/set_graph_function
  567  cat /sys/kernel/debug/tracing/trace_options
  568  cat /sys/kernel/debug/tracing/trace_options
  569  echo funcgraph-cpu 2>&1 > /sys/kernel/debug/tracing/trace_options
  620  cat /sys/kernel/debug/tracing/set_ftrace_pid
  624  echo 7379 > /sys/kernel/debug/tracing/set_ftrace_pid
  625  cat /sys/kernel/debug/tracing/trace 2>&1 > ftrace.log.15November2018
```

```
  639  export JAVA_HOME=/media/Ubuntu2/jdk1.8.0_171/
  640  export PATH=/usr/bin:$PATH
  671  /media/Ubuntu2/spark-2.3.1-bin-hadoop2.7/bin/spark-submit
Spark_USBWWANLogMapReduceParser.py 2>&1 >
testlogs/Spark_USBWWANLogMapReduceParser.FTraceAndUSBMon.log.15November2018
```

4. FTrace traces for specific userspace threads/processes are enabled by previous example commandlines and available through /sys/kernel/debug/tracing/trace (circular buffer). Function graph traces show kernel function invocations as call graph edges (denoted by fn2 <- fn1)
5. Spark_USBWWANLogMapReduceParser.py has been changed to invoke log analyzer for USBmon and FTrace logs for patterns Bi(BULK IN) and usb from USBmon and FTrace logs respectively:
  - usbmon.15November2018.mon
  - ftrace.ping.log.15November2018 (ftraces for ping of an IP address)
6. Logs for Spark Analyzer have been committed to Spark_USBWWANLogMapReduceParser.FTraceAndUSBMon.log.15November2018 which analyze the USBmon logs and WLAN traffic for IP address ping.
7. include/linux/serial.h has been committed (similar to other versions of USBmd - 32 and 64 bits - in SourceForge,GitHub and GitLab)

(FEATURE) USBmd FTrace Kernel Function CallGraph Generation for Analysis - 22 November 2018

1.New bash shell script usb_md_ftrace.sh has been committed to repository which writes out an ftrace.log file containing kernel function call graph sequences for an executable code. It is invoked as:

```
        $usb_md_ftrace.sh <executable-to-trace>
```

usb_md_ftrace.sh summarizes previously mentioned ftrace options enabling commands into single file with an option for commandline argument of an executable to trace.
2.usb_wwan_modified/python-src/Spark_USBWWANLogMapReduceParser.py has been changed to include a new function ftrace_callgraph_dot() which parses an ftrace log generated by usb_md_ftrace.sh for command:

```
        $usb_md_ftrace.sh traceroute <ip-address>
```

3.ftrace_callgraph_dot() parses each line of ftrace.log and adds them as edges in a NetworkX Directed Graph. DOT file for this call graph is written to Spark_USBWWANLogMapReduceParser.ftrace_callgraph.dot
4.As a novelty, PageRank and Degree Centrality measures of the call graph NetworkX DiGraph are printed which show the prominently active regions of the kernel for traceroute . PageRank/Degree Centrality of kernel function callgraph is quite useful by treating every function caller as a voter to function callee. Theoretically, this centrality in kernel throws light on suspicious, malevolent invocations particularly involving memory and locking. In this traceroute ftrace example, lock and kmalloc functions have high centrality, and USB URB related functions are way down the ranking. More the ranking, deeper the function is in callstack trace in kernel.
5. Lot of functions have ISRA optimization of GCC. ISRA is known to cause signed int bugs (0 was erroneously promoted to 1 in loops) and ISRA has been disabled in ARM kernel: https://patchwork.kernel.org/patch/7113091/ by -fno-ipa-sra GCC flag. This kind of instability could be the reason for 32-bit VIRGO heisenbugs in string functions in older kernels.
6.Previous FTrace kernel call graph analysis is not only limited to USBmd WLAN analytics but can be applied to any executable requiring kernel profiling. Usual profilers measure time spent in the function whereas this graph theoretic analysis is superior and finds kernel bottlenecks and malicious patterns by analyzing call graphs within kernel.
7. Malicious code (e.g virus, worms, root-kits, bots, keystroke loggers) are usually associated with high cpu and memory footprint causing abnormal traffic.

Analyzing infected kernel callgraph patterns might help in identifying the root cause.
8. FTrace kernel function call graph complements already implemented Program Analyzers: SATURN CFG driver in VIRGO kernels (accessible only in kernelspace) and Valgrind/KCachegrind/Callgrind userspace call graph analyzer in AsFer. By this kernel activity is partially visible and can be analyzed graph theoretically from userspace.
9. Outbreak of epidemics have been analyzed as Game Theoretic problem (https://blogs.cornell.edu/info2040/2016/09/16/game-theory-in-the-context-of-epidemics/) - on how people behave in epidemics and their conclusion - "faster information limits disease spread". Cybercrimes are epidemics counterpart in cloud of computers only difference being damage inflicted on intellectual property than humans and adversaries are hackers/malicious code in place of viri. This makes Cybercrimes a multi-player adversarial game involving Hackers/Malicious code Versus Aggrieved. Translating the previous conclusion to cybercrimes: Faster information about malicious code limits the damage.
#/
****
#* NEURONRAIN USB-md - Wireless Network and USB Stream Data Analytics
#* This program is free software: you can redistribute it and/or modify
#* it under the terms of the GNU General Public License as published by
#* the Free Software Foundation, either version 3 of the License, or
#* (at your option) any later version.
#*
#* This program is distributed in the hope that it will be useful,
#* but WITHOUT ANY WARRANTY; without even the implied warranty of
#* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#* GNU General Public License for more details.
#*
#* You should have received a copy of the GNU General Public License
#* along with this program. If not, see <http://www.gnu.org/licenses/>.
#*
#----------
K.Srinivasan
#NeuronRain Documentation and Licensing: http://neuronrain-documentation.readthedocs.io/en/latest/
#Personal website(research): https://sites.google.com/site/kuja27/
#----------
#**********/

USBmd driver is an experimental modified version of already existing USB driver in linux.

Purpose of this modified version is for doing more sophisticated debugging of USB endpoints and devices and as
USB packet sniffer. Technical Necessity for this was created due to prolonged data theft, id spoofing and cybercrime that has been happening
in author's personal electronic devices for years that resulted in a Cybercrime Police Complaint also few years ago.

There were also such incidents while developing open source code (some code commits have description of these mysterious occurrences). There is no comprehensive USB debugger available on linux to sift bad traffic though there are strong evidences of such cybercrime and datatheft through other sources. Author is inclined to believe that such recurring events of datatheft that defies all logic can have no other intent but to cause malafide theft or loss of private data and an act of defamation among other things.

This is also done as a technical learning exercise to analyze USB Hosts, packets

and USB's interaction,if any, with wireless devices including
mobiles, wireless LANs(radiotap) etc.,

In the longterm USBmd might have to be integrated into VIRGO. As VIRGO would would have the synergy of AstroInfer machine learning
codebase for "learning" from datasets, this USBmd driver can have the added ability of analyzing large USB traffic (as a dataset)
using some decision making algorithms and evolve as an anti-cybercrime, anti-plagiarism and anti-theft tool to single out
"malevolent" traffic that would save individuals and organisations from the travails of tampering and loss of sensitive confidential data.

The pattern mining of numeric dataset designed for AstroInfer can apply here also since USB bitstream can be analyzed using algorithms for
numerical dataset mining. Also Discrete Fourier Transform used for analyzing data for frequencies (periodicities if any) can be used for
USB data , for example USB wireless traffic.

==========
new UMB driver bind - 27 Feb 2014 (for Bus id 7)
==========
Following example commandlines install umb.ko module, unbind the existing option driver from bus-device id and bind the umb.ko to that bus id:

```
sudo insmod umb.ko
echo -n "7-1:1.0" > /sys/bus/usb/drivers/option/unbind
echo -n "7-1:1.0" > /sys/bus/usb/drivers/umb/bind
```

==========
1055. Commits as on 29 July 2014
==========
Driver has been ported and built on 3.15.5 kernel. Also a driver build script has been committed.

USBmd version 14.9.9 has been release tagged on 9 September 2014

USBmd version 15.1.8 has been release tagged on 8 January 2015
---

http://sourceforge.net/p/usb-md/code-0/HEAD/tree/Adding%20new%20vendor%20and%20product%20IDs%20to%20an%20existing%20USB%20driver%20on%20Linux.html has steps to add new vendor-id.

1056. USB debug messages from "cat /sys/kernel/debug/usb/devices" for UMB bound above:

```
T:  Bus=07 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#= 12 Spd=12   MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=12d1 ProdID=140b Rev= 0.00
S:  Manufacturer=HUAÿWEI TECHNOLOGIES
S:  Product=HUAWEI Mobile
S:  SerialNumber=ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
C:* #Ifs= 4 Cfg#= 1 Atr=a0 MxPwr=500mA
I:* If#= 0 Alt= 0 #EPs= 3 Cls=ff(vend.) Sub=ff Prot=ff Driver=umb
E:  Ad=81(I) Atr=03(Int.) MxPS=  16 Ivl=128ms
E:  Ad=82(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=02(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
I:* If#= 1 Alt= 0 #EPs= 2 Cls=ff(vend.) Sub=ff Prot=ff Driver=option
E:  Ad=84(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=04(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
```

```
I:* If#= 2 Alt= 0 #EPs= 2 Cls=ff(vend.) Sub=ff Prot=ff Driver=option
E:  Ad=86(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=06(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
I:* If#= 3 Alt= 0 #EPs= 2 Cls=08(stor.) Sub=06 Prot=50 Driver=usb-storage
E:  Ad=87(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=08(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms

-----------------------------------------------------------------------
1057. usbmon, libpcap tcpdump and wireshark (or vusb-analyzer) debugging
-----------------------------------------------------------------------
*mount -t debugfs none_debugs /sys/kernel/debug
*modprobe usbmon
*ls /sys/kernel/debug/usb/usbmon/

0s  0u      1s  1t      1u  2s      2t  2u      3s  3t      3u  4s      4t  4u
      5s  5t      5u  6s      6t  6u      7s  7t      7u  8s      8t  8u

*cat /sys/kernel/debug/usb/usbmon/8t > usbmon.mon (any of the above usbmon debug
logs)
*vusb-analyzer usbmon.mon

ef728540 3811287714 S Ci:001:00 s a3 00 0000 0001 0004 4 <
ef728540 3811287743 C Ci:001:00 0 4 = 00010000
ef728540 3811287752 S Ci:001:00 s a3 00 0000 0002 0004 4 <
ef728540 3811287763 C Ci:001:00 0 4 = 00010000
f50f6540 3811287770 S Ii:001:01 -115 2 <
f50f6540 3811287853 C Ii:001:01 -2 0
f5390540 3814543695 S Ci:001:00 s a3 00 0000 0001 0004 4 <
f5390540 3814543715 C Ci:001:00 0 4 = 00010000
f5390540 3814543756 S Ci:001:00 s a3 00 0000 0002 0004 4 <
f5390540 3814543767 C Ci:001:00 0 4 = 00010000
f50f6540 3814543805 S Ii:001:01 -115 2 <

*modprobe usbmon
*ls /dev/usbmon[1-8]
*tcpdump -i usbmon1 -w usbmon.pcap
tcpdump: listening on usbmon1, link-type USB_LINUX_MMAPPED (USB with padded
Linux header), capture size 65535 bytes
^C86 packets captured
86 packets received by filter

*wireshark usbmon.pcap (loads on wireshark)

-----------------------------------------------------------
1058. Dynamic Debug - dev_dbg() and dev_vdbg()
-----------------------------------------------------------

USB Debugging References:
-------------------------
- Texas Instruments -
http://elinux.org/images/1/17/USB_Debugging_and_Profiling_Techniques.pdf

-----------------------------------------------------------
NeuronRain version 15.6.15 release tagged
-----------------------------------------------------------

-----------------------------------------------------------
1059. Commits as on 11 July 2015
-----------------------------------------------------------
usbmd kernel module has been ported to Linux Kernel 4.0.5

-----------------------------------------------------------
1060. Commits as on 26 November 2015
-----------------------------------------------------------
```

- Updated USB-md driver with a lookup of VIRGO kernel_analytics config variable exported by kernel_analytics module in umb_read() as default.
- New header file umb.h has been added that externs the VIRGO kernel_analytics config array variables
- Module.symvers has been imported from VIRGO kernel_analytics and clean target has been commented in build script after initial build as make clean removes Module.symvers.
- kern.log with umb_read() and umb_write() have been added with following commandlines:
      - cat /dev/umb0 - invokes umb_read() but there are kernel panics sometimes
      - cat <file> > /dev/umb0 - invokes umb_write()
  where umb0 is usb-md device name registered with /sys/bus/usb as below:
      - insmod umb.ko
      - echo -n "7-1:1.0" > /sys/bus/usb/drivers/option/unbind
      - echo -n "7-1:1.0" > /sys/bus/usb/drivers/umb/bind
- Updated build generated sources and object files have been added

---
1061. Commits as on 27 November 2015

New folder usb_wwan_modified has been added that contains the USB serial, option and wireless USB modem WWAN drivers from kernel mainline
instrumented with lot of printk()s so that log messages are written to kern.log. Though dev_dbg dynamic debugging can be used by writing to /sys/kernel/debug/<...>/dynamic_debug
printk()s are sufficient for now. This traces through the USB connect and data transfer code:
        - probe
        - buffer is copied from userspace to kernelspace
        - URB is allocated in kernel
        - buffer is memcopied to URB
        - usb send/receive bulk pipe calls
        - usb_fill_bulk_urb
Almost all buffers like in and out buffers in URBs, portdata, interfacedata, serial_data, serial_port_data are printed to kern.log. This log is analyzable by AsFer machine learning code for USB debugging similar to usbmon logs.

These are initial commits only and usb-serial.c, usb_wwan.c, option.c and serial.h might be significantly altered going forward.

1062. Commits as on 30 November 2015

Added usb.h from kernel mainline, instrumented with printk() to print transfer_buffer in usb_fill_[control/bulk/interrupt]_urb() functions. kern.log for this has been added in usb_wwan_modified/testlogs.

1063. Commits as on 1 December 2015

- new kernel function print_buffer() has been added in usb.h that prints contents of char buffer in hex
- Above print_buffer() is invoked to print transfer_buffer in usb_wwan.c, usb-serial.c, option.c
- kern.log with print_buffer() output has been added - This dumps similar to wireshark, usbmon and other usb analyzers.

1064. Commits as on 2 December 2015

- changed print_buffer() printk() to print a delimiter in each byte for AsFer Machine Learning code processing
- add a parser script for kern.log to print print_buffer() lines

- parsed kern.log with print_buffer() lines has been added
- Added an Apache Spark MapReduce python script to compute byte frequency in parsed print_buffer() kern.log

---
879. (FEATURE) NeuronRain USBmd Debug and Malafide Traffic Analytics
---
As mentioned in commit notes above, USB incoming and outgoing data transfer_buffer are dumped byte-by-byte. Given this data various analytics can be performed most of which are already implemented in AsFer codebase:
- frequency of bytes
- most frequent sequence of bytes
- bayesian and decision tree inference
- deep learning
- perceptrons
- streaming algorithms for USB data stream
and so on.

1065. Commits as on 3 December 2015

- Apache Spark script for analyzing the USBWWAN byte stream logs has been updated with byte counts map-reduce functions from print_buffer() logs and temp DataFrame Table creation with SparkSQL.
- logs for the script have been added in usb_wwan_modified/python-src/testlogs/Spark_USBWWANLogMapReduceParser.out.3December2015
- kern.log parser shellscript has been updated

1066. AsFer commits for USBmd as on 4 December 2015

All the Streaming_<>.py Streaming Algorithm implementations in AsFer/python-src/ have been updated with:
- hashlib ripemd160 hash MD algorithm for hash functions and return hexdigest()
- USBWWAN byte stream data from USBmd print_buffer() logs in usb-md/usb_wwan_modified/testlogs/ has been added as a Data Storage and Data Source
- logs for the above have been added to asfer/python-src/testlogs/
- Streaming Abstract Generator has been updated with USB stream data iterable and parametrized for data source and storage
- Some corrections to the asfer/python-src/Streaming_<> scripts

---
1067. Commits as on 7 December 2015

- added Spark Mapreduce and DataFrame log for USBWWAN byte stream
- added a parsed kern.log with only bytes from USBWWAN stream
- Added dict() and sort() for query results and printed cardinality of the stream data set which is the size of the dict.
An example log has been added which prints the cardinality as ~250. In contrast, LogLog and HyperLogLog counter estimations
approximate the cardinality to 140 and 110 respectively

---
880. (FEATURE) AsFer commits for USBmd as on 11 December 2015 - USBWWAN stream data backend in MongoDB

 Dependency Injection code commits for MongoDB backend - With this MongoDB is also a storage backend for AsFer algorithms similar to MySQL:
- Abstract_DBBackend.py has been updated for both MySQL and MongoDB injections

- MongoDB configuration and backend connect/query code has been added. Backend is either populated by Robomongo or pymongo reading from the
Streaming Abstract Generator iterable framework.
- With this AsFer supports both SQL(MySQL) and NoSQL(file,hive,hbase,cassandra backends in Streaming Abstract Generator).
- log with a simple NoSQL table with StreamingData.txt and USBWWAN data has been added to testlogs/.
- MongoDB configuration has a database(asfer-database) and a collection(asfer-collection).
- MongoDB_DBBackend @provides pymongo.collection.Collection which is @inject-ed to Abstract_DBBackend

---
1068. Commits as on 10 January 2016

NeuronRain USBmd enterprise version 2016.1.10 released.

1069. Commits - 4 August 2016

1.New build script for drivers/usb top level folder has been added.
2.Copyleft notices updated
3.print_buffer() in usb.h has been #ifdef-ed based on a build time flag to suppress the buffer bytes dump preferentially so that
kern.log is not flooded.
4.Flag PRINT_BUFFER has to be defined with #define somewhere within KBuild makefiles or externally.
5..ko files rebuilt
6. Miscellaneous code changes to suppress kbuild warnings - cast etc.,
7. PRINT_BUFFER block changed to print the bytes in single line for each buffer

---
1070. Commits - 13 July 2017 - usb-storage driver last sector access slab out of bounds error in 64-bit - committed for analysis
- this error was frequently witnessed in VIRGO 32-bit stability issues and panics - ISRA looks like a GCC
optimization of a function invocation (Interprocedural Scalar Replacement of Aggregates)

Jul 13 15:03:36 localhost kernel: [9837.497280]
==
Jul 13 15:03:36 localhost kernel: [9837.499787]
==
Jul 13 15:03:36 localhost kernel: [9837.499822] BUG: KASAN: slab-out-of-bounds in last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage] at addr ffff88007cdaa758
Jul 13 15:03:36 localhost kernel: [9837.499831] Read of size 8 by task usb-storage/6243
Jul 13 15:03:36 localhost kernel: [9837.499844] CPU: 0 PID: 6243 Comm: usb-storage Tainted: G   B   4.10.3 #18
Jul 13 15:03:36 localhost kernel: [9837.499849] Hardware name: Dell Inc. Inspiron 1545   /0J037P, BIOS A14 12/07/2009
Jul 13 15:03:36 localhost kernel: [9837.499851] Call Trace:
Jul 13 15:03:36 localhost kernel: [9837.499863] dump_stack+0x63/0x8b
Jul 13 15:03:36 localhost kernel: [9837.499870] kasan_object_err+0x21/0x70
Jul 13 15:03:36 localhost kernel: [9837.499877] kasan_report.part.1+0x219/0x4f0
Jul 13 15:03:36 localhost kernel: [9837.499893] ?

```
last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.499899]  kasan_report+0x25/0x30
Jul 13 15:03:36 localhost kernel: [ 9837.499906]  __asan_load8+0x5e/0x70
Jul 13 15:03:36 localhost kernel: [ 9837.499922]
last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.499938]
usb_stor_invoke_transport+0x1a1/0x960 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.499946]  ?
migrate_swap_stop+0x2e0/0x2e0
Jul 13 15:03:36 localhost kernel: [ 9837.499963]  ?
usb_stor_port_reset+0xb0/0xb0 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.499973]  ?
wait_for_completion_interruptible+0x1a7/0x260
Jul 13 15:03:36 localhost kernel: [ 9837.499981]  ?
wait_for_completion_killable+0x2a0/0x2a0
Jul 13 15:03:36 localhost kernel: [ 9837.499989]  ?
raise_softirq_irqoff+0xba/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.499995]  ? wake_up_q+0x80/0x80
Jul 13 15:03:36 localhost kernel: [ 9837.500011]
usb_stor_transparent_scsi_command+0xe/0x10 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
usb_stor_control_thread+0x344/0x510 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ?
usb_stor_disconnect+0x120/0x120 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ?
default_wake_function+0x2f/0x40
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ? __wake_up_common+0x78/0xc0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kthread+0x178/0x1d0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ?
usb_stor_disconnect+0x120/0x120 [usb_storage]
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ?
kthread_create_on_node+0xd0/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ret_from_fork+0x2c/0x40
Jul 13 15:03:36 localhost kernel: [ 9837.500017] Object at ffff88007cdaa668, in
cache kmalloc-192 size: 192
Jul 13 15:03:36 localhost kernel: [ 9837.500017] Allocated:
Jul 13 15:03:36 localhost kernel: [ 9837.500017] PID = 6277
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  save_stack_trace+0x1b/0x20
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  save_stack+0x46/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kasan_kmalloc+0xad/0xe0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
kmem_cache_alloc_trace+0xef/0x210
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kernfs_fop_open+0x14b/0x540
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  do_dentry_open+0x39a/0x560
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  vfs_open+0x84/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  path_openat+0x4ab/0x1e10
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  do_filp_open+0x122/0x1c0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  do_sys_open+0x17c/0x2c0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  compat_SyS_open+0x1b/0x20
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  do_fast_syscall_32+0x188/0x300
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
entry_SYSENTER_compat+0x4c/0x5b
Jul 13 15:03:36 localhost kernel: [ 9837.500017] Freed:
Jul 13 15:03:36 localhost kernel: [ 9837.500017] PID = 6277
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  save_stack_trace+0x1b/0x20
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  save_stack+0x46/0xd0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kasan_slab_free+0x71/0xb0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kfree+0x9e/0x1e0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  kernfs_fop_release+0x87/0xa0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  __fput+0x177/0x350
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ____fput+0xe/0x10
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  task_work_run+0xa0/0xc0
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
exit_to_usermode_loop+0xc5/0xd0
```

```
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  do_fast_syscall_32+0x2ef/0x300
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
entry_SYSENTER_compat+0x4c/0x5b
Jul 13 15:03:36 localhost kernel: [ 9837.500017] Memory state around the buggy
address:
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ffff88007cdaa600: fc fc fc fc
fc fc fc fc fc fc fc fc fc fb fb fb
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ffff88007cdaa680: fb fb fb fb
fb fb fb fb fb fb fb fb fb fb fb fb
Jul 13 15:03:36 localhost kernel: [ 9837.500017] >ffff88007cdaa700: fb fb fb fb
fb fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
^
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ffff88007cdaa780: fc fc fc fc
fc fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:36 localhost kernel: [ 9837.500017]  ffff88007cdaa800: fc fc fc fc
fc fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:36 localhost kernel: [ 9837.500017]
==================================================================
Jul 13 15:03:37 localhost kernel: [ 9837.668157]
==================================================================
Jul 13 15:03:37 localhost kernel: [ 9837.668191] BUG: KASAN: slab-out-of-bounds
in last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage] at addr
ffff88007cdaa758
Jul 13 15:03:37 localhost kernel: [ 9837.668200] Read of size 8 by task usb-
storage/6243
Jul 13 15:03:37 localhost kernel: [ 9837.668213] CPU: 1 PID: 6243 Comm: usb-
storage Tainted: G    B           4.10.3 #18
Jul 13 15:03:37 localhost kernel: [ 9837.668218] Hardware name: Dell Inc.
Inspiron 1545                   /0J037P, BIOS A14 12/07/2009
Jul 13 15:03:37 localhost kernel: [ 9837.668220] Call Trace:
Jul 13 15:03:37 localhost kernel: [ 9837.668233]  dump_stack+0x63/0x8b
Jul 13 15:03:37 localhost kernel: [ 9837.668240]  kasan_object_err+0x21/0x70
Jul 13 15:03:37 localhost kernel: [ 9837.668247]
kasan_report.part.1+0x219/0x4f0
Jul 13 15:03:37 localhost kernel: [ 9837.668263]  ?
last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668269]  kasan_report+0x25/0x30
Jul 13 15:03:37 localhost kernel: [ 9837.668277]  __asan_load8+0x5e/0x70
Jul 13 15:03:37 localhost kernel: [ 9837.668292]
last_sector_hacks.isra.1.part.2+0xc9/0x1d0 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668308]
usb_stor_invoke_transport+0x1a1/0x960 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668316]  ?
migrate_swap_stop+0x2e0/0x2e0
Jul 13 15:03:37 localhost kernel: [ 9837.668332]  ?
usb_stor_port_reset+0xb0/0xb0 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668343]  ?
wait_for_completion_interruptible+0x1a7/0x260
Jul 13 15:03:37 localhost kernel: [ 9837.668351]  ?
wait_for_completion_killable+0x2a0/0x2a0
Jul 13 15:03:37 localhost kernel: [ 9837.668360]  ?
raise_softirq_irqoff+0xba/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668366]  ? wake_up_q+0x80/0x80
Jul 13 15:03:37 localhost kernel: [ 9837.668382]
usb_stor_transparent_scsi_command+0xe/0x10 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668398]
usb_stor_control_thread+0x344/0x510 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668415]  ?
usb_stor_disconnect+0x120/0x120 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668422]  ?
default_wake_function+0x2f/0x40
Jul 13 15:03:37 localhost kernel: [ 9837.668430]  ? __wake_up_common+0x78/0xc0
Jul 13 15:03:37 localhost kernel: [ 9837.668436]  kthread+0x178/0x1d0
```

```
Jul 13 15:03:37 localhost kernel: [ 9837.668454]  ?
usb_stor_disconnect+0x120/0x120 [usb_storage]
Jul 13 15:03:37 localhost kernel: [ 9837.668460]  ?
kthread_create_on_node+0xd0/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668466]  ret_from_fork+0x2c/0x40
Jul 13 15:03:37 localhost kernel: [ 9837.668472] Object at ffff88007cdaa668, in
cache kmalloc-192 size: 192
Jul 13 15:03:37 localhost kernel: [ 9837.668478] Allocated:
Jul 13 15:03:37 localhost kernel: [ 9837.668483] PID = 6277
Jul 13 15:03:37 localhost kernel: [ 9837.668494]  save_stack_trace+0x1b/0x20
Jul 13 15:03:37 localhost kernel: [ 9837.668500]  save_stack+0x46/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668506]  kasan_kmalloc+0xad/0xe0
Jul 13 15:03:37 localhost kernel: [ 9837.668513]
kmem_cache_alloc_trace+0xef/0x210
Jul 13 15:03:37 localhost kernel: [ 9837.668520]  kernfs_fop_open+0x14b/0x540
Jul 13 15:03:37 localhost kernel: [ 9837.668527]  do_dentry_open+0x39a/0x560
Jul 13 15:03:37 localhost kernel: [ 9837.668532]  vfs_open+0x84/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668538]  path_openat+0x4ab/0x1e10
Jul 13 15:03:37 localhost kernel: [ 9837.668544]  do_filp_open+0x122/0x1c0
Jul 13 15:03:37 localhost kernel: [ 9837.668549]  do_sys_open+0x17c/0x2c0
Jul 13 15:03:37 localhost kernel: [ 9837.668554]  compat_SyS_open+0x1b/0x20
Jul 13 15:03:37 localhost kernel: [ 9837.668561]  do_fast_syscall_32+0x188/0x300
Jul 13 15:03:37 localhost kernel: [ 9837.668568]
entry_SYSENTER_compat+0x4c/0x5b
Jul 13 15:03:37 localhost kernel: [ 9837.668570] Freed:
Jul 13 15:03:37 localhost kernel: [ 9837.668575] PID = 6277
Jul 13 15:03:37 localhost kernel: [ 9837.668583]  save_stack_trace+0x1b/0x20
Jul 13 15:03:37 localhost kernel: [ 9837.668589]  save_stack+0x46/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668594]  kasan_slab_free+0x71/0xb0
Jul 13 15:03:37 localhost kernel: [ 9837.668599]  kfree+0x9e/0x1e0
Jul 13 15:03:37 localhost kernel: [ 9837.668605]  kernfs_fop_release+0x87/0xa0
Jul 13 15:03:37 localhost kernel: [ 9837.668611]  __fput+0x177/0x350
Jul 13 15:03:37 localhost kernel: [ 9837.668616]  ____fput+0xe/0x10
Jul 13 15:03:37 localhost kernel: [ 9837.668623]  task_work_run+0xa0/0xc0
Jul 13 15:03:37 localhost kernel: [ 9837.668629]
exit_to_usermode_loop+0xc5/0xd0
Jul 13 15:03:37 localhost kernel: [ 9837.668635]  do_fast_syscall_32+0x2ef/0x300
Jul 13 15:03:37 localhost kernel: [ 9837.668642]
entry_SYSENTER_compat+0x4c/0x5b
Jul 13 15:03:37 localhost kernel: [ 9837.668644] Memory state around the buggy
address:
Jul 13 15:03:37 localhost kernel: [ 9837.668655]  ffff88007cdaa600: fc fc fc fc
fc fc fc fc fc fc fc fc fc fb fb fb
Jul 13 15:03:37 localhost kernel: [ 9837.668664]  ffff88007cdaa680: fb fb fb fb
fb fb fb fb fb fb fb fb fb fb fb fb
Jul 13 15:03:37 localhost kernel: [ 9837.668674] >ffff88007cdaa700: fb fb fb fb
fb fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:37 localhost kernel: [ 9837.668680]
^
Jul 13 15:03:37 localhost kernel: [ 9837.668689]  ffff88007cdaa780: fc fc fc fc
fc fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:37 localhost kernel: [ 9837.668698]  ffff88007cdaa800: fc fc fc fc
fc fc fc fc fc fc fc fc fc fc fc fc
Jul 13 15:03:37 localhost kernel: [ 9837.668704]
==================================================================
Jul 13 15:03:37 localhost NetworkManager[745]: <info>  [1499938417.1889]
address 192.168.1.100

--------------------------------------------------------------------------------
--------------------------------------------------
1071. Commits - 13 August 2017 - Suspicious use-after-free error flagged by
Kernel Address Sanitizer - committed for analysis
This error precedes last_sector_hacks ISRA error above in USB storage driver.
--------------------------------------------------------------------------------
```

```
-------------------------------------------------
Aug 13 14:53:17 localhost kernel: [   47.797146] BUG: KASAN: use-after-free in
sr_probe+0x7e0/0xb20 at addr ffff88000009637e
Aug 13 14:53:17 localhost kernel: [   47.797146] Read of size 1 by task kworker/
u4:1/37
Aug 13 14:53:17 localhost kernel: [   47.797146] page:ffffea0000002580 count:0
mapcount:0 mapping:          (null) index:0x0
Aug 13 14:53:17 localhost kernel: [   47.797146] flags: 0x0()
Aug 13 14:53:17 localhost kernel: [   47.797146] raw: 0000000000000000
0000000000000000 0000000000000000 00000000ffffffff
Aug 13 14:53:17 localhost kernel: [   47.797146] raw: ffffea00000025a0
ffffea00000025a0 0000000000000000 0000000000000000
Aug 13 14:53:17 localhost kernel: [   47.797146] page dumped because: kasan: bad
access detected
Aug 13 14:53:17 localhost kernel: [   47.797146] CPU: 1 PID: 37 Comm:
kworker/u4:1 Tainted: G    B           4.10.3 #18
Aug 13 14:53:17 localhost kernel: [   47.797146] Hardware name: Dell Inc.
Inspiron 1545                   /0J037P, BIOS A14 12/07/2009
Aug 13 14:53:17 localhost kernel: [   47.797146] Workqueue: events_unbound
async_run_entry_fn
Aug 13 14:53:17 localhost kernel: [   47.797146] Call Trace:
Aug 13 14:53:17 localhost kernel: [   47.797146]  dump_stack+0x63/0x8b
Aug 13 14:53:17 localhost kernel: [   47.797146]
kasan_report.part.1+0x4bc/0x4f0
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? sr_probe+0x7e0/0xb20
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? scsi_mode_select+0x370/0x370
Aug 13 14:53:17 localhost kernel: [   47.797146]  kasan_report+0x25/0x30
Aug 13 14:53:17 localhost kernel: [   47.797146]  __asan_load1+0x47/0x50
Aug 13 14:53:17 localhost kernel: [   47.797146]  sr_probe+0x7e0/0xb20
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
kernfs_next_descendant_post+0x93/0xf0
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? sr_block_ioctl+0xe0/0xe0
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
sysfs_do_create_link_sd.isra.2+0x7c/0xc0
Aug 13 14:53:17 localhost kernel: [   47.797146]
driver_probe_device+0x40b/0x670
Aug 13 14:53:17 localhost kernel: [   47.797146]
__device_attach_driver+0xd9/0x160
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? __driver_attach+0x120/0x120
Aug 13 14:53:17 localhost kernel: [   47.797146]  bus_for_each_drv+0x107/0x180
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? bus_rescan_devices+0x20/0x20
Aug 13 14:53:17 localhost kernel: [   47.797146]  __device_attach+0x17e/0x200
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? device_bind_driver+0x80/0x80
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
kobject_uevent_env+0x1ec/0x7f0
Aug 13 14:53:17 localhost kernel: [   47.797146]  device_initial_probe+0x13/0x20
Aug 13 14:53:17 localhost kernel: [   47.797146]  bus_probe_device+0xfe/0x120
Aug 13 14:53:17 localhost kernel: [   47.797146]  device_add+0x5f1/0x9f0
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
device_private_init+0xc0/0xc0
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
scsi_dh_add_device+0xd4/0x130
Aug 13 14:53:17 localhost kernel: [   47.797146]  scsi_sysfs_add_sdev+0xd1/0x350
Aug 13 14:53:17 localhost kernel: [   47.797146]  do_scan_async+0xfd/0x230
Aug 13 14:53:17 localhost kernel: [   47.797146]  ? scsi_scan_host+0x250/0x250
Aug 13 14:53:17 localhost kernel: [   47.797146]  async_run_entry_fn+0x84/0x270
Aug 13 14:53:17 localhost kernel: [   47.797146]  ?
pwq_dec_nr_in_flight+0x8c/0x110
Aug 13 14:53:17 localhost kernel: [   47.797146]  process_one_work+0x2c6/0x7d0
Aug 13 14:53:17 localhost kernel: [   47.797146]  worker_thread+0x90/0x850
Aug 13 14:53:17 localhost kernel: [   47.797146]  kthread+0x178/0x1d0

-------------------------------------------------------------------------------
-------------------------------------------------------
```

881. (FEATURE-DONE) Spark Cloud Analytics for Linux Kernel 4.10.3 64 bit with Kernel Address Sanitizer debug logging enabled - 15 August 2017
- Commits 1

(*) Upgraded Spark version to 2.1.0 on Hadoop 2.7
(*) Changed to SparkContext text file instead of reading the input kernel log in python I/O
(*) Added flatMap to front of MapReduce chain of transformations for tokenizer
(*) Changed the input kernel log to 64bit 4.10.3 Kernel Address Sanitizer enabled kern.log which prints lot of debugging information on
memory accesses especially for USBWWAN and USB Storage drivers.
(*) This is an alternative to traditional promiscuous USB Analyzers like WireShark to get kernel stack traces for USB and WLAN operations.
(*) Particularly useful in malware related untoward memory access and traffic analysis
(*) Unifies Kernel Address Sanitizer, USB storage/WLAN driver and Spark Cloud for analytics
(*) Logs for this have been committed to testlogs/ and python-src/testlogs

---
882. (FEATURE-DONE) Spark Cloud Analytics for Linux Kernel 4.10.3 64 bit with Kernel Address Sanitizer debug logging enabled - 15 August 2017
- Commits 2

(*) Added a substring match filter to RDD map/reduce transformations chain
(*) Presently hardcoded as "+0x" which extracts all kernel functions invoked from Kernel Address Sanitizer kern.log and their frequencies

Previous profiling prints following top kernel function invocations:
(u'last_sector_hacks.isra.1.part.2+0xc9/0x1d0', 159),
 (u'usb_stor_disconnect+0x120/0x120', 106),
 (u'save_stack+0x46/0xd0', 106),
 (u'save_stack_trace+0x1b/0x20', 106),
 (u'entry_SYSENTER_compat+0x4c/0x5b', 85),
 (u'kthread+0x178/0x1d0', 74),
implying heavy dependence on last_sector_hacks.isra gcc optimization. Discussion on https://groups.google.com/forum/#!topic/linux.kernel/IYBXrW7K2Vc shows it to be an old kernel bug.

---
883. (FEATURE-DONE) Commits - 24 September 2017 - USB-md driver for USB and Wireless LAN analytics for 4.13.3 64-bit kernel
---
(*) USB-md driver in GitHub and SourceForge at present are 32-bit based on mainline 4.1.5 kernel
(*) Both USB-md and KingCobra kernel modules are subsidiaries of VIRGO kernel
(*) There is a necessity for 64-bit version of USB-md for interoperability to VIRGO64 64-bit kernel on mainline version 4.13.3
(*) This requires separate repository for USB-md because of significant kernel function changes between 4.1.5 and 4.13.3 and
 idiosyncrasies of 64-bit
(*) USB-md driver has been rebuilt on 4.13.3 64-bit kernel after some changes to function prototypes and new usb-md64 repository is
 initialized with these commits

---
884. USBWWAN Kernel Log Spark Analyzer Update - Refactoring to a new python function - 18 June 2018

---
1. Spark Log Analyzer Spark_USBWWANLogMapReduceParser.py has been changed to modularize the pattern extraction
by defining a new function accepting kern.log file, pattern and filter and also creates Spark DataFrame SQL
table and queries it.
2. This is similar to NeuronRain AsFer log_mapreducer()

---
769. (FEATURE) USBWWAN analytics - USBmon and FTrace logs analysis - 15 November 2018 - this section is an extended draft on respective topics in NeuronRain AstroInfer design -
https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
---
1. Logs Analysis for 2 standard kernel tracing facilities have been included - USBmon and FTrace. USBmon is the
kernel debugfs tracing facility and FTrace is the Kernel functions tracing utility accessible from user space. (Kernel Address Sanitizer - KASAN - is only enabled in kernelspace via KBuild config and kernel build transparent to userspace)
2. USBmon traces are enabled by debugfs in /sys/kernel/debug/usb/usbmon and can be loaded in wireshark in libpcap format or usbmon pseudodevices can be viewed in tcpdump:
467 ls /sys/kernel/debug/
468 modprobe usbmon
472 dumpcap -D
474 ls /dev/usbmon0
475 ls -lrt /dev/usbmon*
487 tcpdump -i usbmon1
488 tcpdump -i usbmon2
489 tcpdump -i usbmon0
490 tcpdump -i usbmon3
491 tcpdump -i usbmon4
520 cat /sys/kernel/debug/usb/usbmon/1t 2>&1 > usbmon.mon
3. FTrace for function graph analysis are enabled by (Kernel.org FTrace Documentation: https://www.kernel.org/doc/Documentation/trace/ftrace.txt):
536 ls /sys/kernel/debug/tracing/current_tracer
537 echo nop > /sys/kernel/debug/tracing/current_tracer
538 echo 0 > /sys/kernel/debug/tracing/tracing_on
539 echo $$ > /sys/kernel/debug/tracing/set_ftrace_pid
541 echo function > /sys/kernel/debug/tracing/current_tracer
545 echo 1 > /sys/kernel/debug/tracing/tracing_on
557 ls -lrt /sys/kernel/debug/tracing/trace
561 cat /sys/kernel/debug/tracing/set_graph_function
562 cat /sys/kernel/debug/tracing/trace_options
563 echo funcgraph-duration > /sys/kernel/debug/tracing/trace_options
566 cat /sys/kernel/debug/tracing/set_graph_function
567 cat /sys/kernel/debug/tracing/trace_options
568 cat /sys/kernel/debug/tracing/trace_options
569 echo funcgraph-cpu 2>&1 > /sys/kernel/debug/tracing/trace_options
620 cat /sys/kernel/debug/tracing/set_ftrace_pid
624 echo 7379 > /sys/kernel/debug/tracing/set_ftrace_pid
625 cat /sys/kernel/debug/tracing/trace 2>&1 > ftrace.log.15November2018
639 export JAVA_HOME=/media/Ubuntu2/jdk1.8.0_171/
640 export PATH=/usr/bin:$PATH
671 /media/Ubuntu2/spark-2.3.1-bin-hadoop2.7/bin/spark-submit Spark_USBWWANLogMapReduceParser.py 2>&1 > testlogs/Spark_USBWWANLogMapReduceParser.FTraceAndUSBMon.log.15November2018
4. FTrace traces for specific userspace threads/processes are enabled by previous example commandlines and available through

/sys/kernel/debug/tracing/trace (circular buffer). Function graph traces show kernel function invocations as call graph edges (denoted by fn2 <- fn1)
5. Spark_USBWWANLogMapReduceParser.py has been changed to invoke log analyzer for USBmon and FTrace logs for
patterns Bi(BULK IN) and usb from USBmon and FTrace logs respectively:
- usbmon.15November2018.mon
- ftrace.ping.log.15November2018 (ftraces for ping of an IP address)
6. Logs for Spark Analyzer have been committed to Spark_USBWWANLogMapReduceParser.FTraceAndUSBMon.log.15November2018 which analyze the USBmon logs and WLAN traffic for IP address ping.

---

770. (THEORY and FEATURE) Program Analysis and Software Analytics - USBmd FTrace Kernel Function CallGraph Generation for Analysis - 22 November 2018 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

---

1.New bash shell script usb_md_ftrace.sh has been committed to repository which writes out an ftrace.log
file containing kernel function call graph sequences for an executable code. It is invoked as:

```
        $usb_md_ftrace.sh <executable-to-trace>
```

usb_md_ftrace.sh summarizes previously mentioned ftrace options enabling commands into single file with an
option for commandline argument of an executable to trace.
2.usb_wwan_modified/python-src/Spark_USBWWANLogMapReduceParser.py has been changed to include a new function
ftrace_callgraph_dot() which parses an ftrace log generated by usb_md_ftrace.sh for command:

```
        $usb_md_ftrace.sh traceroute <ip-address>
```

3.ftrace_callgraph_dot() parses each line of ftrace.log and adds them as edges in a NetworkX Directed Graph. DOT
file for this call graph is written to Spark_USBWWANLogMapReduceParser.ftrace_callgraph.dot
4.As a novelty, PageRank and Degree Centrality measures of the call graph NetworkX DiGraph are printed which show the prominently active regions of the kernel for traceroute . PageRank/Degree Centrality of kernel function callgraph is quite useful by treating every function caller as a voter to function callee. Theoretically, this centrality in kernel throws light on suspicious, malevolent invocations particularly involving memory and locking. In this traceroute ftrace example, lock and kmalloc functions have high centrality, and USB URB related functions are way down the ranking. More the ranking, deeper the function is in callstack trace in kernel.
5. Lot of functions have ISRA optimization of GCC. ISRA is known to cause signed int bugs (0 was erroneously promoted to 1 in loops) and ISRA has been disabled in ARM kernel: https://patchwork.kernel.org/patch/7113091/ by -fno-ipa-sra GCC flag. This kind of instability could be the reason for 32-bit VIRGO heisenbugs in string functions in older kernels.
6.Previous FTrace kernel call graph analysis is not only limited to USBmd WLAN analytics but can be applied to any executable requiring kernel profiling. Usual profilers measure time spent in the function whereas this graph theoretic analysis is superior and finds kernel bottlenecks and malicious patterns by analyzing call graphs within kernel.
7. Malicious code (e.g virus, worms, root-kits, bots, keystroke loggers) are usually associated with high cpu and memory footprint causing abnormal traffic. Analyzing infected kernel callgraph patterns might help in identifying the root cause.
8. FTrace kernel function call graph complements already implemented Program Analyzers: SATURN CFG driver in VIRGO kernels (accessible only in kernelspace) and Valgrind/KCachegrind/Callgrind userspace call graph analyzer in AsFer. By this kernel activity is partially visible and can be analyzed graph

theoretically from userspace.
9. Outbreak of epidemics have been analyzed as Game Theoretic problem (https://blogs.cornell.edu/info2040/2016/09/16/game-theory-in-the-context-of-epidemics/) - on how people behave in epidemics and their conclusion - "faster information limits disease spread". Cybercrimes are epidemics counterpart in cloud of computers only difference being damage inflicted on intellectual property than humans and adversaries are hackers/malicious code in place of viri. This makes Cybercrimes a multi-player adversarial game involving Hackers/Malicious code Versus Aggrieved. Translating the previous conclusion to cybercrimes: Faster information about malicious code limits the damage.

```
/
*********************************************************************************
*******
#-------------------------------------------------------------------------------
------------------------
#NEURONRAIN VIRGO - Cloud, Machine Learning and Queue augmented Linux Kernel
Fork-off
#This program is free software: you can redistribute it and/or modify
#it under the terms of the GNU General Public License as published by
#the Free Software Foundation, either version 3 of the License, or
#(at your option) any later version.
#This program is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#GNU General Public License for more details.
#You should have received a copy of the GNU General Public License
#along with this program.  If not, see <http://www.gnu.org/licenses/>.
#-------------------------------------------------------------------------------
------------------------
#Copyleft (Copyright+):
#Srinivasan Kannan (alias) Ka.Shrinivaasan (alias) Shrinivas Kannan
#Ph: 9791499106, 9003082186
#Krishna iResearch Open Source Products Profiles:
#http://sourceforge.net/users/ka_shrinivaasan,
#https://github.com/shrinivaasanka,
#https://www.openhub.net/accounts/ka_shrinivaasan
#Personal website(research): https://sites.google.com/site/kuja27/
#emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
#kashrinivaasan@live.com
#-------------------------------------------------------------------------------
------------------------
*********************************************************************************
*********/
```

VIRGO is an operating system kernel forked off from Linux kernel mainline to add cloud functionalities (system calls, modules etc.,) within kernel itself with machine learning, analytics, debugging, queueing support in the deepest layer of OSI stack i.e AsFer, USBmd, KingCobra together with VIRGO constitute the previous functionalities. Presently there seems to be no cloud implementation with fine-grained cloud primitives (system calls, modules etc.,) included in kernel itself though there are coarse grained clustering and SunRPC implementations available. VIRGO complements other Clustering and application layer cloud OSes like cloudstack, openstack etc., in these aspects - CloudStack and OpenStack can be deployed on a VIRGO Linux Kernel Cloud - OpenStack nova compute, neutron network, cinder/swift storage subsystems can be augmented to have additional drivers that invoke lowlevel VIRGO syscall and kernel module primitives (assuming there are no coincidental replications of functionalities) thereby acting as a foundation to application layer cloud.

Remote Device Invocation , which is an old terminlogy for Internet-Of-Things has already been experimented in SunRPC and KOrbit CORBA-in-linux-kernel kernel modules in old linux kernels (http://www.csn.ul.ie/~mark/fyp/fypfinal.html -

with Solaris MC and example Remote Device Client-Server Module implementation). VIRGO Linux with the larger encompassing NeuronRain suite is an effort to provide a unified end-to-end application-to-kernel machine-learning propelled cloud and internet-of-things framework.

Memory pooling:

Memory pooling is proposed to be implemented by a new virgo_malloc() system call that transparently allocates a block of virtual memory from memory pooled from virtual memory scattered across individual machines part of the cloud.

CPU pooling or cloud ability in a system call:

Clone() system call is linux specific and internally it invokes sys_clone(). All fork(),vfork() and clone() system calls internally invoke do_fork(). A new system call virgo_clone() is proposed to create a thread transparently on any of the available machines on the cloud.This creates a thread on a free or least-loaded machine on the cloud and returns the results.

virgo_clone() is a wrapper over clone() that looks up a map of machines-to-loadfactor and get the host with least load and invokes clone() on a function on that gets executed on the host. Usual cloud implementations provide userspace API that have something similar to this - call(function,host). Loadfactor can be calculated through any of the prominent loadbalancing algorithm. Any example userspace code that uses clone() can be replaced with virgo_clone() and all such threads will be running in a cloud transparently.Presently Native POSIX threads library(NPTL) and older LinuxThreads thread libraries internally use clone().

Kernel has support for kernel space sockets with kernel_accept(), kernel_bind(), kernel_connect(), kernel_sendmsg() and kernel_recvmsg() that can be used inside a kernel module. Virgo driver implements virgo_clone() system call that does a kernel_connect() to a remote kernel socket already __sock_create()-d, kernel_bind()-ed and kernel_accept()-ed and does kernel_sendmsg() of the function details and kernel_recvmsg() after function has been executed by clone() in remote machine. After kernel_accept() receives a connection it reads the function and parameter details. Using these kthread_create() is executed in the remote machine and results are written back to the originating machine. This is somewhat similar to SunRPC but adapted and made lightweight to suit virgo_clone() implementation without any external data representation.

Experimental Prototype
---
virgo_clone() system call and a kernel module virgocloudexec which implements Sun RPC interface have been implemented.

VIRGO - loadbalancer to get the host:ip of the least loaded node
---
Loadbalancer option 1 - Centralized loadbalancer registry that tracks load:

Virgo_clone() system call needs to lookup a registry or map of host-to-load and get the least loaded host:ip from it. This requires a load monitoring code to run periodically and update the map. If this registry is located on a single machine then simultaneous virgo_clone() calls from many machines on the cloud could choke the registry. Due to this, loadbalancer registry needs to run on a high-end machine. Alternatively,each machine can have its own view of the load and multiple copies of load-to-host registries can be stored in individual machines. Synchronization of the copies becomes a separate task in itself(Cache coherency). Either way gives a tradeoff between accuracy, latency and efficiency.

Many application level userspace load monitoring tools are available but as virgo_clone() is in kernel space, it needs to be investigated if kernel-to-kernel loadmonitoring can be done without userspace data transport.Most Cloud

API explicitly invoke a function on a host. If this functionality is needed, virgo_clone() needs to take host:ip address as extra argument,but it reduces transparent execution.

(Design notes for LB option 1 handwritten by myself are at :http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/MiscellaneousOpenSourceDesignAndAcademicResearchNotes.pdf)

Loadbalancer option 2 - Linux Psuedorandom number generator based load balancer(experimental) instead of centralized registry that tracks load:

Each virgo_clone() client has a PRG which is queried (/dev/random or /dev/urandom) to get the id of the host to send the next virgo_clone() function to be executed
Expected number of requests per node is derived as:

expected number of requests per node = summation(each_value_for_the_random_variable_for_number_of_requests * probability_for_each_value) where random variable ranges from 1 to k where N is number of processors and k is the number of requests to be distributed on N nodes

=expected number of requests per node = (math.pow(N, k+2) - k*math.pow(N,2) + k*math.pow(N,1) - 1) / (math.pow(N, k+3) - 2*math.pow(N,k+2) + math.pow(N,k+1))

This loadbalancer is dependent on efficacy of the PRG and since each request is uniformly, identically, independently distributed use of PRG
would distribute requests evenly. This obviates the need for loadtracking and coherency of the load-to-host table.

(Design notes for LB option 2 handwritten by myself at :http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/MiscellaneousOpenSourceDesignAndAcademicResearchNotes.pdf)

(python script in virgo-python-src/)

****
Implemented VIRGO Linux components (as on 7 March 2016)
****
1. cpupooling virtualization - VIRGO_clone() system call and VIRGO cpupooling driver by which a remote procedure can be invoked in kernelspace.(port: 10000)
2. memorypooling virtualization - VIRGO_malloc(), VIRGO_get(), VIRGO_set(), VIRGO_free() system calls and VIRGO memorypooling driver by which kernel memory can be allocated in remote node, written to, read and freed - A kernelspace memcache-ing.(port: 30000)
3. filesystem virtualization - VIRGO_open(), VIRGO_read(), VIRGO_write(), VIRGO_close() system calls and VIRGO cloud filesystem driver by which file IO in remote node can be done in kernelspace.(port: 50000)
4. config - VIRGO config driver for configuration symbols export.
5. queueing - VIRGO Queuing driver kernel service for queuing incoming requests, handle them with workqueue and invoke KingCobra service routines in kernelspace. (port: 60000)
6. cloudsync - kernel module for synchronization primitives (Bakery algorithm etc.,) with exported symbols that can be used in other VIRGO cloud modules for critical section lock() and unlock()
7. utils - utility driver that exports miscellaneous kernel functions that can be used across VIRGO Linux kernel
8. EventNet - eventnet kernel driver to vfs_read()/vfs_write() text files for EventNet vertex and edge messages (port: 20000)

9. Kernel_Analytics - kernel module that reads machine-learnt config key-value pairs set in /etc/virgo_kernel_analytics.conf. Any machine learning software can be used to get the key-value pairs for the config. This merges three facets - Machine Learning, Cloud Modules in VIRGO Linux-KingCobra-USBmd , Mainline Linux Kernel
10. Testcases and kern.log testlogs for the above
11. SATURN program analysis wrapper driver.

Thus VIRGO Linux at present implements a minimum cloud OS (with cloud-wide cpu, memory and file system management) over Linux and potentially fills in a gap to integrate both software and hardware into cloud with machine learning and analytics abilities that is absent in application layer cloud implementations. Thus VIRGO cloud is an IoT operating system kernel too that enables any hardware to be remote controlled. Data analytics using AsFer can be done by just invoking requisite code from a kernelspace driver above and creating an updated driver binary (or) by kernel_analytics module which reads the userland machine-learnt config.

****
VIRGO ToDo and NiceToHave Features (list is quite dynamic and might be rewritten depending on feasibility - longterm with no deadline)
****
(FEATURE - DONE-minimum separate config file support in client and kernel service)1. More Sophisticated VIRGO config file and read_virgo_config() has to be invoked on syscall clients virgo_clone and virgo_malloc also. Earlier config was being read by kernel module only which would work only on a single machine. A separate config module kernel service has been added for future use while exporting kernel-wide configuration related symbols. VIRGO config files have been split into /etc/virgo_client.conf and /etc/virgo_cloud.conf to delink the cloud client and kernel service config parameters reading and to do away with oft occurring symbol lookup errors and multiple definition errors for num_cloud_nodes and node_ip_addrs_in_cloud - these errors are frequent in 3.15.5 kernel than 3.7.8 kernel. Each VIRGO module and system call now reads the config file independent of others - there is a read_virgo_config_<module>_<client_or_service>() function variant for each driver and system call. Though at present smacks of a replicated code, in future the config reads for each component (system call or module) might vary significantly depending on necessities. New kernel module config has been added in drivers/virgo. This is for future prospective use as a config export driver that can be looked up by any other VIRGO module for config parameters. include/linux/virgo_config.h has the declarations for all the config variables declared within each of the VIRGO kernel modules. Config variables in each driver and system call have been named with prefix and suffix to differentiate the module and/or system call it serves. In geographically distributed cloud virgo_client.conf has to be in client nodes and virgo_cloud.conf has to be in cloud nodes. For VIRGO Queue - KingCobra REQUEST-REPLY peer-to-peer messaging system same node can have virgo_client.conf and virgo_cloud.conf. Above segregation largely simplifies the build process as each module and system call is independently built without need for a symbol to be exported from other module by pre-loading it.(- from commit comments done few months ago)

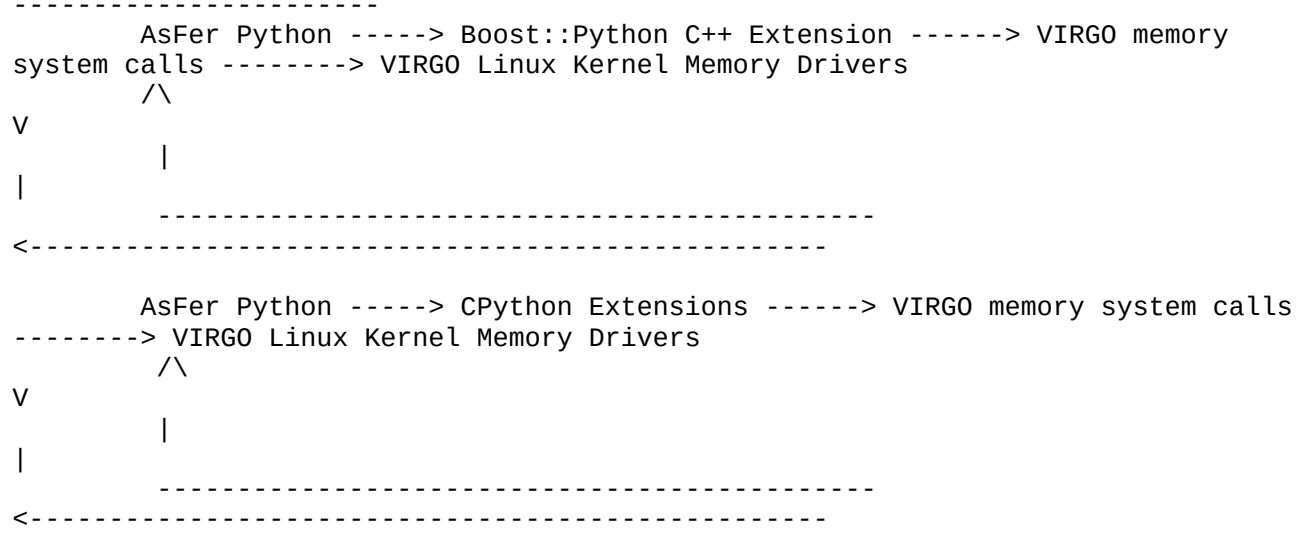
(FEATURE - Special case implementation DONE) 2. Object Marshalling and Unmarshalling (Serialization) Features - Feature 4 is a marshalling feature too as Python world PyObjects are serialized into VIRGO linux kernel and unmarshalled back bottom-up with CPython and Boost::Python C++ invocations - CPython and Boost internally take care of serialization.

(FEATURE - DONE) 3. Virgo_malloc(), virgo_set(), virgo_get() and virgo_free() syscalls that virtualize the physical memory across all cloud nodes into a single logical memory behemoth (NUMA visavis UMA). (There are random crashes in copy_to_user and copy_from_user in syscall path for VIRGO memory pooling

commands that were investigated but turned out to be mystery). These crashes have either been resolved or occur less in 3.15.5 and 4.1.5 kernels.
Initial Design Handwritten notes committed at: http://sourceforge.net/p/virgo-linux/code-0/210/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_initial_design_notes.pdf

(FEATURE - DONE) 4. Integrated testing of AsFer-VIRGO Linux Kernel request roundtrip - invocation of VIRGO linux kernel system calls from AsFer Python via C++ or C extensions - Commits for this have been done on 29 January 2016. This unifies userlevel applications and kernelspace modules so that AsFer Python makes VIRGO linux kernel an extension. Following is schematic diagram and More details in commit notes below.

4.1 Schematic Diagram:



(FEATURE - DONE)5. Multithreading of VIRGO cloudexec kernel module (if not already done by kernel module subsystem internally)

(FEATURE - DONE) 6. Sophisticated queuing and persistence of CPU and Memory pooling requests in Kernel Side (by possibly improving already existing kernel workqueues). Either open source implementations like ZeroMQ/ActiveMQ can be used or Queuing implementation has to be written from scratch or both. ActiveMQ supports REST APIs and is JMS implementation. This feature has been marked completed because recently NeuronRain AsFer backend has been updated to support KingCobra REQUEST_REPLY.queue as a datasource for Streaming Abstract Generator. By enabling use_as_kingcobra_service=1 in cpupooling and memorypooling VIRGO drivers, any incoming CPU and Memory related request can be routed to KingCobra by linux workqueue in VIRGO queue and disk persisted (/var/log/REQUEST_REPLY.queue) by KingCobra servicerequest recipient. Also Kafka Publisher/Subscriber have been implemented in NeuronRain AsFer which invoke Streaming Abstract Generator with KingCobra REQUEST_REPLY.queue as datasource to publish persisted already received CPU and Memory requests to Kafka Message Queue. Thus queuing and persistence for VIRGO CPU and Memory is in place. ZeroMQ does not have persistence and is used for NeuronRain client side Router-Dealer concurrent request servicing pattern.

(FEATURE - DONE-Minimum Functionality) 7. Integration of Asfer(AstroInfer) algorithm codes into VIRGO which would add machine learning capabilities into VIRGO - That is, VIRGO cloud subsystem which is part of a linux kernel installation "learns" and "adapts" to the processes that are executed on VIRGO. This catapults the power of the Kernel and Operating System into an artificially (rather approximately naturally) intelligent computing platform (a software "brain"). For example VIRGO can "learn" about "execution times" of processes and suitably act for future processes. PAC Learning of functions could be theoretical basis for this. Initial commits for Kernel Analytics Module which reads the /etc/virgo_kernel_analytics.conf config have been done. This config

file virgo_kernel_analytics.conf having csv(s) of key-value pairs of analytics variables is set by AsFer or any other Machine Learning code. With this VIRGO Linux Kernel is endowed with abilities to dynamically evolve than being just a platform for user code. Implications are huge - for example, a config variable "MaxNetworkBandwidth=255" set by the ML miner in userspace based on a Perceptron or Logistic Regression executed on network logs can be read by a kernel module that limits the network traffic to 255Mbps. Thus kernel is no longer a static predictable blob behemoth. With this, VIRGO is an Internet-of-Things kernel that does analytics and based on analytics variable values integrated hardware can be controlled across the cloud through remote kernel module function invocation. This facility has been made dynamic with Boost::Python C++ and CPython extensions that permit flow of objects from machine learnt AsFer kernel analytics variables to VIRGO Linux Kernel memory drivers via VIRGO system calls directly and back - Commits on 29 January 2016 - this should obviate re-reading /etc/virgo_kernel_analytics.conf and is an exemplary implementation which unifies C++/Python into C/Kernel.

Example scenario 1 without implementation:

- Philips Hue IoT mobile app controlled bulb - http://www2.meethue.com/en-xx/
- kernel_analytics module learns key-value pairs from the AsFer code and exports it VIRGO kernel wide
- A driver function with in bulb embedded device driver can be invoked through VIRGO cpupooling (invoked from remote virgo_clone() system_call)
based on if-else clause of the kernel_analytics variable i.e remote_client invokes virgo_clone() with function argument "lights on" which is routed to another cloud node. The recipient cloud node "learns" from AsFer kernel_analytics that Voltage is low or Battery is low from logs and decides to switch in high beam or low beam.

Example scenario 2 without implementation:

- A swivel security camera driver is remotely invoked via virgo_clone() in the VIRGO cloud.
- The camera driver uses a machine learnt variable exported by kernel_analytics-and-AsFer to pan the camera by how much degrees.

Example scenario 3 without implementation - probably one of the best applications of NeuronRain IoT OS:

- Automatic Driverless Automobiles - a VIRGO driver for a vehicle which learns kernel analytics variables (driving directions) set by kernel_analytics driver and AsFer Machine Learning. A naive algorithm for Driverless Car (with some added modifications over A-Star and Motion planning algorithms):
	- AsFer analytics receives obstacle distance data 360+360 degrees (vertical and horizontal) around the vehicle (e.g ultrasound sensors) which is updated in a Spark DataFrame table with high frequency (100 times per second).
	- VIRGO Linux kernel on vehicle has two special drivers for Gear-Clutch-Break-Accelerator-Fuel(GCBAF) and Steering listening on some ports.
	- AsFer analytics with high frequency computes threshold variables for applying break, clutch, gear, velocity, direction, fuel changes which are written to kernel_analytics.conf realtime based on distance data from Spark table.
	- These analytics variables are continuously read by GCBAF and Steering drivers which autopilot the vehicle.
	- Above applies to Fly-by-wire aeronautics too with appropriate changes in analytics variables computed.
	- The crucial parameter is the response time in variable computation and table updates which requires a huge computing power unless the vehicle is hooked onto a Spark cloud in motion by wireless which process the table and compute analytic variables.

References for Machine Learning + Linux Kernel

7.1 KernTune - http://repository.uwc.ac.za/xmlui/bitstream/handle/10566/53/Yi_KernTune(2007).pdf?sequence=3
7.2 Self-learning, Predictive Systems - https://icri-ci.technion.ac.il/projects/past-projects/machine-learning-for-architecture-self-learning-predictive-computer-systems/
7.3 Linux Process Scheduling and Machine Learning - http://www.cs.ucr.edu/~kishore/papers/tencon.pdf
7.4 Network Latency and Machine Learning - https://users.soe.ucsc.edu/~slukin/rtt_paper.pdf
7.5 Machine Learning based Meta-Scheduler for Multicore processors - https://books.google.co.in/books?id=1GWcHmCrl0QC&pg=PA528&lpg=PA528&dq=linux+kernel+machine+learning&source=bl&ots=zfJsq_uu5q&sig=mMIUZ-oyJIwZXtYj4HntrQE8NSk&hl=en&sa=X&ved=0CCAQ6AEwATgKahUKEwjs9sqF9vPIAhVBFZQKHbNtA6A

8. A Symmetric Multi Processing subsystem Scheduler that virtualizes all nodes in cloud (probably this would involve improving the loadbalancer into a scheduler with priority queues)

(FEATURE - ONGOING) 9. Virgo is an effort to virtualize the cloud as a single machine - Here cloud is not limited to servers and desktops but also mobile devices that run linux variants like Android, and other Mobile OSes. In the longterm, Virgo may have to be ported or optimized for handheld devices. Boost::Python AsFer-VIRGO system call invocations implemented in NeuronRain is framework for implementing python applications interfacing with kernel. If deployed on Mobile processors (e.g by overlaying Android Kernel with VIRGO layer) there are IDEs like QPython to develop python apps for Android.

(FEATURE - DONE) 10. Memory Pooling Subsystem Driver - Virgo_malloc(), Virgo_set(), Virgo_get() and Virgo_free() system calls and their Kernel Module Implementations. In addition to syscall path, telnet or userspace socket client interface is also provided for both VIRGO CPU pooling(virgo_clone()) and VIRGO Memory Pooling Drivers.

(FEATURE - DONE) 11. Virgo Cloud File System with virgo_cloud_open(), virgo_cloud_read() , virgo_cloud_write() and virgo_cloud_close() commands invoked through telnet path has been implemented that transcends disk storage in all nodes in the cloud. It is also fanciful feature addition that would make VIRGO a complete all-pervading cloud platform. The remote telnet clients send the file path and the buf to be read or data to be written. The Virgo File System kernel driver service creates a unique Virgo File Descriptor for each struct file* opened by filp_open() and is returned to client. Earlier design option to use a hashmap (linux/hashmap.h) looked less attractive as file desciptor is an obvious unique description for open file and also map becomes unscalable. The kernel upcall path has been implemented (paramIsExecutable=0) and may not be necessary in most cases and all above cloudfs commands work in kernelspace using VFS calls.

(FEATURE - DONE) 12. VIRGO Cloud File System commands through syscall paths - virgo_open(),virgo_close(),virgo_read() and virgo_write(). All the syscalls have been implemented with testcases and more bugs fixed. After fullbuild and testing, virgo_open() and virgo_read() work and copy_to_user() is working.

(FEATURE - DONE) 13. VIRGO memory pooling feature is also a distributed key-value store similar to other prominent key-store software like BigTable implementations, Dynamo, memory caching tools etc., but with a difference that VIRGO mempool is implemented as part of Linux Kernel itself thus circumventing userspace latencies. Due to Kernel space VIRGO mempool has an added power to

store and retrieve key-value pair in hardware devices directly which otherwise is difficult in userspace implementations.

14. VIRGO memory pooling can be improved with disk persistence for in-memory key-value store using virgo_malloc(),virgo_set(),virgo_get() and virgo_free() calls. Probably this might be just a set of invocations of read and write ops in disk driver or using sysfs. Probably this could be redundant as the VIRGO filesystem primitives have been implemented that write to a remote host's filesystem in kernelspace.

15. (FEATURE-DONE) Socket Debugging, Program Analysis and Verification features for user code that can find bugs statically. Socket skbuff debug utility and SATURN Program Analysis Software has been integrated into NEURONRAIN VIRGO Linux Kernel.

16(FEATURE - DONE-Minimum Functionality). Operating System Logfile analysis using Machine Learning code in AstroInfer for finding patterns of processes execution and learn rules from the log. Kernel_Analytics VIRGO module reads /etc/virgo_kernel_analytics.conf config key-value pairs which are set by AsFer or other Machine Learning Software. At present an Apache Spark usecase that mines Uncomplicated Fire Wall logs in kern.log for most prominent source IP has been implemented in AsFer codebase : http://sourceforge.net/p/asfer/code/704/tree/python-src/SparkKernelLogMapReduceParser.py . This is set as a key-value config in /etc/virgo_kernel_analytics.conf read and exported by kernel_analytics module.

17. Implementations of prototypical Software Transactional Memory and LockFree Datastructures for VIRGO memory pooling.

18. Scalability features for Multicore machines - references: (http://halobates.de/lk09-scalability.pdf, http://pdos.csail.mit.edu/papers/linux:osdi10.pdf)

19. Read-Copy-Update algorithm implementation for VIRGO memory pooling that supports multiple simultaneous versions of memory for readers - widely used in redesigned Linux Kernel.

20. (FEATURE - SATURN integration - minimum functionality DONE) Program Comprehension features as an add-on described in : https://sites.google.com/site/kuja27/PhDThesisProposal.pdf?attredirects=0. SATURN program analysis has been integrated into VIRGO linux with a stub driver.

21. (FEATURE - DONE) Bakery Algorithm implementation - cloudsync kernel module

22. (FEATURE - ONGOING) Implementation of Distributed Systems primitives for VIRGO cloud viz., Logical Clocks, Termination Detection, Snapshots, Cache Coherency subsystem etc.,(as part of cloudsync driver module). Already a simple timestamp generation feature has been implemented for KingCobra requests with <ipaddress>:<localmachinetimestamp> format

23. (FEATURE - minimum functionality DONE) Enhancements to kmem if it makes sense, because it is better to rely on virgo_malloc() for per machine memory management and wrap it around with a cloudwide VIRGO Unique ID based address lookup implementation of which is already in place.
Kernel Malloc syscall kmalloc() internally works as follows:
- kmem_cache_t object has pointers to 3 lists
- These 3 lists are full objects SLAB list, partial objects SLAB list and free objects SLAB list - all are lists of objects of same size
and cache_cache is the global list of all caches created thus far.
- Any kmalloc() allocation searches partial objects SLAB list and allocates a memory block with kmem_cache_alloc() from the first SLAB available - returned to caller.
- Any kfree() returns an object to a free SLAB list
- Full SLABs are removed from partial SLAB list and appended to full SLAB

list
      - SLABs are virtual memory pages created with kmem_cache_create
      - Each SLAB in SLABs list has blocks of similar sized objects (e.g. multiples of two). Closest matching block is returned and fragmentation is minimized (incidentally this is the knapsack and packing optimization LP problem and thus NP-complete).

KERNELSPACE:
VIRGO address translation table already implements a tree registry of vtables each of capacity 3000 that keep track of kmalloc() allocations across all cloud nodes. Implementation of SLAB allocator for kmalloc() creates a kmem_cache(s) of similar sized objects and kmem_cache_alloc() allocates from these caches. kmalloc() already does lot of per-machine optimizations. VIRGO vtable registry tree maintained in VIRGO memory syscall end combined with per-machine kmalloc() cache_cache already look sufficient. Instrumenting kmem_cache_create() with #ifdef SLAB_CLOUD_MALLOC flags to do RPC looks superfluous. Hence marking this action item as done. Any further optimization can be done on top of existing VIRGO address translation table struct - e.g bookkeeping flags, function pointers etc.,.
USERSPACE: sbrk() and brk() are no longer used internally in malloc() library routines. Instead mmap() has replaced it (http://web.eecs.utk.edu/courses/spring2012/cs360/360/notes/Malloc1/lecture.html, http://web.eecs.utk.edu/courses/spring2012/cs360/360/notes/Malloc1/diff.html).

24.(FEATURE - ONGOING) Cleanup the code and remove unnecessary comments.

25.(FEATURE - DONE) Documentation - This design document is also a documentation for commit notes and other build and debugging technical details. Doxygen html cross-reference documentation for AsFer, USBmd, VIRGO, KingCobra and Acadpdrafts has been created along with summed-up design document and committed to GitHub Repository at https://github.com/shrinivaasanka/Krishna_iResearch_DoxygenDocs

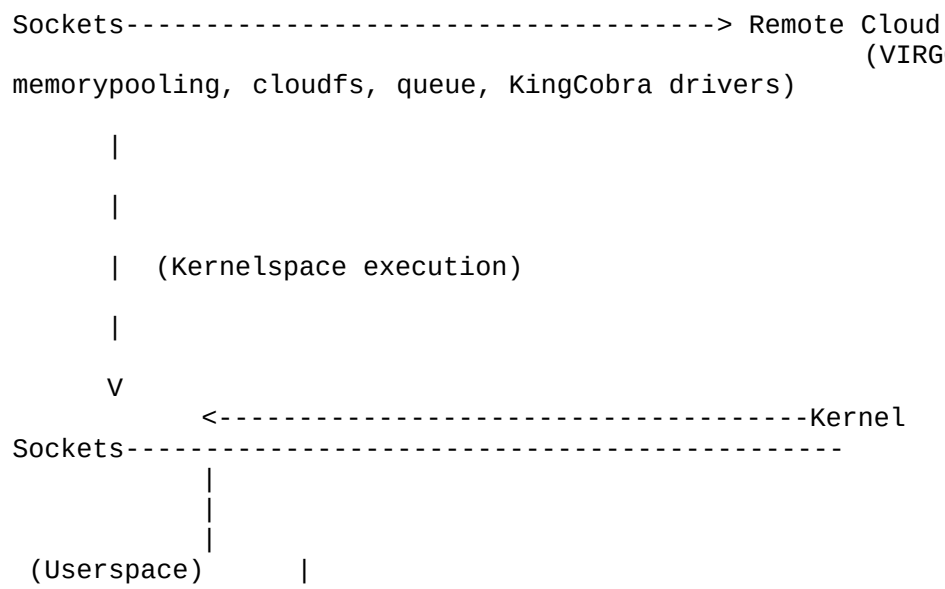
26. (FEATURE - DONE) Telnet path to virgo_cloud_malloc,virgo_cloud_set and virgo_cloud_get has been tested and working. This is similar to memcached but stores key-value in kernelspace (and hence has the ability to write to and retrieve from any device driver memory viz., cards, handheld devices).An optional todo is to write a script or userspace socket client that connects to VIRGO mempool driver for these commands.

27. Augment the Linux kernel workqueue implementation (http://lxr.free-electrons.com/source/kernel/workqueue.c) with disk persistence if feasible and doesn't break other subsystems - this might require additional persistence flags in work_struct and additional #ifdefs in most of the queue functions that write and read from the disk. Related to item 6 above.

28.(FEATURE - DONE) VIRGO queue driver with native userspace queue and kernel workqueue-handler framework that is optionally used for KingCobra and is invoked through VIRGO cpupooling and memorypooling drivers. (Schematic in http://sourceforge.net/p/kcobra/code-svn/HEAD/tree/KingCobraDesignNotes.txt and http://sourceforge.net/p/acadpdrafts/code/ci/master/tree/Krishna_iResearch_opensourceproducts_archdiagram.pdf)

29.(FEATURE - DONE) KERNELSPACE EXECUTION ACROSS CLOUD NODES which geographically distribute userspace and kernelspace execution creating a logical abstraction for a cloudwide virtualized kernel:

      Remote Cloud Node Client
      (cpupooling, eventnet, memorypooling, cloudfs, queueing - telnet and syscalls clients)
            |
            |
 (Userspace) |
            |-----------------------------------Kernel



30. (FEATURE - DONE) VIRGO platform as on 5 May 2014 implements a minimum set of features and kernelsocket commands required for a cloud OS kernel: CPU virtualization(virgo_clone), Memory virtualization(virgo_malloc,virgo_get,virgo_set,virgo_free) and a distributed cloud file system(virgo_open,virgo_close,virgo_read,virgo_write) on the cloud nodes and thus gives a logical view of one unified, distributed linux kernel across all cloud nodes that splits userspace and kernelspace execution across cloud as above.

31. (FEATURE - DONE) VIRGO Queue standalone kernel service has been implemented in addition to paths in schematics above. VIRGO Queue listens on hardcoded port 60000 and enqueues the incoming requests to VIRGO queue which is serviced by KingCobra:

VIRGO Queue client(e.g telnet) ------> VIRGO Queue kernel service ---> Linux Workqueue handler ------> KingCobra

32. (FEATURE - DONE) EventNet kernel module service:
VIRGO eventnet client (telnet) -------> VIRGO EventNet kernel service -----> EventNet graph text files

33. (FEATURE - DONE) Related to point 22 - Reuse EventNet cloudwide logical time infinite graph in AsFer in place of Logical clocks. At present the eventnet driver listens on port 20000 and writes the edges and vertices files in kernel using vfs_read()/vfs_write(). These text files can then be read by the AsFer code to generate DOT files and visualize the graph with graphviz.

34. (FEATURE - OPTIONAL) The kernel modules services listening on ports could return a JSON response when connected instead of plaintext, conforming to REST protocol. Additional options for protocol buffers which are becoming a standard data interchange format.

35. (FEATURE-Minimum Functionality DONE) Pointer Swizzling and Unswizzling of VIRGO addressspace pointers to/from VIRGO Unique ID (VUID). Presently VIRGO memory system calls implement a basic minimal pointer address translation to unique kmem location identifier.

****
CODE COMMIT RELATED NOTES
****

VIRGO code commits as on 16/05/2013
---
1. VIRGO cloudexec driver with a listener kernel thread service has been implemented and it listens on port 10000 on system startup through /etc/modules load-on-bootup facility

2. VIRGO cloudexec virgo_clone() system call has been implemented that would kernel_connect() to the VIRGO cloudexec service listening at port 10000

3. VIRGO cloudexec driver has been split into virgo.h (VIRGO typedefs), virgocloudexecsvc.h(VIRGO cloudexec service that is invoked by module_init() of VIRGO cloudexec driver) and virgo_cloudexec.c (with module ops definitions)

4. VIRGO does not implement SUN RPC interface anymore and now has its own virgo ops.

5. Lot of Kbuild related commits with commented lines for future use have been done viz., to integrate VIRGO to Kbuild, KBUILD_EXTRA_SYMBOLS for cross-module symbol reference.

VIRGO code commits as on 20/05/2013
---
1. test_virgo_clone.c testcase for sys_virgo_clone() system call works and connections are established to VIRGO cloudexec kernel module.

2. Makefile for test_virgo_clone.c and updated buildscript.sh for headers_install for custom-built linux.

VIRGO code commits as on 6/6/2013
---
1. Message header related bug fixes

VIRGO code commits as on 25/6/2013

1.telnet to kernel service was tested and found working
2.GFP_KERNEL changed to GFP_ATOMIC in VIRGO cloudexec kernel service

VIRGO code commits as on 1/7/2013
---
1. Instead of printing iovec, printing buffer correctly prints the messages
2. wake_up_process() added and function received from virgo_clone() syscall is executed with kernel_thread and results returned to virgo_clone() syscall client.

commit as on 03/07/2013

PRG loadbalancer preliminary code implemented. More work to be done

commit as on 10/07/2013

Tested PRG loadbalancer read config code through telnet and virgo_clone. VFS code to read from virgo_cloud.conf commented for testing

commits as on 12/07/2013

PRG loadbalancer prototype has been completed and tested with test_virgo_clone and telnet and symbol export errors and PRG errors have been fixed

commits as on 16/07/2013
---

read_virgo_config() and read_virgo_clone_config()(replica of
read_virgo_config()) have been implemented and tested to read the
virgo_cloud.conf config parameters(at present the virgo_cloud.conf has comma
separated list of ip addresses. Port is hardcoded to 10000 for uniformity across
all nodes). Thus minimal cloud functionality with config file support is in
place. Todo things include function pointer lookup in kernel service, more
parameters to cloud config file if needed, individual configs for virgo_clone()
and virgo kernel service, kernel-to-userspace upcall and execution instead of
kernel space, performance tuning etc.,

commits as on 17/07/2013
----
moved read_virgo_config() to VIRGOcloudexec's module_init so that config is read
at boot time and exported symbols are set beforehand.
Also commented read_virgo_clone_config() as it is redundant

commits as on 23/07/2013

Lack of reflection kind of facilities requires map of function_names to
pointers_to_functions to be executed
on cloud has to be lookedup in the map to get pointer to function. This map is
not scalable if number of functions are
in millions and size of the map increases linearly. Also having it in memory is
both CPU and memory intensive.
Moreover this map has to be synchronized in all nodes for coherency and
consistency which is another intensive task.
Thus name to pointer function table is at present not implemented. Suitable way
to call a function by name of the function
is yet to be found out and references in this topic are scarce.

If parameterIsExecutable is set to 1 the data received from virgo_clone() is not
a function but name of executable
This executable is then run on usermode using call_usermodehelper() which
internally takes care of queueing the workstruct
and executes the binary as child of keventd and reaps silently. Thus workqueue
component of kernel is indirectly made use of.
This is sometimes more flexible alternative that executes a binary itself on
cloud and
is preferable to clone()ing a function on cloud. Virgo_clone() syscall client or
telnet needs to send the message with name of binary.

If parameterIsExecutable is set to 0 then data received from virgo_clone() is
name of a function and is executed in else clause
using dlsym() lookup and pthread_create() in user space. This unifies both
call_usermodehelper() and creating a userspace thread
with a fixed binary which is same for any function. The dlsym lookup requires
mangled function names which need to be sent by
virgo_clone or telnet. This is far more efficient than a function pointer table.

call_usermodehelper() Kernel upcall to usermode to exec a fixed binary that
would inturn execute the cloneFunction in userspace
by spawning a pthread. cloneFunction is name of the function and not binary.
This clone function will be dlsym()ed
and a pthread will be created by the fixed binary. Name of the fixed binary is
hardcoded herein as
"virgo_kernelupcall_plugin". This fixed binary takes clone function as argument.
For testing libvirgo.so has been created from
virgo_cloud_test.c and separate build script to build the cloud function
binaries has been added.

 - Ka.Shrinivaasan (alias) Shrinivas Kannan (alias) Srinivasan Kannan
   (https://sites.google.com/site/kuja27)

commits as on 24/07/2013
----

test_virgo_clone unit test case updated with mangled function name to be sent to remote cloud node. Tested with test_virgo_clone
end-to-end and all features are working. But sometimes kernel_connect hangs randomly (this was observed only today and looks similar
to blocking vs non-blocking problem. Origin unknown).

- Ka.Shrinivaasan (alias) Shrinivas Kannan (alias) Srinivasan Kannan
  (https://sites.google.com/site/kuja27)

commits as on 29/07/2013

Added kernel mode execution in the clone_func and created a sample kernel_thread for a cloud function. Some File IO logging added to upcall
binaries and parameterIsExecutable has been moved to virgo.h

commits as on 30/07/2013

New usecase virgo_cloud_test_kernelspace.ko kernel module has been added. This exports a function virgo_cloud_test_kernelspace() and is
accessed by virgo_cloudexec kernel service to spawn a kernel thread that is executed in kernel addresspace. This Kernel mode execution
on cloud adds a unique ability to VIRGO cloud platform to seamlessly integrate hardware devices on to cloud and transparently send commands
to them from a remote cloud node through virgo_clone().

Thus above feature adds power to VIRGO cloud to make it act as a single "logical device driver" though devices are in geographically in a remote server.

commits as on 01/08/2013 and 02/08/2013

Added Bash shell commandline with -c option for call_usermodehelper upcall clauses to pass in remote virgo_clone command message as
arguments to it. Also tried output redirection but it works some times that too with a fatal kernel panic.

Ideal solutions are :
1. either to do a copy_from_user() for message buffer from user address space (or)
2. somehow rebuild the kernel with fd_install() pointing stdout to a VFS file* struct. In older kernels like 2.6.x, there is an fd_install code
with in kmod.c (___call_usermodehelper()) which has been redesigned in kernel 3.x versions and fd_install has been removed in kmod.c .
3. Create a Netlink socket listener in userspace and send message up from kernel Netlink socket.

All the above are quite intensive and time consuming to implement.Moreover doing FileIO in usermode helper is strongly discouraged in kernel docs

Since Objective of VIRGO is to virtualize the cloud as single execution "machine", doing an upcall (which would run with root abilities) is
redundant often and kernel mode execution is sufficient. Kernel mode execution with intermodule function invocation can literally take over
the entire board in remote machine (since it can access PCI bus, RAM and all other device cards)

As a longterm design goal, VIRGO can be implemented as a separate protocol itself and sk_buff packet payload from remote machine
can be parsed by kernel service and kernel_thread can be created for the message.

commits as on 05/08/2013:

Major commits done for kernel upcall usermode output logging with fd_install redirection to a VFS file. With this it has become easy for user space to communicate runtime data to Kernel space. Also a new strip_control_M() function has been added to strip \r\n or " ".

11 August 2013:

Open Source Design and Academic Research Notes uploaded to http://sourceforge.net/projects/acadpdrafts/files/MiscellaneousOpenSourceDesignAndAcademicResearchNotes_2013-08-11.pdf/download

commits as on 23 August 2013

New Multithreading Feature added for VIRGO Kernel Service - action item 5 in ToDo list above (virgo_cloudexec driver module). All dependent headers changed for kernel threadlocalizing global data.

commits as on 1 September 2013

GNU Copyright license and Product Owner Profile (for identity of license issuer) have been committed. Also Virgo Memory Pooling - virgo_malloc() related initial design notes (handwritten scanned) have been committed(http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_initial_design_notes.pdf)

commits as on 14 September 2013

Updated virgo malloc design handwritten nodes on kmalloc() and malloc() usage in kernelspace and userspace execution mode of virgo_cloudexec service (http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_design_notes_2_14September2013.pdf). As described in handwritten notes, virgo_malloc() and related system calls might be needed when a large scale allocation of kernel memory is needed when in kernel space execution mode and large scale userspace memory when in user modes (function and executable modes). Thus a cloud memory pool both in user and kernel space is possible.

VIRGO virtual addressing

VIRGO virtual address is defined with the following datatype:

```
struct virgo_address
{
      int node_id;
      void* addr;
};
```

VIRGO address translation table is defined with following datatype:

```
struct virgo_addr_transtable
{
      int node_id;
      void* addr;
};
```

VIRGO memory pooling prototypical implementation

VIRGO memory pooling implementation as per the design notes committed as above is to be implemented as a prototype under separate directory

under drivers/virgo/memorypooling and $LINUX_SRC_ROOT/virgo_malloc. But the underlying code is more or less similar to drivers/virgo/cpupooling and $LINUX_SRC_ROOT/virgo_clone.

virgo_malloc() and related syscalls and virgo mempool driver connect to and listen on port different from cpupooling driver. Though all these code can be within cpupooling itself, mempooling is implemented as separate driver and co-exists with cpupooling on bootup (/etc/modules). This enables clear demarcation of functionalities for CPU and Memory virtualization.

Commits as on 17 September 2013

Initial untested prototype code - virgo_malloc and virgo mempool driver - for VIRGO Memory Pooling has been committed - copied and modified from virgo_clone client and kernel driver service.

Commits as on 19 September 2013

3.7.8 Kernel full build done and compilation errors in VIRGO malloc and mempool driver code and more functions code added

Commits as on 23 September 2013

Updated virgo_malloc.c with two functions, int_to_str() and addr_to_str(), using kmalloc() with full kernel re-build.
(Rather a re-re-build because some source file updates in previous build got deleted somehow mysteriously. This could be related to Cybercrime issues mentioned in https://sourceforge.net/p/usb-md/code-0/HEAD/tree/USBmd_notes.txt)

Commits as on 24 September 2013

Updated syscall*.tbl files, staging.sh, Makefiles for virgo_malloc(),virgo_set(),virgo_get() and virgo_free() memory pooling syscalls. New testcase test_virgo_malloc for virgo_malloc(), virgo_set(), virgo_get(), virgo_free() has been added to repository. This testcase might have to be updated if return type and args to virgo_malloc+ syscalls are to be changed.

Commits as on 25 September 2013

All build related errors fixed after kernel rebuild some changes made to function names to reflect their
names specific to memory pooling. Updated /etc/modules also has been committed to repository.

Commits as on 26 September 2013

Circular dependency error in standalone build of cpu pooling and memory pooling drivers fixed and
datatypes and declarations for CPU pooling and Memory Pooling drivers have been segregated into respective header files (virgo.h and
virgo_mempool.h with corresponding service header files) to avoid any dependency error.

Commits as on 27 September 2013

Major commits for Memory Pooling Driver listen port change and parsing VIRGO memory pooling commands have been done.

Commits as on 30 September 2013

New parser functions added for parameter parsing and initial testing on virgo_malloc() works with telnet client with logs in test_logs/

Commits as on 1 October 2013

Removed strcpy in virgo_malloc as ongoing bugfix for buffer truncation in syscall path.

Commits as on 7 October 2013

Fixed the buffer truncation error from virgo_malloc syscall to mempool driver service which was caused by
sizeof() for a char*. BUF_SIZE is now used for size in both syscall client and mempool kernel service.

Commits as on 9 October 2013 and 10 October 2013

Mempool driver kernelspace virgo mempool ops have been rewritten due to lack of facilities to return a
value from kernel thread function. Since mempool service already spawns a kthread, this seems to be sufficient. Also the iov.iov_len in virgo_malloc has been changed from BUF_SIZE to strlen(buf) since BUF_SIZE
causes the kernel socket to block as it waits for more data to be sent.

Commits as on 11 October 2013
---
sscanf format error for virgo_cloud_malloc() return pointer address and sock_release() null pointer exception has been rectified.
Added str_to_addr() utility function.

Commits as on 14 October 2013 and 15 October 2013

Updated todo list.

Rewritten virgo_cloud_malloc() syscall with:
- mutexed virgo_cloud_malloc() loop
- redefined virgo address translation table in virgo_mempool.h
- str_to_addr(): removed (void**) cast due to null sscanf though it should have worked

Commits as on 18 October 2013

Continued debugging of null sscanf - added str_to_addr2() which uses simple_strtoll() kernel function
for scanning pointer as long long from string and casting it to void*. Also more %p qualifiers where
added in str_to_addr() for debugging.

Based on latest test_virgo_malloc run, simple_strtoll() correctly parses the address string into a long long base 16 and then is reinterpret_cast to void*.
Logs in test/

Commits as on 21 October 2013

Kern.log for testing after vtranstable addr fix with simple_strtoll() added to repository and still the other %p qualifiers do not work and only simple_strtoll() parses the address correctly.

Commits as on 24 October 2013

Lot of bugfixes made to virgo_malloc.c for scanning address into VIRGO transtable and size computation. Testcase test_virgo_malloc.c has also been modified to do reinterpret cast of long long into (struct virgo_address*) and corresponding test logs have been added to repository under virgo_malloc/test.

Though the above sys_virgo_malloc() works, the return value is a kernel pointer if the virgo_malloc executes in the Kernel mode which is more likely than User mode (call_usermodehelper which is circuitous). Moreover copy_from_user() or

copy_to_user() may not be directly useful here as this is an address allocation routine. The long long reinterpret cast obfuscates the virgo_address(User or Kernel) as a large integer which is a unique id for the allocated memory on cloud. Initial testing of sys_virgo_set() causes a Kernel Panic as usual probably due to direct access of struct virgo_address*. Alternatives are to use only long long for allocation unique-id everywhere or do copy_to_user() or copy_from_user() of the address on a user supplied buffer. Also vtranstable can be made into a bucketed hash table that maps each alloc_id to a chained virgo malloc chunks than the present sequential addressing which is more similar to open addressing.

Commits as on 25 October 2013
---
virgo_malloc.c has been rewritten by adding a userspace __user pointer to virgo_get() and virgo_set() syscalls which are internally copied with copy_from_user() and copy_to_user() kernel function to get and set userspace from kernelspace.Header file syscalls.h has been updated with changed syscalls prototypes.Two functions have been added to map a VIRGO address to a unique virgo identifier and viceversa for abstracting hardware addresses from userspace as mentioned in previous commit notes. VIRGO cloud mempool kernelspace driver has been updated to use virgo_mempool_args* instead of void* and VIRGO cloudexec mempool driverhas been updated accordingly during intermodule invocation.The virgo_malloc syscall client has been updated to modified signatures and return types for all mempool alloc,get,set,free syscalls.

Commits as on 29 October 2013

Miscellaneous ongoing bugfixes for virgo_set() syscall error in copy_from_user().

Commits as on 2 November 2013

Due to an issue which corrupts the kernel memory, presently telnet path to VIRGO mempool driver has been
tested after commits on 31 October 2013 and 1 November 2013 and is working but again there is an issue in kstrtoul() that returns the wrong address in virgo_cloud_mempool_kernelspace.ko that gives the address for
data to set.

Commits as on 6 November 2013

New parser function virgo_parse_integer() has been added to virgo_cloud_mempool_kernelspace driver module which is carried over from lib/kstrtox.c and modified locally to add an if clause to discard quotes and unquotes. With this the telnet path commands for virgo_malloc()
and virgo_set() are working. Today's kern.log has been added to repository in test_logs/.

Commits as on 7 November 2013

In addition to virgo_malloc and virgo_set, virgo_get is also working through telnet path after today's commit for "virgodata:" prefix in virgo_cloud_mempool_kernelspace.ko. This prefix is needed to differentiate data and address so that toAddressString() can be invoked to sprintf() the address in virgo_cloudexec_mempool.ko. Also mempool command parser has been updated to strcmp() virgo_cloud_get command also.

Commits as on 11 November 2013

More testing done on telnet path for virgo_malloc, virgo_set and virgo_get commands which work correctly. But there seem to be unrelated kmem_cache_trace_alloc panics that follow each successful virgo command execution. kern.log for this has been added to repository.

Commits as on 22 November 2013

More testing done on telnet path for virgo_malloc,virgo_set and virgo_set after commenting kernel socket shutdown code in the VIRGO cloudexec mempool sendto code. Kernel panics do not occur after commenting kernel socket shutdown.

Commits as on 2 December 2013

Lots of testing were done on telnet path and syscall path connection to VIRGO mempool driver and screenshots for working telnet path (virgo_malloc, virgo_set and virgo_get) have been committed to repository. Intriguingly, the syscall path is suddenly witnessing series of broken pipe erros, blocking errors etc., which are mostly Heisenbugs.

Commits as on 5 December 2013

More testing on system call path done for virgo_malloc(), virgo_set() and virgo_get() system calls with test_virgo_malloc.c. All three syscalls work in syscall path after lot of bugfixes. Kern.log that has logs for allocating memory in remote cloud node with virgo_malloc, sets data "test_virgo_malloc_data" with virgo_set and retrieves data with virgo_get.

VIRGO version 12.0 tagged.

Commits as on 12 March 2014

Initial VIRGO queueing driver implemented that flips between two internal queues: 1) a native queue implemented locally and 2) wrapper around linux kernel's workqueue facility 3) push_request() modified to pass on the request data to the workqueue handler using container_of on a wrapper structure virgo_workqueue_request.

Commits as on 20 March 2014

- VIRGO queue with additional boolean flags for its use as KingCobra queue
- KingCobra kernel space driver that is invoked by the VIRGO workqueue handler

Commits as on 30 March 2014

- VIRGO mempool driver has been augmented with use_as_kingcobra_service flags in CPU pooling and Memory pooling drivers

Commits as on 6 April 2014

- VIRGO mempool driver recvfrom() function's if clause for KingCobra has been updated for REQUEST header formatting mentioned in KingCobra design notes

Commits as on 7 April 2014

- generate_logical_timestamp() function has been implemented in VIRGO mempool driver that generates timestamps based on 3 boolean flags. At present machine_timestamp is generated and prepended to the request to be pushed to VIRGO queue driver and then serviced by KingCobra.

Commits as on 25 April 2014

- client ip address in VIRGO mempool recvfrom KingCobra if clause is converted to host byte order from network byte order with ntohl()

Commits as on 5 May 2014

- Telnet path commands for VIRGO cloud file system - virgo_cloud_open(),

virgo_cloud_read(), virgo_cloud_write(), virgo_cloud_close() has been implemented and test logs have been added to repository (drivers/virgo/cloudfs/ and cloudfs/testlogs) and kernel upcall path for paramIsExecutable=0

Commits as on 7 May 2014

- Bugfixes to tokenization in kernel upcall plugin with strsep() for args passed on to the userspace

Commits as on 8 May 2014

- Bugfixes to virgo_cloud_fs.c for kernel upcall (parameterIsExecutable=0) and with these the kernel to userspace upcall and writing to a file in userspace (virgofstest.txt) works. Logs and screenshots for this are added to repository in test_logs/

Commits as on 6 June 2014

- VIRGO File System Calls Path implementation has been committed. Lots of Linux Full Build compilation errors fixed and new integer parsing functionality added (similar to driver modules). For the timebeing all syscalls invoke loadbalancer. This may be further optimized with a sticky flag to remember the first invocation which might be usually virgo_open syscall to get the VFS descriptor that is used in subsequent syscalls.

Commits as on 3 July 2014

- More testing and bugfixes for VIRGO File System syscalls have been done. virgo_write() causes kernel panic.

7 July 2014 - virgo_write() kernel panic notes:
---
warning within http://lxr.free-electrons.com/source/arch/x86/kernel/smp.c#L121:

```
static void native_smp_send_reschedule(int cpu)
{
        if (unlikely(cpu_is_offline(cpu))) {
                WARN_ON(1);
                return;
        }
        apic->send_IPI_mask(cpumask_of(cpu), RESCHEDULE_VECTOR);
}
```

This is probably a fixed kernel bug in <3.7.8 but recurring in 3.7.8:
- http://lkml.iu.edu/hypermail/linux/kernel/1205.3/00653.html
- http://www.kernelhub.org/?p=3&msg=74473&body_id=72338
- http://lists.openwall.net/linux-kernel/2012/09/07/22
- https://bugzilla.kernel.org/show_bug.cgi?id=54331
- https://bbs.archlinux.org/viewtopic.php?id=156276

Commits as on 29 July 2014

All VIRGO drivers(cloudfs, queuing, cpupooling and memorypooling) have been built on 3.15.5 kernel with some Makefile changes for ccflags and paths

---
Commits as on 17 August 2014

(FEATURE - DONE) VIRGO Kernel Modules and System Calls major rewrite for 3.15.5 kernel - 17 August 2014

---
1. VIRGO config files have been split into /etc/virgo_client.conf and /etc/virgo_cloud.conf to delink the cloud client and kernel service config parameters reading and to do away with oft occurring symbol lookup errors and multiple definition errors for num_cloud_nodes and node_ip_addrs_in_cloud - these errors are frequent in 3.15.5 kernel than 3.7.8 kernel.

2. Each VIRGO module and system call now reads the config file independent of others - there is a read_virgo_config_<module>_<client_or_service>() function variant for each driver and system call. Though at present smacks of a replicated code, in future the config reads for each component (system call or module) might vary significantly depending on necessities.

3. New kernel module config has been added in drivers/virgo. This is for future prospective use as a config export driver that can be looked up by any other VIRGO module for config parameters.

4. include/linux/virgo_config.h has the declarations for all the config variables declared within each of the VIRGO kernel modules.

5. Config variables in each driver and system call have been named with prefix and suffix to differentiate the module and/or system call it serves.

6. In geographically distributed cloud virgo_client.conf has to be in client nodes and virgo_cloud.conf has to be in cloud nodes. For VIRGO Queue - KingCobra REQUEST-REPLY peer-to-peer messaging system same node can have virgo_client.conf and virgo_cloud.conf.

7. Above segregation largely simplifies the build process as each module and system call is independently built without need for a symbol to be exported from other module by pre-loading it.

8. VIRGO File system driver and system calls have been tested with above changes and the virgo_open(),virgo_read() and virgo_write() calls work with much less crashes and freeze problems compared to 3.7.8 (some crashes in VIRGO FS syscalls in 3.7.8 where already reported kernel bugs which seem to have been fixed in 3.15.5). Today's kern.log test logs have been committed to repository.

Committed as on 23 August 2014

Commenting use_as_kingcobra_service if clauses temporarily as disabling also doesnot work and only commenting the block works for VIRGO syscall path. Quite weird as to how this relates to the problem. As this is a heisenbug further testing is difficult and sufficient testing has been done with logs committed to repository. Probably a runtime symbol lookup for kingcobra causes the freeze.
For forwarding messages to KingCobra and VIRGO queues, cpupooling driver is sufficient which also has the use_as_kingcobra_service clause.

Committed as on 23 August 2014 and 24 August 2014

As cpupooling driver has the same crash problem with kernel_accept() when KingCobra has benn enabled, KingCobra clauses have been commented in both cpupooling and memorypooling drivers. Instead queueing driver has been updated with a kernel service infrastructure to accept connections at port 60000. With this following paths are available for KingCobra requests:

VIRGO cpupooling or memorypooling ====> VIRGO Queue =====> KingCobra

(or)

VIRGO Queue kernel service ==========================> KingCobra

---
Committed as on 26 August 2014

- all kmallocs have been made into GFP_ATOMIC instead of GFP_KERNEL
- moved some kingcobra related header code before kernel_recvmsg()
- some header file changes for set_fs()

This code has been tested with modified code for KingCobra and the standalone kernel service that accepts requests from telnet directly at port 60000, pushes to virgo_queue
and is handled to invoke KingCobra servicerequest kernelspace function, works (the kernel_recvmsg() crash was most probably due to Read-Only filesystem -errno printed is -30)

VIRGO version 14.9.9 has been release tagged on 9 September 2014

Committed as on 26 November 2014

New kernel module cloudsync has been added to repository under drivers/virgo that can be used for synchronization(lock() and unlock()) necessities in VIRGO cloud. Presently Bakery Algorithm has been implemented.

Committed as on 27 November 2014
---
virgo_bakery.h bakery_lock() has been modified to take 2 parameters - thread_id and number of for loops (1 or 2)

Committed as on 2 December 2014

VIRGO bakery algorithm implementation has been rewritten with some bugfixes. Sometimes there are soft lockup errors due to looping in kernel time durations for which are kernel build configurable.

Committed as on 17 December 2014

Initial code commits for VIRGO EventNet kernel module service:

1.EventNet Kernel Service listens on port 20000

2.It receives eventnet log messages from VIRGO cloud nodes and writes the log messages
after parsing into two text files /var/log/eventnet/EventNetEdges.txt and /var/log/eventnet/EventNetVertices.txt by VFS calls

3.These text files can then be processed by the EventNet implementations in AsFer (python pygraph and
C++ boost::graph based)

4.Two new directories virgo/utils and virgo/eventnet have been added.

5.virgo/eventnet has the new VIRGO EventNet kernel module service implementation that listens on
port 20000.

6.virgo/utils is the new generic utilities driver that has a virgo_eventnet_log()

exported function which connects to EventNet kernel service and sends the vertex and edge eventnet
log messages which are parsed by kernel service and written to the two text files above.

7.EventNet log messages have two formats:
   - Edge message - "eventnet_edgemsg#<id>#<from_event>#<to_event>"
   - Vertex message - "eventnet_vertextmsg#<id>-<partakers csv>-<partaker conversations csv>"

8.The utilities driver Module.symvers have to be copied to any driver which are then merged with the symbol files of the corresponding driver. Target clean has to be commented while
building the unified Module.symvers because it erases symvers carried over earlier.

9.virgo/utils driver can be populated with all necessary utility exported functions that might be needed
in other VIRGO drivers.

10.Calls to virgo_eventnet_log() have to be #ifdef guarded as this is quite network intensive.

Commits as on 18 December 2014

Miscellaneous bugfixes,logs and screenshot

- virgo_cloudexec_eventnet.c - eventnet messages parser errors and eventnet_func bugs fixed
- virgo_cloud_eventnet_kernelspace.c - filp_open() args updated due to vfs_write() kernel panics. The vertexmessage vfs_write is done after looping through the vertice textfile and appending the conversation to the existing vertex.Some more code has to be added.
- VIRGO EventNet build script updated for copying Module.symvers from utils driver for merging with eventnet Module.symvers during Kbuild
- Other build generated sources and kernel objects
- new testlogs directory with screenshot for edgemsg sent to EventNet kernel service and kern.log with previous history for vfs_write() panics due to permissions and the logs for working filp_open() fixed version
- vertex message update

Commits as on 2,3,4 January 2015

- fixes for virgo eventnet vertex and edge message text file vfs_write() errors
- kern.logs and screenshots

VIRGO version 15.1.8 release tagged on 8 January 2015

Commits as on 3 March 2015 - Initial commits for Kernel Analytics Module which reads the /etc/virgo_kernel_analytics.conf config (and) VIRGO memorypooling Key-Value Store Architecture Diagram
---
- Architecture of Key-Value Store in memorypooling (virgo_malloc,virgo_get,virgo_set,virgo_free) has been
uploaded as a diagram at
http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGOLinuxKernel_KeyValueStore_and_Modules_Interaction.jpg

- new kernel_analytics driver for AsFer <=> VIRGO+USBmd+KingCobra interface has been added.
- virgo_kernel_analytics.conf having csv(s) of key-value pairs of analytics variables is set by AsFer or any other Machine Learning code. With this VIRGO Linux Kernel is endowed with abilities to dynamically evolve than being just a platform for user code. Implications are huge - for example, a config variable "MaxNetworkBandwidth=255" set by the ML miner in userspace based on a Perceptron or Logistic Regression executed on network logs can be read by a kernel module that limits the network traffic to 255Mbps. Thus kernel dynamically changes behaviour.
- kernel_analytics Driver build script has been added

Commits as on 6 March 2015

- code has been added in VIRGO config module to import EXPORTed kernel_analytics config key-pair array
set by Apache Spark (mined from Uncomplicated Fire Wall logs) and manually and write to kern.log.

NeuronRain version 15.6.15 release tagged

Portability to linux kernel 4.0.5

The VIRGO kernel module drivers are based on kernel 3.15.5. With kernel 4.0.5 kernel which is the latest following
compilation and LD errors occur - this is on cloudfs VIRGO File System driver :
- msghdr has to be user_msghdr for iov and iov_len as there is a segregation of msghdr
- modules_install throws an error in scripts/Makefile.modinst while overwriting already installed module

---
Commits as on 9 July 2015

VIRGO cpupooling driver has been ported to linux kernel 4.0.5 with msghdr changes as mentioned previously
with kern.log for VIRGO cpupooling driver invoked in parameterIsExecutable=2 (kernel module invocation)
added in testlogs

Commits as on 10,11 July 2015

VIRGO Kernel Modules:
- memorypooling
- cloudfs
- utils
- config
- kernel_analytics
- cloudsync
- eventnet
- queuing
along with cpupooling have been ported to Linux Kernel 4.0.5 - Makefile and header files have been
updated wherever required.

Commits as on 20,21,22 July 2015

Due to SourceForge Storage Disaster(http://sourceforge.net/blog/sourceforge-infrastructure-and-service-restoration/),
the github replica of VIRGO is urgently updated with some important changes for msg_iter,iovec
etc., in 4.0.5 kernel port specifically for KingCobra and VIRGO Queueing. These have to be committed to SourceForge Krishna_iResearch
repository at http://sourceforge.net/users/ka_shrinivaasan once SourceForge repos are restored.
Time to move on to the manufacturing hub? GitHub ;-)

VIRGO Queueing Kernel Module Linux Kernel 4.0.5 port:

- msg_iter is used instead of user_msghdr
- kvec changed to iovec
- Miscellaneous BUF_SIZE related changes
- kern.logs for these have been added to testlogs
- Module.symvers has been recreated with KingCobra Module.symvers from 4.0.5 KingCobra build
- clean target commented in build script as it wipes out Module.symvers
- updated .ko and .mod.c

KingCobra Module Linux Kernel 4.0.5 port

- vfs_write() has a problem in 4.0.5
- the filp_open() args and flags which were working in 3.15.5 cause a kernel panic implicitly and nothing was written to logs
- It took a very long time to figure out the reason to be vfs_write and filp_open
- O_CREAT, O_RDWR and O_LARGEFILE cause the panic and only O_APPEND is working, but
does not do vfs_write(). All other VIRGO Queue + KingCobra functionalities work viz.,
enqueueing, workqueue handler invocation, dequeueing, invoking kingcobra kernelspace service
request function from VIRGO queue handler, timestamp, timestamp and IP parser, reply_to_publisher etc.,
- As mentioned in Greg Kroah Hartman's "Driving me nuts", persistence in Kernel space is
a bad idea but still seems to be a necessary stuff - yet only vfs calls are used which have to be safe
- Thus KingCobra has to be in-memory only in 4.0.5 if vfs_write() doesn't work
- Intriguingly cloudfs filesystems primitives - virgo_cloud_open, virgo_cloud_read, virgo_cloud_write etc.,
work perfectly and append to a file.
- kern.logs for these have been added to testlogs
- Module.symvers has been recreated for 4.0.5
- updated .ko and .mod.c

Due to SourceForge outage and for a future code diversification
NeuronRain codebases (AsFer, USBmd, VIRGO, KingCobra)
in http://sourceforge.net/u/userid-769929/profile/ have been
replicated in GitHub also - https://github.com/shrinivaasanka
excluding some huge logs due to Large File Errors in GitHub.

Commits as on 30 July 2015

VIRGO system calls have been ported to Linux Kernel 4.0.5 with commented gcc option -Wimplicit-function-declaration,
msghdr and iovec changes similar to drivers mentioned in previous commit notes above. But Kernel 4.1.3 has some Makefile and build issues.
The NeuronRain codebases in SourceForge and GitHub would henceforth be mostly

and always out-of-sync and not guaranteed to be replicas - might get diversified into different research and development directions (e.g one codebase might be more focussed on IoT while the other towards enterprise bigdata analytics integration with kernel and training which is yet to be designed- depend on lot of constraints)

Commits as on 2,3 August 2015

- new .config file added which is created from menuconfig
- drivers/Kconfig has been updated with 4.0.5 drivers/Kconfig for trace event linker errors
Linux Kernel 4.0.5 - KConfig is drivers/ has been updated to resolve RAS driver trace event linker error. RAS was not included in KConfig earlier.
- link-vmlinux.sh has been replaced with 4.0.5 kernel version

---
Commits as on 12 August 2015

VIRGO Linux Kernel 4.1.5 port - related code changes - some important notes:

- Linux Kernel 4.0.5 build suddenly had a serious root shell drop error in initramfs which was not resolved by:
      - adding rootdelay in grub
      - disabling uuid for block devices in grub config
        - mounting in read/write mode in recovery mode
        - no /dev/mapper related errors
        - repeated exits in root shell
      - delay before mount of root device in initrd scripts
- mysteriously there were some firmware microcode bundle executions in ieucodetool
- Above showed a serious grub corruption or /boot MBR bug or 4.0.5 VIRGO kernel build problem
- Linux 4.0.x kernels are EOL-ed
- Hence VIRGO is ported to 4.1.5 kernel released few days ago
- Only minimum files have been changed as in commit log for Makefiles and syscall table and headers and a build script has been added
for 4.1.5:
    Changed paths:
    A buildscript_4.1.5.sh
    M linux-kernel-extensions/Makefile
    M linux-kernel-extensions/arch/x86/syscalls/Makefile
    M linux-kernel-extensions/arch/x86/syscalls/syscall_32.tbl
    M linux-kernel-extensions/drivers/Makefile
    M linux-kernel-extensions/include/linux/syscalls.h

- Above minimum changes were enough to build an overlay-ed Linux Kernel with VIRGO codebase

---
Commits as on 14,15,16 August 2015

Executed the minimum end-end telnet path primitives in Linux kernel 4.1.5 VIRGO code:
- cpu virtualization
- memory virtualization
- filesystem virtualization (updated filp_open flags)
and committed logs and screenshots for the above.

Commits as on 17 August 2015

VIRGO queue driver:
- Rebuilt Module.symvers

- kern.log for telnet request to VIRGO Queue + KingCobra queueing system in kernelspace

---
Commits as on 25,26 September 2015

VIRGO Linux Kernel 4.1.5 - memory system calls:

- updated testcases and added logs for syscalls invoked separately(malloc,set,get,free)
- The often observed unpredictable heisen kernel panics occur with 4.1.5 kernel too. The logs are 2.3G and
only grepped output is committed to repository.
- virgo_malloc.c has been updated with kstrdup() to copy the buf to iov.iov_base which was earlier
crashing in copy_from_iter() within tcp code. This problem did not happen in 3.15.5 kernel.
- But virgo_clone syscall code works without any changes to iov_base as above which does a strcpy()
 which is an internal memcpy() though. So what causes this crash in memory system calls alone
is a mystery.
- new insmod script has been added to load the VIRGO memory modules as necessary instead of at boot time.
- test_virgo_malloc.c and its Makefile has been updated.

VIRGO Linux Kernel 4.1.5 - filesystem calls- testcases and logs:

  - added insmod script for VIRGO filesystem drivers
  - test_virgo_filesystem.c has been updated for syscall numbers in 4.1.5 VIRGO kernel
  - virgo_fs.c syscalls code has been updated for iov.iov_base kstrdup() - without this there are kernel panics in copy_from_iter(). kern.log
testlogs have been added, but there are heisen kernel panics. The virgo syscalls are executed but not written to kern.log due to these crashes.
Thus execution logs are missing for VIRGO filesystem syscalls.

Commits as on 28,29 September 2015

VIRGO Linux Kernel 4.1.5 filesystem syscalls:

- Rewrote iov_base code with a separate iovbuf set to iov_base and strcpy()-ing the syscall command to iov_base similar to VIRGO
memory syscalls
- Pleasantly the same iovbuf code that crashes in memory syscalls works for VIRGO FS without crash.Thus both virgo_clone and virgo_filesystem
syscalls work without issues in 4.1.5 and virgo_malloc() works erratically in 4.1.5 which remains as issue.
- kern.log for VIRGO FS syscalls and virgofstest text file written by virgo_write() have been added to repository

VIRGO Linux 4.1.5 kernel memory syscalls:

- rewrote the iov_base buffer code for all VIRGO memory syscalls by allocating separate iovbuf and copying the message to it - this just replicates the virgo_clone syscall behaviour which works without any crashes mysteriously.
- did extensive repetitive tests that were frequented by numerous kernel panics and crashes
- The stability of syscalls code with 3.15.5 kernel appears to be completely absent in 4.1.5
- The telnet path works relatively better though

- Difference between virgo_clone and virgo_malloc syscalls despite having same kernel sockets code looks like a non-trivial bug and a kernel stability issue.
- kernel OOPS traces are quite erratic.
- Makefile path in testcase has been updated

Commits as on 4 October 2015

VIRGO Linux Kernel 4.1.5 - Memory System Calls:

- replaced copy_to_user() with a memcpy()
- updated the testcase with an example VUID hardcoded.
- str_to_addr2() is done on iov_base instead of buf which was causing NULL parsing
- kern.log with above resolutions and multiple VIRGO memory syscalls tests - malloc,get,set
- With above VIRGO malloc and set syscalls work relatively causing less number of random kernel panics
- return values of memory calls set to 0
- in virgo_get() syscall, memcpy() of iov_base is done to data_out userspace pointer
- kern.log with working logs for syscalls - virgo_malloc(), virgo_set(), virgo_get() but still there are random kernel panics
- Abridged kern.log for VIRGO Memory System Calls with 4.1.5 Kernel - shows example logs for virgo_malloc(), virgo_set() and virgo_get()

Commits as on 14 October 2015

VIRGO Queue Workqueue handler usermode clause has been updated with 4.1.5 kernel paths and kingcobra in user mode is enabled for invoking KingCobra Cloud Perfect Forwarding.

Commits as on 15 October 2015

- Updated VIRGO Queue kernel binaries and build generated sources
- virgo_queue.h has been modified for call_usermodehelper() - set_ds() and fd_install() have been uncommented for output redirection. Output redirection works but there are "alloc_fd: slot 1 not NULL!" errors at random (kern.log in kingcobra testlogs) which seems to be a new feature in 4.1.5 kernel. This did not happen in 3.7.8-3.15.5 kernels

---
Commits as on 3 November 2015

- kern.log for VIRGO kernel_analytics+config drivers which export the analytics variables from /etc/virgo_kernel_analytics.conf kernel-wide and print them in config driver has been added to config/testlogs

---
Commits as on 10 January 2016

NeuronRain VIRGO enterprise version 2016.1.10 released.

NeuronRain - AsFer commits for VIRGO - C++ and C Python extensions
- Commits as on 29 January 2016

(FEATURE - DONE) Python-C++-VIRGOKernel and Python-C-VIRGOKernel boost::python

and cpython implementations:

- It is a known idiom that Linux Kernel and C++ are not compatible.
- In this commit an important feature to invoke VIRGO Linux Kernel from userspace python libraries via two alternatives have been added.
- In one alternative, C++ boost::python extensions have been added to encapsulate access to VIRGO memory system calls - virgo_malloc(), virgo_set(), virgo_get(), virgo_free(). Initial testing reveals that C++ and Kernel are not too incompatible and all the VIRGO memory system calls work well though initially there were some errors because of config issues.
- In the other alternative, C Python extensions have been added that replicate boost::python extensions above in C - C Python with Linux kernel works exceedingly well compared to boost::python.
- This functionality is required when there is a need to set kernel analytics configuration variables learnt by AsFer Machine Learning Code dynamically without re-reading /etc/virgo_kernel_analytics.conf.
- This completes a major integration step of NeuronRain suite - request travel roundtrip to-and-fro top level machine-learning C++/python code and rock-bottom C linux kernel - bull tamed ;-).
- This kind of python access to device drivers is available for Graphics Drivers already on linux (GPIO - for accessing device states)
- logs for both C++ and C paths have been added in cpp_boost_python_extensions/ and cpython_extensions.
- top level python scripts to access VIRGO kernel system calls have been added in both directories:
        CPython - python cpython_extensions/asferpythonextensions.py
        C++ Boost::Python - python cpp_boost_python_extensions/asferpythonextensions.py
- .so, .o files with build commandlines(asferpythonextensions.build.out) for "python setup.py build" have been added in build lib and temp directories.
- main implementations for C++ and C are in cpp_boost_python_extensions/asferpythonextensions.cpp and cpython_extensions/asferpythonextensions.c

---
Commits as on 12 February 2016

Commits for Telnet/System Call Interface to VIRGO CPUPooling -> VIRGO Queue -> KingCobra
---
*) This was commented earlier for the past few years due to a serious kernel panic in previous kernel versions - <= 3.15.5
*) In 4.1.5 a deadlock between VIRGO CPUPooling and VIRGO queue driver init was causing following error in "use_as_kingcobra_service" clause :
        - "gave up waiting for virgo_queue init, unknown symbol push_request()"
*) To address this a new boolean flag to selectively enable and disable VIRGO Queue kernel service mode "virgo_queue_reactor_service_mode" has been added.
*) With this flag VIRGO Queue is both a kernel service driver and a standalone exporter of function symbols - push_request/pop_request
*) Incoming request data from telnet/virgo_clone() system call into cpupooling kernel service reactor pattern (virgo cpupooling listener loop) is treated as generic string and handed over to VIRGO queue and KingCobra which publishes it.
*) This resolves a long standing deadlock above between VIRGO cpupooling "use_as_kingcobra_service" clause and VIRGO queue init.
*) This makes virgo_clone() systemcall/telnet both synchronous and asynchronous - requests from telnet client/virgo_clone() system call can be either synchronous RPC functions executed on a remote cloud node in kernelspace (or) an asynchronous invocation through "use_as_kingcobra_service" clause path to VIRGO Queue driver which enqueues the data in kernel workqueue and subsequently popped

by KingCobra.
*) Above saves an additional code implementation for virgo_queue syscall paths - virgo_clone() handles, based on config selected, incoming data passed to it either as a remote procedure call or as a data that is pushed to VIRGO Queue/KingCobra pub-sub kernelspace.

Prerequisites:

- insmod kingcobra_main_kernelspace.ko
- insmod virgo_queue.ko compiled with flag virgo_queue_reactor_service_mode=1
  (when virgo_queue_reactor_service_mode=0, listens on port 60000 for direct telnet requests)
- insmod virgo_cloud_test_kernelspace.ko
- insmod virgo_cloudexec.ko (listens on port 10000)

Schematic Diagram

VIRGO clone system call/telnet client ---> VIRGO cpupooling(compiled with use_as_kingcobra_service=1) ------> VIRGO Queue kernel service (compiled with virgo_queue_reactor_service_mode=1) ---> Linux Workqueue handler ------> KingCobra

---

Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports

---

- Imported Kernel Analytics variables into CloudFS kernel module - printed in driver init()
- Module.symvers from kernel_analytics has been merged with CloudFS Module.symvers
- Logs for above has been added in cloudfs/test_logs/
- Makefile updated with correct fs path
- Copyleft notices updated

Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports

---

- Kernel Analytics driver exported variables have been imported in CPU virtualization driver
- Module.symvers from kernel_analytics has been merged with Module.symvers in cpupooling
- kern.log for this import added to cpupooling/virgocloudexec/test_logs/

Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports

---

- Imported kernel analytics variables into memory virtualization driver init() , exported from kernel_analytics driver
- build shell script updated
- logs added to test_logs/
- Module.symvers from kernel_analytics has been merged with memory driver Module.symvers

- Makefile updated

Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports
---
- Imported kernel analytics variables into VIRGO Queueing Driver
- logs for this added in test_logs/
- Makefile updated
- Module.symvers from kernel_analytics has been merged with Queueing driver's Module.symvers
- .ko, .o and build generated sources

Commits as on 16,17 February 2016

(FEATURE-DONE) Socket Buffer Debug Utility Function - uses linux skbuff facility
---
- In this commit a multipurpose socket buffer debug utility function has been added in utils driver and exported kernelwide.
- It takes a socket as function argument does the following:
      - dereference the socket buffer head of skbuff per-socket transmit data queue
      - allocate skbuff with alloc_skb()
      - reserve head room with skb_reserve()
      - get a pointer to data payload with skb_put()
      - memcpy() an example const char* to skbuff data
      - Iterate through the linked list of skbuff queue in socket and print headroom and data pointers
      - This can be used as a packet sniffer anywhere within VIRGO linux network stack
- Any skb_*() functions can be plugged-in here as deemed necessary.
- kern.log(s) which print the socket internal skbuff data have been added to a new testlogs/ directory
- .cmd files generated by kbuild

(FEATURE-DONE) Commits as on 24 February 2016
---
skbuff debug function in utils/ driver:
(*) Added an if clause to check NULLity of skbuff headroom before doing skb_alloc()
(*) kern.log for this commit has been added testlogs/
(*) Rebuilt kernel objects and sources

Commits as on 29 February 2016

(FEATURE-DONE) Software Analytics - SATURN Program Analysis added to VIRGO Linux kernel drivers
---
- SATURN (saturn.stanford.edu) Program Analysis and Verification software has been
integrated into VIRGO Kernel as a Verification+SoftwareAnalytics subsystem
- A sample driver that can invoke an exported function has been added in drivers
- saturn_program_analysis
- Detailed document for an example null pointer analysis usecase has been created in virgo-docs/VIRGO_SATURN_Program_Analysis_Integration.txt
-

linux-kernel-extensions/drivers/virgo/saturn_program_analysis/saturn_program_ana
lysis_trees/error.txt is the error report from SATURN
- SATURN generated preproc and trees are in
linux-kernel-extensions/drivers/virgo/saturn_program_analysis/preproc and
linux-kernel-extensions/drivers/virgo/saturn_program_analysis/
saturn_program_analysis_trees/

Commits as on 10 March 2016

- SATURN analysis databases (.db) for locking, memory and CFG analysis.
- DOT and PNG files for locking, memory and CFG analysis.
- new folder saturn_calypso_files/ has been added in saturn_program_analysis/
with new .clp files virgosaturncfg.clp and virgosaturnmemory.clp
- SATURN alias analysis .db files

(FEATURE-DONE) NEURONRAIN - ASFER Commits for VIRGO - CloudFS systems calls
integrated into Boost::Python C++ and Python CAPI invocations
---

AsFer Commits as on 30 May 2016

VIRGO CloudFS system calls have been added (invoked by unique number from
syscall_32.tbl) for C++ Boost::Python interface to
VIRGO Linux System Calls. Switch clause with a boolean flag has been introduced
to select either VIRGO memory or filesystem calls.
kern.log and CloudFS textfile Logs for VIRGO memory and filesystem invocations
from AsFer python have been committed to testlogs/

AsFer Commits as on 31 May 2016

Python CAPI interface to NEURONRAIN VIRGO Linux System Calls has been updated to
include File System open, read, write primitives also.
Rebuilt extension binaries, kern.logs and example appended text file have been
committed to testlogs/. This is exactly similar to
commits done for Boost::Python C++ interface. Switch clause has been added to
select memory or filesystem VIRGO syscalls.

(BUG - STABILITY ISSUES) Commits - 25 July 2016 - Static Analysis of VIRGO Linux
kernel for investigating heisencrashes

Initial Documentation for Smatch and Coccinelle kernel static analyzers executed
on VIRGO Linux kernel - to be updated
periodically with further analysis.

(BUG - STABILITY ISSUES) Commits - 1 August 2016 - VIRGO Linux Stability Issues
- Ongoing Random oops and panics investigation

---
1. GFP_KERNEL has been replaced with GFP_ATOMIC flags in kmem allocations.
2. NULL checks have been introduced in lot of places involving strcpy, strcat,

strcmp etc., to circumvent
buffer overflows.
3. Though this has stabilized the driver to some extent, still there are OOPS in unrelated places deep
with in kernel where paging datastructures are accessed - kmalloc somehow corrupts paging
4. OOPS are debugged via gdb as:
        4.1 gdb ./vmlinux /proc/kcore
        or
        4.2 gdb <loadable_kernel_module>.o
   followed by
        4.3 l *(address+offset in OOPS dump)
5. kern.log(s) for the above have been committed in tar.gz format and have numerous OOPS occurred during repetitive telnet and syscall invocation(boost::python C++) invocations of virgo memory system calls.
6. Paging related OOPS look like an offshoot of set_fs() encompassing the filp_open VFS calls.

(BUG-STABILITY ISSUES) Commits - 26 September 2016 - Ongoing Random Panic investigation
---
Further analysis on direct VIRGO memory cache primitives telnet invocation - problems are similar
to Boost::Python AsFer VIRGO system calls invocations.

(BUG-STABILITY ISSUES) Commits - 27 September 2016 - Ongoing Random Panic investigation
---
Analysis of VIRGO memory cache primitives reveal more inconsistencies in cacheline flushes between CPU and GPU.

Srinivasan Kannan (alias) Ka.Shrinivaasan (alias) Shrinivas Kannan
http://sites.google.com/site/kuja27

/
****
****
#-----

#NEURONRAIN VIRGO - Cloud, Machine Learning and Queue augmented Linux Kernel Fork-off
#This program is free software: you can redistribute it and/or modify
#it under the terms of the GNU General Public License as published by
#the Free Software Foundation, either version 3 of the License, or
#(at your option) any later version.
#This program is distributed in the hope that it will be useful,
#but WITHOUT ANY WARRANTY; without even the implied warranty of
#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#GNU General Public License for more details.
#You should have received a copy of the GNU General Public License
#along with this program. If not, see <http://www.gnu.org/licenses/>.
#-----

#K.Srinivasan

#NeuronRain Documentation and Licensing: http://neuronrain-documentation.readthedocs.io/en/latest/
#Personal website(research): https://sites.google.com/site/kuja27/
#---
**********/

885. VIRGO is an operating system kernel forked off from Linux kernel mainline to add cloud functionalities (system calls, modules etc.,) within kernel itself with machine learning, analytics, debugging, queueing support in the deepest layer of OSI stack i.e AsFer, USBmd, KingCobra together with VIRGO constitute the previous functionalities. Presently there seems to be no cloud implementation with fine-grained cloud primitives (system calls, modules etc.,) included in kernel itself though there are coarse grained clustering and SunRPC implementations available. VIRGO complements other Clustering and application layer cloud OSes like cloudstack, openstack etc., in these aspects - CloudStack and OpenStack can be deployed on a VIRGO Linux Kernel Cloud - OpenStack nova compute, neutron network, cinder/swift storage subsystems can be augmented to have additional drivers that invoke lowlevel VIRGO syscall and kernel module primitives (assuming there are no coincidental replications of functionalities) thereby acting as a foundation to application layer cloud.

886. Remote Device Invocation , which is an old terminlogy for Internet-Of-Things has already been experimented in SunRPC and KOrbit CORBA-in-linux-kernel kernel modules in old linux kernels (http://www.csn.ul.ie/~mark/fyp/fypfinal.html - with Solaris MC and example Remote Device Client-Server Module implementation). VIRGO Linux with the larger encompassing NeuronRain suite is an effort to provide a unified end-to-end application-to-kernel machine-learning propelled cloud and internet-of-things framework.

887. Memory pooling:

Memory pooling is proposed to be implemented by a new virgo_malloc() system call that transparently allocates a block of virtual memory from memory pooled from virtual memory scattered across individual machines part of the cloud.

888. CPU pooling or cloud ability in a system call:

Clone() system call is linux specific and internally it invokes sys_clone(). All fork(),vfork() and clone() system calls internally invoke do_fork(). A new system call virgo_clone() is proposed to create a thread transparently on any of the available machines on the cloud.This creates a thread on a free or least-loaded machine on the cloud and returns the results.

virgo_clone() is a wrapper over clone() that looks up a map of machines-to-loadfactor and get the host with least load and invokes clone() on a function on that gets executed on the host. Usual cloud implementations provide userspace API that have something similar to this - call(function,host). Loadfactor can be calculated through any of the prominent loadbalancing algorithm. Any example userspace code that uses clone() can be replaced with virgo_clone() and all such threads will be running in a cloud transparently.Presently Native POSIX threads library(NPTL) and older LinuxThreads thread libraries internally use clone().

Kernel has support for kernel space sockets with kernel_accept(), kernel_bind(), kernel_connect(), kernel_sendmsg() and kernel_recvmsg() that can be used inside a kernel module. Virgo driver implements virgo_clone() system call that does a kernel_connect() to a remote kernel socket already __sock_create()-d, kernel_bind()-ed and kernel_accept()-ed and does kernel_sendmsg() of the function details and kernel_recvmsg() after function has been executed by clone() in remote machine. After kernel_accept() receives a connection it reads the function and parameter details. Using these kthread_create() is executed in

the remote machine and results are written back to the originating machine. This is somewhat similar to SunRPC but adapted and made lightweight to suit virgo_clone() implementation without any external data representation.

Experimental Prototype
---
virgo_clone() system call and a kernel module virgocloudexec which implements Sun RPC interface have been implemented.

VIRGO - loadbalancer to get the host:ip of the least loaded node
---
889. Loadbalancer option 1 - Centralized loadbalancer registry that tracks load:

Virgo_clone() system call needs to lookup a registry or map of host-to-load and get the least loaded host:ip from it. This requires a load monitoring code to run periodically and update the map. If this registry is located on a single machine then simultaneous virgo_clone() calls from many machines on the cloud could choke the registry. Due to this, loadbalancer registry needs to run on a high-end machine. Alternatively,each machine can have its own view of the load and multiple copies of load-to-host registries can be stored in individual machines. Synchronization of the copies becomes a separate task in itself(Cache coherency). Either way gives a tradeoff between accuracy, latency and efficiency.

Many application level userspace load monitoring tools are available but as virgo_clone() is in kernel space, it needs to be investigated if kernel-to-kernel loadmonitoring can be done without userspace data transport.Most Cloud API explicitly invoke a function on a host. If this functionality is needed, virgo_clone() needs to take host:ip address as extra argument,but it reduces transparent execution.

(Design notes for LB option 1 handwritten by myself are at
:http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/MiscellaneousOpenSourceDesignAndAcademicResearchNotes.pdf)

890. Loadbalancer option 2 - Linux Psuedorandom number generator based load balancer(experimental) instead of centralized registry that tracks load:

Each virgo_clone() client has a PRG which is queried (/dev/random or /dev/urandom) to get the id of the host to send the next virgo_clone() function to be executed
Expected number of requests per node is derived as:

expected number of requests per node = summation(each_value_for_the_random_variable_for_number_of_requests * probability_for_each_value) where random variable ranges from 1 to k where N is number of processors and k is the number of requests to be distributed on N nodes

=expected number of requests per node = (math.pow(N, k+2) - k*math.pow(N,2) + k*math.pow(N,1) - 1) / (math.pow(N, k+3) - 2*math.pow(N,k+2) + math.pow(N,k+1))

This loadbalancer is dependent on efficacy of the PRG and since each request is uniformly, identically, independently distributed use of PRG
would distribute requests evenly. This obviates the need for loadtracking and coherency of the load-to-host table.

(Design notes for LB option 2 handwritten by myself at
:http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/MiscellaneousOpenSourceDesignAndAcademicResearchNotes.pdf)

(python script in virgo-python-src/)

****
891. Implemented VIRGO Linux components (as on 7 March 2016)
****
1. cpupooling virtualization - VIRGO_clone() system call and VIRGO cpupooling driver by which a remote procedure can be invoked in kernelspace.(port: 10000)
2. memorypooling virtualization - VIRGO_malloc(), VIRGO_get(), VIRGO_set(), VIRGO_free() system calls and VIRGO memorypooling driver by which kernel memory can be allocated in remote node, written to, read and freed - A kernelspace memcache-ing.(port: 30000)
3. filesystem virtualization - VIRGO_open(), VIRGO_read(), VIRGO_write(), VIRGO_close() system calls and VIRGO cloud filesystem driver by which file IO in remote node can be done in kernelspace.(port: 50000)
4. config - VIRGO config driver for configuration symbols export.
5. queueing - VIRGO Queuing driver kernel service for queuing incoming requests, handle them with workqueue and invoke KingCobra service routines in kernelspace. (port: 60000)
6. cloudsync - kernel module for synchronization primitives (Bakery algorithm etc.,) with exported symbols that can be used in other VIRGO cloud modules for critical section lock() and unlock()
7. utils - utility driver that exports miscellaneous kernel functions that can be used across VIRGO Linux kernel
8. EventNet - eventnet kernel driver to vfs_read()/vfs_write() text files for EventNet vertex and edge messages (port: 20000)
9. Kernel_Analytics - kernel module that reads machine-learnt config key-value pairs set in /etc/virgo_kernel_analytics.conf. Any machine learning software can be used to get the key-value pairs for the config. This merges three facets - Machine Learning, Cloud Modules in VIRGO Linux-KingCobra-USBmd , Mainline Linux Kernel
10. Testcases and kern.log testlogs for the above
11. SATURN program analysis wrapper driver.

Thus VIRGO Linux at present implements a minimum cloud OS (with cloud-wide cpu, memory and file system management) over Linux and potentially fills in a gap to integrate both software and hardware into cloud with machine learning and analytics abilities that is absent in application layer cloud implementations. Thus VIRGO cloud is an IoT operating system kernel too that enables any hardware to be remote controlled. Data analytics using AsFer can be done by just invoking requisite code from a kernelspace driver above and creating an updated driver binary (or) by kernel_analytics module which reads the userland machine-learnt config.

****
VIRGO Features (list is quite dynamic and might be rewritten depending on feasibility - longterm with no deadline)
****
892. (FEATURE - DONE-minimum separate config file support in client and kernel service)1. More Sophisticated VIRGO config file and read_virgo_config() has to be invoked on syscall clients virgo_clone and virgo_malloc also. Earlier config was being read by kernel module only which would work only on a single machine. A separate config module kernel service has been added for future use while exporting kernel-wide configuration related symbols. VIRGO config files have been split into /etc/virgo_client.conf and /etc/virgo_cloud.conf to delink the cloud client and kernel service config parameters reading and to do away with oft occurring symbol lookup errors and multiple definition errors for num_cloud_nodes and node_ip_addrs_in_cloud - these errors are frequent in 3.15.5 kernel than 3.7.8 kernel. Each VIRGO module and system call now reads the config file independent of others - there is a

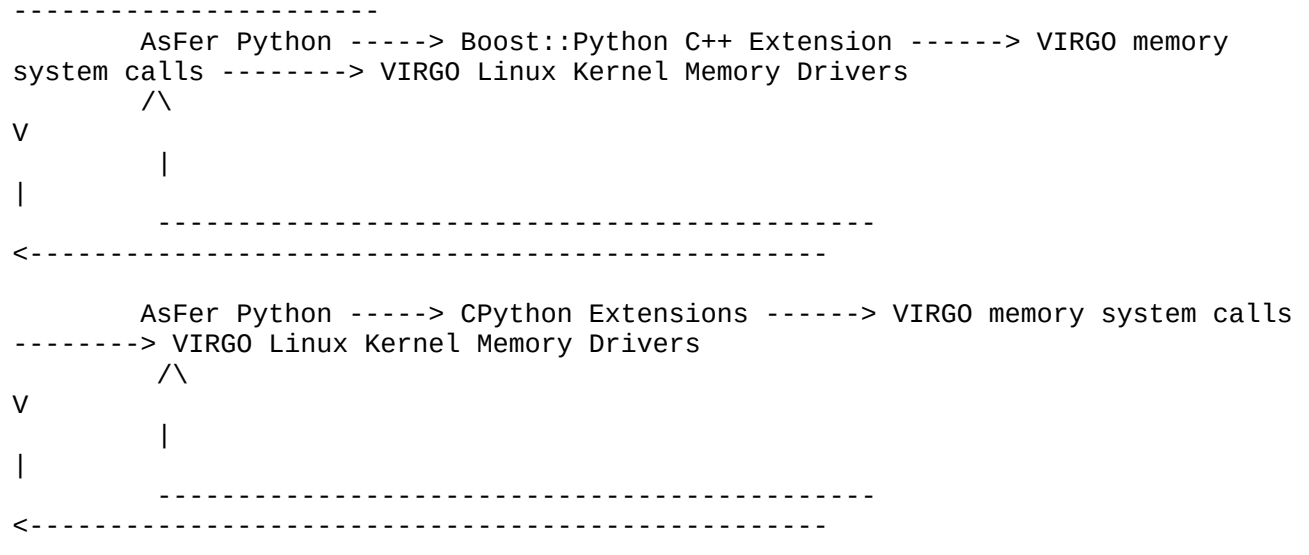
read_virgo_config_<module>_<client_or_service>() function variant for each driver and system call. Though at present smacks of a replicated code, in future the config reads for each component (system call or module) might vary significantly depending on necessities. New kernel module config has been added in drivers/virgo. This is for future prospective use as a config export driver that can be looked up by any other VIRGO module for config parameters. include/linux/virgo_config.h has the declarations for all the config variables declared within each of the VIRGO kernel modules. Config variables in each driver and system call have been named with prefix and suffix to differentiate the module and/or system call it serves. In geographically distributed cloud virgo_client.conf has to be in client nodes and virgo_cloud.conf has to be in cloud nodes. For VIRGO Queue - KingCobra REQUEST-REPLY peer-to-peer messaging system same node can have virgo_client.conf and virgo_cloud.conf. Above segregation largely simplifies the build process as each module and system call is independently built without need for a symbol to be exported from other module by pre-loading it.(- from commit comments done few months ago)

893. (FEATURE - Special case implementation DONE) 2. Object Marshalling and Unmarshalling (Serialization) Features - Feature 4 is a marshalling feature too as Python world PyObjects are serialized into VIRGO linux kernel and unmarshalled back bottom-up with CPython and Boost::Python C++ invocations - CPython and Boost internally take care of serialization.

894. (FEATURE - DONE) Virgo_malloc(), virgo_set(), virgo_get() and virgo_free() syscalls that virtualize the physical memory across all cloud nodes into a single logical memory behemoth (NUMA visavis UMA). (There are random crashes in copy_to_user and copy_from_user in syscall path for VIRGO memory pooling commands that were investigated but turned out to be mystery). These crashes have either been resolved or occur less in 3.15.5 and 4.1.5 kernels.
Initial Design Handwritten notes committed at: http://sourceforge.net/p/virgo-linux/code-0/210/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_initial_design_notes.pdf

895. (FEATURE - DONE) Integrated testing of AsFer-VIRGO Linux Kernel request roundtrip - invocation of VIRGO linux kernel system calls from AsFer Python via C++ or C extensions - Commits for this have been done on 29 January 2016. This unifies userlevel applications and kernelspace modules so that AsFer Python makes VIRGO linux kernel an extension. Following is schematic diagram and More details in commit notes below.

895.1 Schematic Diagram:



896. (FEATURE - DONE) Multithreading of VIRGO cloudexec kernel module (if not already done by kernel module subsystem internally)

897. (FEATURE - DONE) Sophisticated queuing and persistence of CPU and Memory pooling requests in Kernel Side (by possibly improving already existing kernel workqueues). Either open source implementations like ZeroMQ/ActiveMQ can be used or Queuing implementation has to be written from scratch or both. ActiveMQ supports REST APIs and is JMS implementation. This feature has been marked completed because recently NeuronRain AsFer backend has been updated to support KingCobra REQUEST_REPLY.queue as a datasource for Streaming Abstract Generator. By enabling use_as_kingcobra_service=1 in cpupooling and memorypooling VIRGO drivers, any incoming CPU and Memory related request can be routed to KingCobra by linux workqueue in VIRGO queue and disk persisted (/var/log/REQUEST_REPLY.queue) by KingCobra servicerequest recipient. Also Kafka Publisher/Subscriber have been implemented in NeuronRain AsFer which invoke Streaming Abstract Generator with KingCobra REQUEST_REPLY.queue as datasource to publish persisted already received CPU and Memory requests to Kafka Message Queue. Thus queuing and persistence for VIRGO CPU and Memory is in place. ZeroMQ does not have persistence and is used for NeuronRain client side Router-Dealer concurrent request servicing pattern.

898. (FEATURE - DONE-Minimum Functionality - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt) Integration of Asfer(AstroInfer) algorithm codes into VIRGO which would add machine learning capabilities into VIRGO - That is, VIRGO cloud subsystem which is part of a linux kernel installation "learns" and "adapts" to the processes that are executed on VIRGO. This catapults the power of the Kernel and Operating System into an artificially (rather approximately naturally) intelligent computing platform (a software "brain"). For example VIRGO can "learn" about "execution times" of processes and suitably act for future processes. PAC Learning of functions could be theoretical basis for this. Initial commits for Kernel Analytics Module which reads the /etc/virgo_kernel_analytics.conf config have been done. This config file virgo_kernel_analytics.conf having csv(s) of key-value pairs of analytics variables is set by AsFer or any other Machine Learning code. With this VIRGO Linux Kernel is endowed with abilities to dynamically evolve than being just a platform for user code. Implications are huge - for example, a config variable "MaxNetworkBandwidth=255" set by the ML miner in userspace based on a Perceptron or Logistic Regression executed on network logs can be read by a kernel module that limits the network traffic to 255Mbps. Thus kernel is no longer a static predictable blob behemoth. With this, VIRGO is an Internet-of-Things kernel that does analytics and based on analytics variable values integrated hardware can be controlled across the cloud through remote kernel module function invocation. This facility has been made dynamic with Boost::Python C++ and CPython extensions that permit flow of objects from machine learnt AsFer kernel analytics variables to VIRGO Linux Kernel memory drivers via VIRGO system calls directly and back - Commits on 29 January 2016 - this should obviate re-reading /etc/virgo_kernel_analytics.conf and is an exemplary implementation which unifies C++/Python into C/Kernel.

```
----------------------------------------
Example scenario 898.1 without implementation:
----------------------------------------
- Philips Hue IoT mobile app controlled bulb - http://www2.meethue.com/en-xx/
- kernel_analytics module learns key-value pairs from the AsFer code and exports it VIRGO kernel wide
- A driver function with in bulb embedded device driver can be invoked through VIRGO cpupooling (invoked from remote virgo_clone() system_call)
based on if-else clause of the kernel_analytics variable i.e remote_client invokes virgo_clone() with function argument "lights on" which is routed to another cloud node. The recipient cloud node "learns" from AsFer kernel_analytics that Voltage is low or Battery is low from logs and decides to switch in high beam or low beam.
----------------------------------------
Example scenario 898.2 without implementation:
----------------------------------------
```

- A swivel security camera driver is remotely invoked via virgo_clone() in the VIRGO cloud.
- The camera driver uses a machine learnt variable exported by kernel_analytics-and-AsFer to pan the camera by how much degrees.

Example scenario 898.3 without implementation - probably one of the best applications of NeuronRain IoT OS:

- Autonomous Driverless Automobiles - a VIRGO driver for a vehicle which learns kernel analytics variables (driving directions) set by kernel_analytics driver and AsFer Machine Learning. A naive algorithm for Driverless Car (with some added modifications over A-Star and Motion planning algorithms):
    - AsFer analytics receives obstacle distance data 360+360 degrees (vertical and horizontal) around the vehicle (e.g ultrasound sensors) which is updated in a Spark DataFrame table with high frequency (100 times per second).
    - VIRGO Linux kernel on vehicle has two special drivers for Gear-Clutch-Break-Accelerator-Fuel(GCBAF) and Steering listening on some ports.
    - AsFer analytics with high frequency computes threshold variables for applying break, clutch, gear, velocity, direction, fuel changes which are written to kernel_analytics.conf realtime based on distance data from Spark table.
    - These analytics variables are continuously read by GCBAF and Steering drivers which autopilot the vehicle.
    - Above applies to Fly-by-wire aeronautics too with appropriate changes in analytics variables computed.
    - The crucial parameter is the response time in variable computation and table updates which requires a huge computing power unless the vehicle is hooked onto a Spark cloud in motion by wireless which process the table and compute analytic variables.

E.g. Autopilot in Tesla Cars processes Petabytes of information (Smooth-as-Silk algorithm) from sensors which are fed to neural networks computed on a cloud - https://www.teslarati.com/tesla-firmware-v8-1-17-22-26-autopilot-2-0-smooth-silk-update-video/.

References for Machine Learning + Linux Kernel

898.4 KernTune - http://repository.uwc.ac.za/xmlui/bitstream/handle/10566/53/Yi_KernTune(2007).pdf?sequence=3
898.5 Self-learning, Predictive Systems - https://icri-ci.technion.ac.il/projects/past-projects/machine-learning-for-architecture-self-learning-predictive-computer-systems/
898.6 Linux Process Scheduling and Machine Learning - http://www.cs.ucr.edu/~kishore/papers/tencon.pdf
898.7 Network Latency and Machine Learning - https://users.soe.ucsc.edu/~slukin/rtt_paper.pdf
898.8 Machine Learning based Meta-Scheduler for Multicore processors - https://books.google.co.in/books?id=1GWcHmCrl0QC&pg=PA528&lpg=PA528&dq=linux+kernel+machine+learning&source=bl&ots=zfJsq_uu5q&sig=mMIUZ-oyJIwZXtYj4HntrQE8NSk&hl=en&sa=X&ved=0CCAQ6AEwATgKahUKEwjs9sqF9vPIAhVBFZQKHbNtA6A

899. A Symmetric Multi Processing subsystem Scheduler that virtualizes all nodes in cloud (probably this would involve improving the loadbalancer into a scheduler with priority queues)

900. (FEATURE - ONGOING) Virgo is an effort to virtualize the cloud as a single machine - Here cloud is not limited to servers and desktops but also mobile devices that run linux variants like Android, and other Mobile OSes. In the

longterm, Virgo may have to be ported or optimized for handheld devices. Boost::Python AsFer-VIRGO system call invocations implemented in NeuronRain is framework for implementing python applications interfacing with kernel. If deployed on Mobile processors (e.g by overlaying Android Kernel with VIRGO layer) there are IDEs like QPython to develop python apps for Android.

901. (FEATURE - DONE) Memory Pooling Subsystem Driver - Virgo_malloc(), Virgo_set(), Virgo_get() and Virgo_free() system calls and their Kernel Module Implementations. In addition to syscall path, telnet or userspace socket client interface is also provided for both VIRGO CPU pooling(virgo_clone()) and VIRGO Memory Pooling Drivers.

902. (FEATURE - DONE) Virgo Cloud File System with virgo_cloud_open(), virgo_cloud_read() , virgo_cloud_write() and virgo_cloud_close() commands invoked through telnet path has been implemented that transcends disk storage in all nodes in the cloud. It is also fanciful feature addition that would make VIRGO a complete all-pervading cloud platform. The remote telnet clients send the file path and the buf to be read or data to be written. The Virgo File System kernel driver service creates a unique Virgo File Descriptor for each struct file* opened by filp_open() and is returned to client. Earlier design option to use a hashmap (linux/hashmap.h) looked less attractive as file desciptor is an obvious unique description for open file and also map becomes unscalable. The kernel upcall path has been implemented (paramIsExecutable=0) and may not be necessary in most cases and all above cloudfs commands work in kernelspace using VFS calls.

903. (FEATURE - DONE) VIRGO Cloud File System commands through syscall paths - virgo_open(),virgo_close(),virgo_read() and virgo_write(). All the syscalls have been implemented with testcases and more bugs fixed. After fullbuild and testing, virgo_open() and virgo_read() work and copy_to_user() is working.

904. (FEATURE - DONE) VIRGO memory pooling feature is also a distributed key-value store similar to other prominent key-store software like BigTable implementations, Dynamo, memory caching tools etc., but with a difference that VIRGO mempool is implemented as part of Linux Kernel itself thus circumventing userspace latencies. Due to Kernel space VIRGO mempool has an added power to store and retrieve key-value pair in hardware devices directly which otherwise is difficult in userspace implementations.

905. VIRGO memory pooling can be improved with disk persistence for in-memory key-value store using virgo_malloc(),virgo_set(),virgo_get() and virgo_free() calls. Probably this might be just a set of invocations of read and write ops in disk driver or using sysfs. Probably this could be redundant as the VIRGO filesystem primitives have been implemented that write to a remote host's filesystem in kernelspace.

906. (FEATURE-DONE) Socket Debugging, Program Analysis and Verification features for user code that can find bugs statically. Socket skbuff debug utility and SATURN Program Analysis Software has been integrated into NEURONRAIN VIRGO Linux Kernel.

907. (FEATURE - DONE-Minimum Functionality) Operating System Logfile analysis using Machine Learning code in AstroInfer for finding patterns of processes execution and learn rules from the log. Kernel_Analytics VIRGO module reads /etc/virgo_kernel_analytics.conf config key-value pairs which are set by AsFer or other Machine Learning Software. At present an Apache Spark usecase that mines Uncomplicated Fire Wall logs in kern.log for most prominent source IP has been implemented in AsFer codebase : http://sourceforge.net/p/asfer/code/704/tree/python-src/SparkKernelLogMapReduceParser.py . This is set as a key-value config in /etc/virgo_kernel_analytics.conf read and exported by kernel_analytics module.

908. (USERSPACE C++ usecase implemented in GRAFIT course material - https://gitlab.com/shrinivaasanka/Grafit) Implementations of prototypical

Software Transactional Memory and LockFree Datastructures for VIRGO memory pooling.

909. Scalability features for Multicore machines - references: (http://halobates.de/lk09-scalability.pdf, http://pdos.csail.mit.edu/papers/linux:osdi10.pdf)

910. (USERSPACE C++ usecase implemented in GRAFIT course material - https://gitlab.com/shrinivaasanka/Grafit) Read-Copy-Update algorithm implementation for VIRGO memory pooling that supports multiple simultaneous versions of memory for readers - widely used in redesigned Linux Kernel.

911. (FEATURE - SATURN integration - minimum functionality DONE) Program Comprehension features as an add-on described in : https://sites.google.com/site/kuja27/PhDThesisProposal.pdf?attredirects=0. SATURN program analysis has been integrated into VIRGO linux with a stub driver.

912. (FEATURE - DONE) Bakery Algorithm implementation - cloudsync kernel module

913. (FEATURE - minimal EventNet Logical Clock primitive implemented in AstroInfer and this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt) Implementation of Distributed Systems primitives for VIRGO cloud viz., Logical Clocks, Termination Detection, Snapshots, Cache Coherency subsystem etc.,(as part of cloudsync driver module). Already a simple timestamp generation feature has been implemented for KingCobra requests with <ipaddress>:<localmachinetimestamp> format

914. (FEATURE - minimum functionality DONE - this section is an extended draft on respective packing/filling/tiling topics in NeuronRain AstroInfer design specific to kernel SLAB allocator - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt) Enhancements to kmem if it makes sense, because it is better to rely on virgo_malloc() for per machine memory management and wrap it around with a cloudwide VIRGO Unique ID based address lookup implementation of which is already in place.
Kernel Malloc syscall kmalloc() internally works as follows:
- kmem_cache_t object has pointers to 3 lists
- These 3 lists are full objects SLAB list, partial objects SLAB list and free objects SLAB list - all are lists of objects of same size
and cache_cache is the global list of all caches created thus far.
- Any kmalloc() allocation searches partial objects SLAB list and allocates a memory block with kmem_cache_alloc() from the first SLAB available - returned to caller.
- Any kfree() returns an object to a free SLAB list
- Full SLABs are removed from partial SLAB list and appended to full SLAB list
- SLABs are virtual memory pages created with kmem_cache_create
- Each SLAB in SLABs list has blocks of similar sized objects (e.g. multiples of two). Closest matching block is returned and fragmentation is minimized (incidentally this is the knapsack and packing optimization LP problem and thus NP-complete).

KERNELSPACE:
VIRGO address translation table already implements a tree registry of vtables each of capacity 3000 that keep track of kmalloc() allocations across all cloud nodes. Implementation of SLAB allocator for kmalloc() creates a kmem_cache(s) of similar sized objects and kmem_cache_alloc() allocates from these caches. kmalloc() already does lot of per-machine optimizations. VIRGO vtable registry tree maintained in VIRGO memory syscall end combined with per-machine kmalloc() cache_cache already look sufficient. Instrumenting kmem_cache_create() with #ifdef SLAB_CLOUD_MALLOC flags to do RPC looks superfluous. Hence marking this action item as done. Any further optimization can be done on top of existing VIRGO address translation table struct - e.g bookkeeping flags, function pointers etc.,.

USERSPACE: sbrk() and brk() are no longer used internally in malloc() library routines. Instead mmap() has replaced it (http://web.eecs.utk.edu/courses/spring2012/cs360/360/notes/Malloc1/lecture.html, http://web.eecs.utk.edu/courses/spring2012/cs360/360/notes/Malloc1/diff.html).

915. (FEATURE - ONGOING) Cleanup the code and remove unnecessary comments.

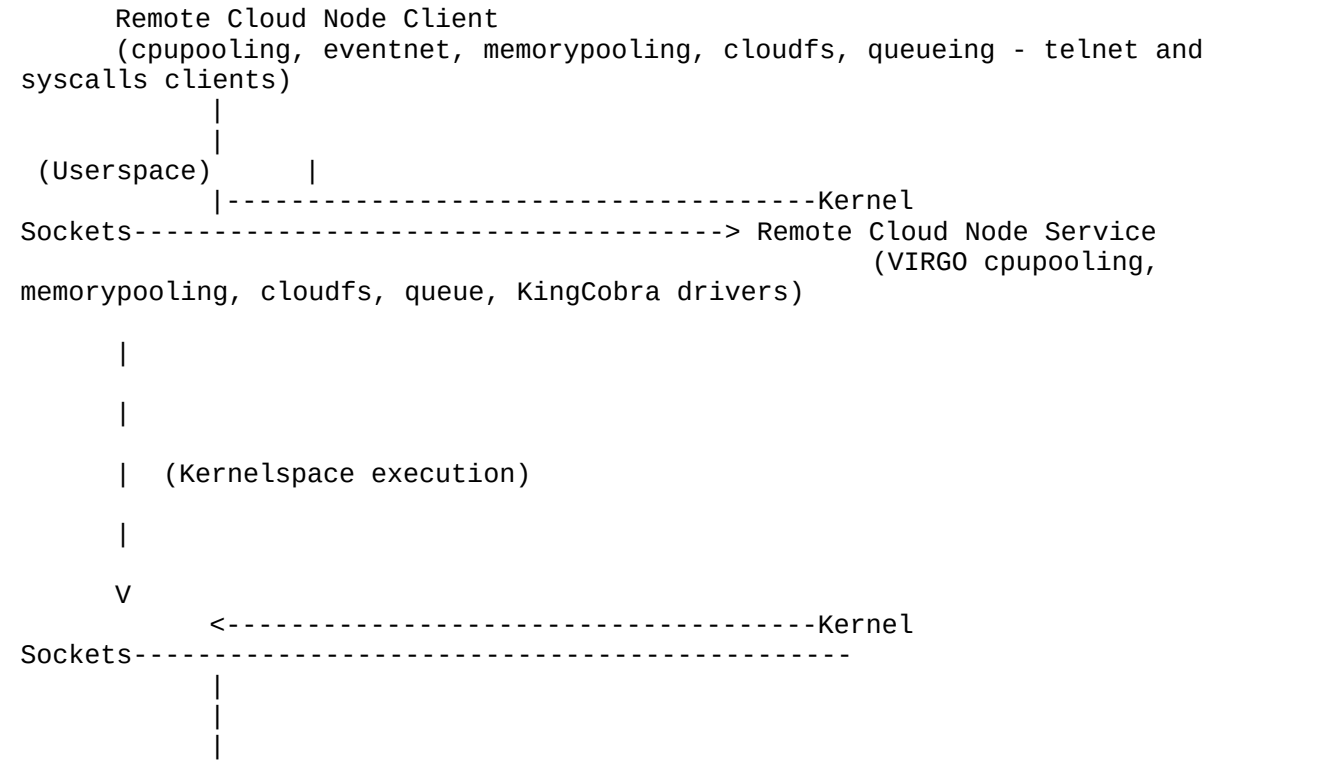
916. (FEATURE - DONE) Documentation - This design document is also a documentation for commit notes and other build and debugging technical details. Doxygen html cross-reference documentation for AsFer, USBmd, VIRGO, KingCobra and Acadpdrafts has been created along with summed-up design document and committed to GitHub Repository at https://github.com/shrinivaasanka/Krishna_iResearch_DoxygenDocs

917. (FEATURE - DONE) Telnet path to virgo_cloud_malloc,virgo_cloud_set and virgo_cloud_get has been tested and working. This is similar to memcached but stores key-value in kernelspace (and hence has the ability to write to and retrieve from any device driver memory viz., cards, handheld devices).An optional todo is to write a script or userspace socket client that connects to VIRGO mempool driver for these commands.

918. Augment the Linux kernel workqueue implementation (http://lxr.free-electrons.com/source/kernel/workqueue.c) with disk persistence if feasible and doesn't break other subsystems - this might require additional persistence flags in work_struct and additional #ifdefs in most of the queue functions that write and read from the disk. Related to item 6 above.

919. (FEATURE - DONE) VIRGO queue driver with native userspace queue and kernel workqueue-handler framework that is optionally used for KingCobra and is invoked through VIRGO cpupooling and memorypooling drivers. (Schematic in http://sourceforge.net/p/kcobra/code-svn/HEAD/tree/KingCobraDesignNotes.txt and http://sourceforge.net/p/acadpdrafts/code/ci/master/tree/Krishna_iResearch_opensourceproducts_archdiagram.pdf)

920. (FEATURE - DONE) KERNELSPACE EXECUTION ACROSS CLOUD NODES which geographically distribute userspace and kernelspace execution creating a logical abstraction for a cloudwide virtualized kernel:



(Userspace) |

921. (FEATURE - DONE) VIRGO platform as on 5 May 2014 implements a minimum set of features and kernelsocket commands required for a cloud OS kernel: CPU virtualization(virgo_clone), Memory virtualization(virgo_malloc,virgo_get,virgo_set,virgo_free) and a distributed cloud file system(virgo_open,virgo_close,virgo_read,virgo_write) on the cloud nodes and thus gives a logical view of one unified, distributed linux kernel across all cloud nodes that splits userspace and kernelspace execution across cloud as above.

922. (FEATURE - DONE) VIRGO Queue standalone kernel service has been implemented in addition to paths in schematics above. VIRGO Queue listens on hardcoded port 60000 and enqueues the incoming requests to VIRGO queue which is serviced by KingCobra:

VIRGO Queue client(e.g telnet) ------> VIRGO Queue kernel service ---> Linux Workqueue handler ------> KingCobra

923. (FEATURE - DONE) EventNet kernel module service:
VIRGO eventnet client (telnet) -------> VIRGO EventNet kernel service -----> EventNet graph text files

924. (FEATURE - DONE) Related to point 22 - Reuse EventNet cloudwide logical time infinite graph in AsFer in place of Logical clocks. At present the eventnet driver listens on port 20000 and writes the edges and vertices files in kernel using vfs_read()/vfs_write(). These text files can then be read by the AsFer code to generate DOT files and visualize the graph with graphviz.

925. (FEATURE - OPTIONAL) The kernel modules services listening on ports could return a JSON response when connected instead of plaintext, conforming to REST protocol. Additional options for protocol buffers which are becoming a standard data interchange format.

926. (FEATURE-Minimum Functionality DONE) Pointer Swizzling and Unswizzling of VIRGO addressspace pointers to/from VIRGO Unique ID (VUID). Presently VIRGO memory system calls implement a basic minimal pointer address translation to unique kmem location identifier.

****
CODE COMMIT RELATED NOTES
****

927. VIRGO code commits as on 16/05/2013
---
1. VIRGO cloudexec driver with a listener kernel thread service has been implemented and it listens on port 10000 on system startup through /etc/modules load-on-bootup facility

2. VIRGO cloudexec virgo_clone() system call has been implemented that would kernel_connect() to the VIRGO cloudexec service listening at port 10000

3. VIRGO cloudexec driver has been split into virgo.h (VIRGO typedefs), virgocloudexecsvc.h(VIRGO cloudexec service that is invoked by module_init() of VIRGO cloudexec driver) and virgo_cloudexec.c (with module ops definitions)

4. VIRGO does not implement SUN RPC interface anymore and now has its own virgo ops.

5. Lot of Kbuild related commits with commented lines for future use have been done viz., to integrate VIRGO to Kbuild, KBUILD_EXTRA_SYMBOLS for cross-module symbol reference.

928. VIRGO code commits as on 20/05/2013
---
1. test_virgo_clone.c testcase for sys_virgo_clone() system call works and connections are established to VIRGO cloudexec kernel module.

2. Makefile for test_virgo_clone.c and updated buildscript.sh for headers_install for custom-built linux.

929. VIRGO code commits as on 6/6/2013
---
1. Message header related bug fixes

930. VIRGO code commits as on 25/6/2013

1.telnet to kernel service was tested and found working
2.GFP_KERNEL changed to GFP_ATOMIC in VIRGO cloudexec kernel service

931. VIRGO code commits as on 1/7/2013
---
1. Instead of printing iovec, printing buffer correctly prints the messages
2. wake_up_process() added and function received from virgo_clone() syscall is executed with kernel_thread and results returned to
virgo_clone() syscall client.

932. commit as on 03/07/2013

PRG loadbalancer preliminary code implemented. More work to be done

933. commit as on 10/07/2013

Tested PRG loadbalancer read config code through telnet and virgo_clone. VFS code to read from virgo_cloud.conf commented for testing

934. commits as on 12/07/2013

PRG loadbalancer prototype has been completed and tested with test_virgo_clone and telnet and symbol export errors and PRG errors have been fixed

935. commits as on 16/07/2013
---
read_virgo_config() and read_virgo_clone_config()(replica of read_virgo_config()) have been implemented and tested to read the virgo_cloud.conf config parameters(at present the virgo_cloud.conf has comma separated list of ip addresses. Port is hardcoded to 10000 for uniformity across all nodes). Thus minimal cloud functionality with config file support is in place. Todo things include function pointer lookup in kernel service, more parameters to cloud config file if needed, individual configs for virgo_clone() and virgo kernel service, kernel-to-userspace upcall and execution instead of kernel space, performance tuning etc.,

936. commits as on 17/07/2013
---
moved read_virgo_config() to VIRGOcloudexec's module_init so that config is read at boot time and exported symbols are set beforehand.
Also commented read_virgo_clone_config() as it is redundant

937. commits as on 23/07/2013

Lack of reflection kind of facilities requires map of function_names to pointers_to_functions to be executed
on cloud has to be lookedup in the map to get pointer to function. This map is not scalable if number of functions are
in millions and size of the map increases linearly. Also having it in memory is both CPU and memory intensive.
Moreover this map has to be synchronized in all nodes for coherency and consistency which is another intensive task.
Thus name to pointer function table is at present not implemented. Suitable way to call a function by name of the function
is yet to be found out and references in this topic are scarce.

If parameterIsExecutable is set to 1 the data received from virgo_clone() is not a function but name of executable
This executable is then run on usermode using call_usermodehelper() which internally takes care of queueing the workstruct
and executes the binary as child of keventd and reaps silently. Thus workqueue component of kernel is indirectly made use of.
This is sometimes more flexible alternative that executes a binary itself on cloud and
is preferable to clone()ing a function on cloud. Virgo_clone() syscall client or telnet needs to send the message with name of binary.

If parameterIsExecutable is set to 0 then data received from virgo_clone() is name of a function and is executed in else clause
using dlsym() lookup and pthread_create() in user space. This unifies both call_usermodehelper() and creating a userspace thread
with a fixed binary which is same for any function. The dlsym lookup requires mangled function names which need to be sent by
virgo_clone or telnet. This is far more efficient than a function pointer table.

call_usermodehelper() Kernel upcall to usermode to exec a fixed binary that would inturn execute the cloneFunction in userspace
by spawning a pthread. cloneFunction is name of the function and not binary. This clone function will be dlsym()ed
and a pthread will be created by the fixed binary. Name of the fixed binary is hardcoded herein as
"virgo_kernelupcall_plugin". This fixed binary takes clone function as argument. For testing libvirgo.so has been created from
virgo_cloud_test.c and separate build script to build the cloud function binaries has been added.

 - Ka.Shrinivaasan (alias) Shrinivas Kannan (alias) Srinivasan Kannan
   (https://sites.google.com/site/kuja27)

938. commits as on 24/07/2013
----

test_virgo_clone unit test case updated with mangled function name to be sent to remote cloud node. Tested with test_virgo_clone
end-to-end and all features are working. But sometimes kernel_connect hangs randomly (this was observed only today and looks similar
to blocking vs non-blocking problem. Origin unknown).

- Ka.Shrinivaasan (alias) Shrinivas Kannan (alias) Srinivasan Kannan
  (https://sites.google.com/site/kuja27)

939. commits as on 29/07/2013

Added kernel mode execution in the clone_func and created a sample kernel_thread for a cloud function. Some File IO logging added to upcall
binaries and parameterIsExecutable has been moved to virgo.h

940. commits as on 30/07/2013

New usecase virgo_cloud_test_kernelspace.ko kernel module has been added. This exports a function virgo_cloud_test_kernelspace() and is
accessed by virgo_cloudexec kernel service to spawn a kernel thread that is executed in kernel addresspace. This Kernel mode execution
on cloud adds a unique ability to VIRGO cloud platform to seamlessly integrate hardware devices on to cloud and transparently send commands
to them from a remote cloud node through virgo_clone().

Thus above feature adds power to VIRGO cloud to make it act as a single "logical device driver" though devices are in geographically in a remote server.

941. commits as on 01/08/2013 and 02/08/2013

Added Bash shell commandline with -c option for call_usermodehelper upcall clauses to pass in remote virgo_clone command message as
arguments to it. Also tried output redirection but it works some times that too with a fatal kernel panic.

Ideal solutions are :
1. either to do a copy_from_user() for message buffer from user address space (or)
2. somehow rebuild the kernel with fd_install() pointing stdout to a VFS file* struct. In older kernels like 2.6.x, there is an fd_install code
with in kmod.c (___call_usermodehelper()) which has been redesigned in kernel 3.x versions and fd_install has been removed in kmod.c .
3. Create a Netlink socket listener in userspace and send message up from kernel Netlink socket.

All the above are quite intensive and time consuming to implement.Moreover doing FileIO in usermode helper is strongly discouraged in kernel docs

Since Objective of VIRGO is to virtualize the cloud as single execution "machine", doing an upcall (which would run with root abilities) is
redundant often and kernel mode execution is sufficient. Kernel mode execution with intermodule function invocation can literally take over
the entire board in remote machine (since it can access PCI bus, RAM and all other device cards)

As a longterm design goal, VIRGO can be implemented as a separate protocol itself and sk_buff packet payload from remote machine
can be parsed by kernel service and kernel_thread can be created for the message.

942. commits as on 05/08/2013:

Major commits done for kernel upcall usermode output logging with fd_install redirection to a VFS file. With this it has become easy for user space to communicate runtime data to Kernel space. Also a new strip_control_M() function has been added to strip \r\n or " ".

943. 11 August 2013:

Open Source Design and Academic Research Notes uploaded to http://sourceforge.net/projects/acadpdrafts/files/MiscellaneousOpenSourceDesignAndAcademicResearchNotes_2013-08-11.pdf/download

944. commits as on 23 August 2013

New Multithreading Feature added for VIRGO Kernel Service - action item 5 in ToDo list above (virgo_cloudexec driver module). All dependent headers changed

for kernel threadlocalizing global data.

945. commits as on 1 September 2013

GNU Copyright license and Product Owner Profile (for identity of license issuer) have been committed. Also Virgo Memory Pooling - virgo_malloc() related initial design notes (handwritten scanned) have been committed(http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_initial_design_notes.pdf)

946. commits as on 14 September 2013

Updated virgo malloc design handwritten nodes on kmalloc() and malloc() usage in kernelspace and userspace execution mode of virgo_cloudexec service (http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGO_Memory_Pooling_virgomalloc_design_notes_2_14September2013.pdf). As described in handwritten notes, virgo_malloc() and related system calls might be needed when a large scale allocation of kernel memory is needed when in kernel space execution mode and large scale userspace memory when in user modes (function and executable modes). Thus a cloud memory pool both in user and kernel space is possible.

---
947. VIRGO virtual addressing

VIRGO virtual address is defined with the following datatype:

```
struct virgo_address
{
      int node_id;
      void* addr;
};
```

VIRGO address translation table is defined with following datatype:

```
struct virgo_addr_transtable
{
      int node_id;
      void* addr;
};
```

---
948. VIRGO memory pooling prototypical implementation

VIRGO memory pooling implementation as per the design notes committed as above is to be implemented as a prototype under separate directory under drivers/virgo/memorypooling and $LINUX_SRC_ROOT/virgo_malloc. But the underlying code is more or less similar to drivers/virgo/cpupooling and $LINUX_SRC_ROOT/virgo_clone.

virgo_malloc() and related syscalls and virgo mempool driver connect to and listen on port different from cpupooling driver. Though all these code can be within cpupooling itself, mempooling is implemented as separate driver and co-exists with cpupooling on bootup (/etc/modules). This enables clear demarcation of functionalities for CPU and Memory virtualization.

949. Commits as on 17 September 2013

Initial untested prototype code - virgo_malloc and virgo mempool driver - for VIRGO Memory Pooling has been committed - copied and modified from virgo_clone client and kernel driver service.

950. Commits as on 19 September 2013

3.7.8 Kernel full build done and compilation errors in VIRGO malloc and mempool driver code and more functions code added

951. Commits as on 23 September 2013

Updated virgo_malloc.c with two functions, int_to_str() and addr_to_str(), using kmalloc() with full kernel re-build.
(Rather a re-re-build because some source file updates in previous build got deleted somehow mysteriously. This could be related to Cybercrime issues mentioned in https://sourceforge.net/p/usb-md/code-0/HEAD/tree/USBmd_notes.txt)

952. Commits as on 24 September 2013

Updated syscall*.tbl files, staging.sh, Makefiles for virgo_malloc(),virgo_set(),virgo_get() and virgo_free() memory pooling syscalls. New testcase test_virgo_malloc for virgo_malloc(), virgo_set(), virgo_get(), virgo_free() has been added to repository. This testcase might have to be updated if return type and args to virgo_malloc+ syscalls are to be changed.

953. Commits as on 25 September 2013

All build related errors fixed after kernel rebuild some changes made to function names to reflect their
names specific to memory pooling. Updated /etc/modules also has been committed to repository.

954. Commits as on 26 September 2013

Circular dependency error in standalone build of cpu pooling and memory pooling drivers fixed and
datatypes and declarations for CPU pooling and Memory Pooling drivers have been segregated into respective header files (virgo.h and
virgo_mempool.h with corresponding service header files) to avoid any dependency error.

955. Commits as on 27 September 2013

Major commits for Memory Pooling Driver listen port change and parsing VIRGO memory pooling commands have been done.

956. Commits as on 30 September 2013

New parser functions added for parameter parsing and initial testing on virgo_malloc() works with telnet client with logs in test_logs/

957. Commits as on 1 October 2013

Removed strcpy in virgo_malloc as ongoing bugfix for buffer truncation in syscall path.

958. Commits as on 7 October 2013

Fixed the buffer truncation error from virgo_malloc syscall to mempool driver service which was caused by
sizeof() for a char*. BUF_SIZE is now used for size in both syscall client and mempool kernel service.

959. Commits as on 9 October 2013 and 10 October 2013

Mempool driver kernelspace virgo mempool ops have been rewritten due to lack of facilities to return a
value from kernel thread function. Since mempool service already spawns a kthread, this seems to be sufficient. Also the iov.iov_len in virgo_malloc has been changed from BUF_SIZE to strlen(buf) since BUF_SIZE

causes the kernel socket to block as it waits for more data to be sent.

960. Commits as on 11 October 2013
---
sscanf format error for virgo_cloud_malloc() return pointer address and sock_release() null pointer exception has been rectified.
Added str_to_addr() utility function.

961. Commits as on 14 October 2013 and 15 October 2013

Updated todo list.

Rewritten virgo_cloud_malloc() syscall with:
- mutexed virgo_cloud_malloc() loop
- redefined virgo address translation table in virgo_mempool.h
- str_to_addr(): removed (void**) cast due to null sscanf though it should have worked

962. Commits as on 18 October 2013

Continued debugging of null sscanf - added str_to_addr2() which uses simple_strtoll() kernel function
for scanning pointer as long long from string and casting it to void*. Also more %p qualifiers where
added in str_to_addr() for debugging.

Based on latest test_virgo_malloc run, simple_strtoll() correctly parses the address string into a long long base 16 and then is reinterpret_cast to void*. Logs in test/

963. Commits as on 21 October 2013

Kern.log for testing after vtranstable addr fix with simple_strtoll() added to repository and still the other %p qualifiers do not work and only simple_strtoll() parses the address correctly.

964. Commits as on 24 October 2013

Lot of bugfixes made to virgo_malloc.c for scanning address into VIRGO transtable and size computation. Testcase test_virgo_malloc.c has also been modified to do reinterpret cast of long long into (struct virgo_address*) and corresponding test logs have been added to repository under virgo_malloc/test.

Though the above sys_virgo_malloc() works, the return value is a kernel pointer if the virgo_malloc executes in the Kernel mode which is more likely than User mode (call_usermodehelper which is circuitous). Moreover copy_from_user() or copy_to_user() may not be directly useful here as this is an address allocation routine. The long long reinterpret cast obfuscates the virgo_address(User or Kernel) as a large integer which is a unique id for the allocated memory on cloud. Initial testing of sys_virgo_set() causes a Kernel Panic as usual probably due to direct access of struct virgo_address*. Alternatives are to use only long long for allocation unique-id everywhere or do copy_to_user() or copy_from_user() of the address on a user supplied buffer. Also vtranstable can be made into a bucketed hash table that maps each alloc_id to a chained virgo malloc chunks than the present sequential addressing which is more similar to open addressing.

965. Commits as on 25 October 2013
---
virgo_malloc.c has been rewritten by adding a userspace __user pointer to virgo_get() and virgo_set() syscalls which are internally copied with copy_from_user() and copy_to_user() kernel function to get and set userspace from kernelspace.Header file syscalls.h has been updated with changed syscalls prototypes.Two functions have been added to map a VIRGO address to a unique

virgo identifier and viceversa for abstracting hardware addresses from userspace as mentioned in previous commit notes. VIRGO cloud mempool kernelspace driver has been updated to use virgo_mempool_args* instead of void* and VIRGO cloudexec mempool driverhas been updated accordingly during intermodule invocation.The virgo_malloc syscall client has been updated to modified signatures and return types for all mempool alloc,get,set,free syscalls.

966. Commits as on 29 October 2013

Miscellaneous ongoing bugfixes for virgo_set() syscall error in copy_from_user().

967. Commits as on 2 November 2013

Due to an issue which corrupts the kernel memory, presently telnet path to VIRGO mempool driver has been
tested after commits on 31 October 2013 and 1 November 2013 and is working but again there is an issue in kstrtoul() that returns the wrong address in virgo_cloud_mempool_kernelspace.ko that gives the address for
data to set.

968. Commits as on 6 November 2013

New parser function virgo_parse_integer() has been added to virgo_cloud_mempool_kernelspace driver module which is carried over from lib/kstrtox.c and modified locally to add an if clause to discard quotes and unquotes. With this the telnet path commands for virgo_malloc()
and virgo_set() are working. Today's kern.log has been added to repository in test_logs/.

969. Commits as on 7 November 2013

In addition to virgo_malloc and virgo_set, virgo_get is also working through telnet path after today's commit for "virgodata:" prefix in virgo_cloud_mempool_kernelspace.ko. This prefix is needed to differentiate data and address so that toAddressString() can be invoked to sprintf() the address in virgo_cloudexec_mempool.ko. Also mempool command parser has been updated to strcmp() virgo_cloud_get command also.

970. Commits as on 11 November 2013

More testing done on telnet path for virgo_malloc, virgo_set and virgo_get commands which work correctly. But there seem to be unrelated kmem_cache_trace_alloc panics that follow each successful virgo command execution. kern.log for this has been added to repository.

971. Commits as on 22 November 2013

More testing done on telnet path for virgo_malloc,virgo_set and virgo_set after commenting kernel socket shutdown code in the VIRGO cloudexec
mempool sendto code. Kernel panics do not occur after commenting kernel socket shutdown.

972. Commits as on 2 December 2013

Lots of testing were done on telnet path and syscall path connection to VIRGO mempool driver and screenshots for working telnet path (virgo_malloc, virgo_set and virgo_get) have been committed to repository. Intriguingly, the syscall path is suddenly witnessing series of broken pipe erros, blocking errors etc., which are mostly Heisenbugs.

973. Commits as on 5 December 2013

More testing on system call path done for virgo_malloc(), virgo_set() and

virgo_get() system calls with test_virgo_malloc.c. All three syscalls work in syscall path after lot of bugfixes. Kern.log that has logs for allocating memory in remote cloud node with virgo_malloc, sets data "test_virgo_malloc_data" with virgo_set and retrieves data with virgo_get.

VIRGO version 12.0 tagged.

974. Commits as on 12 March 2014

Initial VIRGO queueing driver implemented that flips between two internal queues: 1) a native queue implemented locally and 2) wrapper around linux kernel's workqueue facility 3) push_request() modified to pass on the request data to the workqueue handler using container_of on a wrapper structure virgo_workqueue_request.

975. Commits as on 20 March 2014

- VIRGO queue with additional boolean flags for its use as KingCobra queue
- KingCobra kernel space driver that is invoked by the VIRGO workqueue handler

976. Commits as on 30 March 2014

- VIRGO mempool driver has been augmented with use_as_kingcobra_service flags in CPU pooling and Memory pooling drivers

977. Commits as on 6 April 2014

- VIRGO mempool driver recvfrom() function's if clause for KingCobra has been updated for REQUEST header formatting mentioned in KingCobra design notes

978. Commits as on 7 April 2014

- generate_logical_timestamp() function has been implemented in VIRGO mempool driver that generates timestamps based on 3 boolean flags. At present machine_timestamp is generated and prepended to the request to be pushed to VIRGO queue driver and then serviced by KingCobra.

979. Commits as on 25 April 2014

- client ip address in VIRGO mempool recvfrom KingCobra if clause is converted to host byte order from network byte order with ntohl()

980. Commits as on 5 May 2014

- Telnet path commands for VIRGO cloud file system - virgo_cloud_open(), virgo_cloud_read(), virgo_cloud_write(), virgo_cloud_close() has been implemented and test logs have been added to repository (drivers/virgo/cloudfs/ and cloudfs/testlogs) and kernel upcall path for paramIsExecutable=0

981. Commits as on 7 May 2014

- Bugfixes to tokenization in kernel upcall plugin with strsep() for args passed on to the userspace

982. Commits as on 8 May 2014

- Bugfixes to virgo_cloud_fs.c for kernel upcall (parameterIsExecutable=0) and with these the kernel to userspace upcall and writing to a file in userspace (virgofstest.txt) works. Logs and screenshots for this are added to repository in test_logs/

983. Commits as on 6 June 2014

- VIRGO File System Calls Path implementation has been committed. Lots of Linux Full Build compilation errors fixed and new integer parsing functionality added (similar to driver modules). For the timebeing all syscalls invoke loadbalancer. This may be further optimized with a sticky flag to remember the first invocation which might be usually virgo_open syscall to get the VFS descriptor that is used in subsequent syscalls.

984. Commits as on 3 July 2014

- More testing and bugfixes for VIRGO File System syscalls have been done. virgo_write() causes kernel panic.

985. 7 July 2014 - virgo_write() kernel panic notes:
---
warning within http://lxr.free-electrons.com/source/arch/x86/kernel/smp.c#L121:

```
static void native_smp_send_reschedule(int cpu)
{
        if (unlikely(cpu_is_offline(cpu))) {
                WARN_ON(1);
                return;
        }
        apic->send_IPI_mask(cpumask_of(cpu), RESCHEDULE_VECTOR);
}
```

This is probably a fixed kernel bug in <3.7.8 but recurring in 3.7.8:
- http://lkml.iu.edu/hypermail/linux/kernel/1205.3/00653.html
- http://www.kernelhub.org/?p=3&msg=74473&body_id=72338
- http://lists.openwall.net/linux-kernel/2012/09/07/22
- https://bugzilla.kernel.org/show_bug.cgi?id=54331
- https://bbs.archlinux.org/viewtopic.php?id=156276

986. Commits as on 29 July 2014

All VIRGO drivers(cloudfs, queuing, cpupooling and memorypooling) have been built on 3.15.5 kernel with some Makefile changes for ccflags and paths

---
Commits as on 17 August 2014
---
987. (FEATURE - DONE) VIRGO Kernel Modules and System Calls major rewrite for 3.15.5 kernel - 17 August 2014
---
1. VIRGO config files have been split into /etc/virgo_client.conf and /etc/virgo_cloud.conf to delink the cloud client and kernel service config parameters reading and to do away with oft occurring symbol lookup errors and multiple definition errors for num_cloud_nodes and node_ip_addrs_in_cloud - these errors are frequent in 3.15.5 kernel than 3.7.8 kernel.

2. Each VIRGO module and system call now reads the config file independent of others - there is a read_virgo_config_<module>_<client_or_service>() function variant for each driver and system call. Though at present smacks of a replicated code, in future the config reads for each component (system call or module) might vary significantly depending on necessities.

3. New kernel module config has been added in drivers/virgo. This is for future prospective use as a config export driver that can be looked up by any other VIRGO module for config parameters.

4. include/linux/virgo_config.h has the declarations for all the config variables declared within each of the VIRGO kernel modules.

5. Config variables in each driver and system call have been named with prefix and suffix to differentiate the module and/or system call it serves.

6. In geographically distributed cloud virgo_client.conf has to be in client nodes and virgo_cloud.conf has to be in cloud nodes. For VIRGO Queue - KingCobra REQUEST-REPLY peer-to-peer messaging system same node can have virgo_client.conf and virgo_cloud.conf.

7. Above segregation largely simplifies the build process as each module and system call is independently built without need for a symbol to be exported from other module by pre-loading it.

8. VIRGO File system driver and system calls have been tested with above changes and the virgo_open(),virgo_read() and virgo_write() calls work with much less crashes and freeze problems compared to 3.7.8 (some crashes in VIRGO FS syscalls in 3.7.8 where already reported kernel bugs which seem to have been fixed in 3.15.5). Today's kern.log test logs have been committed to repository.

---
988. Committed as on 23 August 2014

Commenting use_as_kingcobra_service if clauses temporarily as disabling also doesnot work and only commenting the block
works for VIRGO syscall path. Quite weird as to how this relates to the problem. As this is a heisenbug further testing is
difficult and sufficient testing has been done with logs committed to repository. Probably a runtime symbol lookup for kingcobra
causes the freeze.
For forwarding messages to KingCobra and VIRGO queues, cpupooling driver is sufficient which also has the use_as_kingcobra_service clause.

---
989. Committed as on 23 August 2014 and 24 August 2014

As cpupooling driver has the same crash problem with kernel_accept() when KingCobra has benn enabled, KingCobra clauses have been commented in both cpupooling and memorypooling drivers. Instead queueing driver has been updated with a kernel service infrastructure to accept connections at port 60000. With this following paths are available for KingCobra requests:

VIRGO cpupooling or memorypooling ====> VIRGO Queue =====> KingCobra

(or)

VIRGO Queue kernel service ============================> KingCobra

---
990. Committed as on 26 August 2014

- all kmallocs have been made into GFP_ATOMIC instead of GFP_KERNEL
- moved some kingcobra related header code before kernel_recvmsg()
- some header file changes for set_fs()

This code has been tested with modified code for KingCobra and the standalone kernel service that accepts requests from telnet directly at port 60000, pushes to virgo_queue
and is handled to invoke KingCobra servicerequest kernelspace function, works (the kernel_recvmsg() crash was most probably due to Read-Only filesystem -errno printed is -30)

VIRGO version 14.9.9 has been release tagged on 9 September 2014

---
991. Committed as on 26 November 2014

New kernel module cloudsync has been added to repository under drivers/virgo
that can be used for synchronization(lock() and unlock()) necessities in VIRGO
cloud. Presently Bakery Algorithm has been implemented.

---
992. Committed as on 27 November 2014
---
virgo_bakery.h bakery_lock() has been modified to take 2 parameters - thread_id
and number of for loops (1 or 2)

---
993. Committed as on 2 December 2014

VIRGO bakery algorithm implementation has been rewritten with some bugfixes.
Sometimes there are soft lockup errors due to looping in kernel time durations
for which are kernel build configurable.

---
994. Committed as on 17 December 2014

Initial code commits for VIRGO EventNet kernel module service:

1.EventNet Kernel Service listens on port 20000

2.It receives eventnet log messages from VIRGO cloud nodes and writes the log
messages
after parsing into two text files /var/log/eventnet/EventNetEdges.txt and
/var/log/eventnet/EventNetVertices.txt by VFS calls

3.These text files can then be processed by the EventNet implementations in
AsFer (python pygraph and
C++ boost::graph based)

4.Two new directories virgo/utils and virgo/eventnet have been added.

5.virgo/eventnet has the new VIRGO EventNet kernel module service implementation
that listens on
port 20000.

6.virgo/utils is the new generic utilities driver that has a
virgo_eventnet_log()
exported function which connects to EventNet kernel service and sends the vertex
and edge eventnet
log messages which are parsed by kernel service and written to the two text
files above.

7.EventNet log messages have two formats:
   - Edge message - "eventnet_edgemsg#<id>#<from_event>#<to_event>"
   - Vertex message - "eventnet_vertextmsg#<id>-<partakers csv>-<partaker
conversations csv>"

8.The utilities driver Module.symvers have to be copied to any driver which are
then merged with the symbol files of the corresponding driver. Target clean has
to be commented while
building the unified Module.symvers because it erases symvers carried over
earlier.

9.virgo/utils driver can be populated with all necessary utility exported
functions that might be needed

in other VIRGO drivers.

10.Calls to virgo_eventnet_log() have to be #ifdef guarded as this is quite network intensive.

---
995. Commits as on 18 December 2014

Miscellaneous bugfixes,logs and screenshot

- virgo_cloudexec_eventnet.c - eventnet messages parser errors and eventnet_func bugs fixed
- virgo_cloud_eventnet_kernelspace.c - filp_open() args updated due to vfs_write() kernel panics. The vertexmessage vfs_write is done after looping through the vertice textfile and appending the conversation to the existing vertex.Some more code has to be added.
- VIRGO EventNet build script updated for copying Module.symvers from utils driver for merging with eventnet Module.symvers during Kbuild
- Other build generated sources and kernel objects
- new testlogs directory with screenshot for edgemsg sent to EventNet kernel service and kern.log with previous history for vfs_write() panics due to permissions and the logs for working filp_open() fixed version
- vertex message update

---
996. Commits as on 2,3,4 January 2015

- fixes for virgo eventnet vertex and edge message text file vfs_write() errors
- kern.logs and screenshots

VIRGO version 15.1.8 release tagged on 8 January 2015

---
772. (FEATURE) Commits as on 3 March 2015 - Initial commits for Kernel Analytics Module which reads the /etc/virgo_kernel_analytics.conf config (and) VIRGO memorypooling Key-Value Store Architecture Diagram - - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
---
- Architecture of Key-Value Store in memorypooling (virgo_malloc,virgo_get,virgo_set,virgo_free) has been uploaded as a diagram at http://sourceforge.net/p/virgo-linux/code-0/HEAD/tree/trunk/virgo-docs/VIRGOLinuxKernel_KeyValueStore_and_Modules_Interaction.jpg

- new kernel_analytics driver for AsFer <=> VIRGO+USBmd+KingCobra interface has been added.
- virgo_kernel_analytics.conf having csv(s) of key-value pairs of analytics variables is set by AsFer or any other Machine Learning code. With this VIRGO Linux Kernel is endowed with abilities to dynamically evolve than being just a platform for user code. Implications are huge - for example, a config variable "MaxNetworkBandwidth=255" set by the ML miner in userspace based on a Perceptron or Logistic Regression executed on network logs can be read by a kernel module that limits the network traffic to 255Mbps. Thus kernel dynamically changes behaviour.
- kernel_analytics Driver build script has been added

---
997. Commits as on 6 March 2015

- code has been added in VIRGO config module to import EXPORTed kernel_analytics config key-pair array
set by Apache Spark (mined from Uncomplicated Fire Wall logs) and manually and write to kern.log.

NeuronRain version 15.6.15 release tagged

---
998. Portability to linux kernel 4.0.5

The VIRGO kernel module drivers are based on kernel 3.15.5. With kernel 4.0.5 kernel which is the latest following
compilation and LD errors occur - this is on cloudfs VIRGO File System driver :
- msghdr has to be user_msghdr for iov and iov_len as there is a segregation of msghdr
- modules_install throws an error in scripts/Makefile.modinst while overwriting already installed module

---
999. Commits as on 9 July 2015

VIRGO cpupooling driver has been ported to linux kernel 4.0.5 with msghdr changes as mentioned previously
with kern.log for VIRGO cpupooling driver invoked in parameterIsExecutable=2 (kernel module invocation)
added in testlogs

1000. Commits as on 10,11 July 2015

VIRGO Kernel Modules:
- memorypooling
- cloudfs
- utils
- config
- kernel_analytics
- cloudsync
- eventnet
- queuing
along with cpupooling have been ported to Linux Kernel 4.0.5 - Makefile and header files have been
updated wherever required.

1001. Commits as on 20,21,22 July 2015

Due to SourceForge Storage Disaster(http://sourceforge.net/blog/sourceforge-infrastructure-and-service-restoration/),
the github replica of VIRGO is urgently updated with some important changes for msg_iter,iovec
etc., in 4.0.5 kernel port specifically for KingCobra and VIRGO Queueing. These have to be committed to SourceForge Krishna_iResearch
repository at http://sourceforge.net/users/ka_shrinivaasan once SourceForge repos are restored.
Time to move on to the manufacturing hub? GitHub ;-)

1002. VIRGO Queueing Kernel Module Linux Kernel 4.0.5 port:

- msg_iter is used instead of user_msghdr
- kvec changed to iovec
- Miscellaneous BUF_SIZE related changes

- kern.logs for these have been added to testlogs
- Module.symvers has been recreated with KingCobra Module.symvers from 4.0.5 KingCobra build
- clean target commented in build script as it wipes out Module.symvers
- updated .ko and .mod.c

1003. KingCobra Module Linux Kernel 4.0.5 port

- vfs_write() has a problem in 4.0.5
- the filp_open() args and flags which were working in 3.15.5 cause a kernel panic implicitly and nothing was written to logs
- It took a very long time to figure out the reason to be vfs_write and filp_open
- O_CREAT, O_RDWR and O_LARGEFILE cause the panic and only O_APPEND is working, but does not do vfs_write(). All other VIRGO Queue + KingCobra functionalities work viz., enqueueing, workqueue handler invocation, dequeueing, invoking kingcobra kernelspace service request function from VIRGO queue handler, timestamp, timestamp and IP parser, reply_to_publisher etc.,
- As mentioned in Greg Kroah Hartman's "Driving me nuts", persistence in Kernel space is a bad idea but still seems to be a necessary stuff - yet only vfs calls are used which have to be safe
- Thus KingCobra has to be in-memory only in 4.0.5 if vfs_write() doesn't work
- Intriguingly cloudfs filesystems primitives - virgo_cloud_open, virgo_cloud_read, virgo_cloud_write etc., work perfectly and append to a file.
- kern.logs for these have been added to testlogs
- Module.symvers has been recreated for 4.0.5
- updated .ko and .mod.c

Due to SourceForge outage and for a future code diversification NeuronRain codebases (AsFer, USBmd, VIRGO, KingCobra) in http://sourceforge.net/u/userid-769929/profile/ have been replicated in GitHub also - https://github.com/shrinivaasanka excluding some huge logs due to Large File Errors in GitHub.

1004. Commits as on 30 July 2015

VIRGO system calls have been ported to Linux Kernel 4.0.5 with commented gcc option -Wimplicit-function-declaration, msghdr and iovec changes similar to drivers mentioned in previous commit notes above. But Kernel 4.1.3 has some Makefile and build issues.
The NeuronRain codebases in SourceForge and GitHub would henceforth be mostly and always out-of-sync and not guaranteed to be replicas - might get diversified into different research and development directions (e.g one codebase might be more focussed on IoT while the other towards enterprise bigdata analytics integration with kernel and training which is yet to be designed- depend on lot of constraints)

1005. Commits as on 2,3 August 2015

- new .config file added which is created from menuconfig
- drivers/Kconfig has been updated with 4.0.5 drivers/Kconfig for trace event linker errors
Linux Kernel 4.0.5 - KConfig is drivers/ has been updated to resolve RAS driver trace event linker error. RAS was not included in KConfig earlier.
- link-vmlinux.sh has been replaced with 4.0.5 kernel version

---
1006. Commits as on 12 August 2015

VIRGO Linux Kernel 4.1.5 port - related code changes - some important notes:

- Linux Kernel 4.0.5 build suddenly had a serious root shell drop error in initramfs which was not resolved by:
      - adding rootdelay in grub
      - disabling uuid for block devices in grub config
        - mounting in read/write mode in recovery mode
        - no /dev/mapper related errors
        - repeated exits in root shell
      - delay before mount of root device in initrd scripts
- mysteriously there were some firmware microcode bundle executions in ieucodetool
- Above showed a serious grub corruption or /boot MBR bug or 4.0.5 VIRGO kernel build problem
- Linux 4.0.x kernels are EOL-ed
- Hence VIRGO is ported to 4.1.5 kernel released few days ago
- Only minimum files have been changed as in commit log for Makefiles and syscall table and headers and a build script has been added
for 4.1.5:
    Changed paths:
    A buildscript_4.1.5.sh
    M linux-kernel-extensions/Makefile
    M linux-kernel-extensions/arch/x86/syscalls/Makefile
    M linux-kernel-extensions/arch/x86/syscalls/syscall_32.tbl
    M linux-kernel-extensions/drivers/Makefile
    M linux-kernel-extensions/include/linux/syscalls.h

- Above minimum changes were enough to build an overlay-ed Linux Kernel with VIRGO codebase

---
1007. Commits as on 14,15,16 August 2015

Executed the minimum end-end telnet path primitives in Linux kernel 4.1.5 VIRGO code:
- cpu virtualization
- memory virtualization
- filesystem virtualization (updated filp_open flags)
and committed logs and screenshots for the above.

1008. Commits as on 17 August 2015

VIRGO queue driver:
- Rebuilt Module.symvers
- kern.log for telnet request to VIRGO Queue + KingCobra queueing system in kernelspace

---
1009. Commits as on 25,26 September 2015

VIRGO Linux Kernel 4.1.5 - memory system calls:

- updated testcases and added logs for syscalls invoked separately(malloc,set,get,free)
- The often observed unpredictable heisen kernel panics occur with 4.1.5 kernel too. The logs are 2.3G and
only grepped output is committed to repository.
- virgo_malloc.c has been updated with kstrdup() to copy the buf to iov.iov_base which was earlier

crashing in copy_from_iter() within tcp code. This problem did not happen in 3.15.5 kernel.
- But virgo_clone syscall code works without any changes to iov_base as above which does a strcpy()
 which is an internal memcpy() though. So what causes this crash in memory system calls alone
is a mystery.
- new insmod script has been added to load the VIRGO memory modules as necessary instead of at boot time.
- test_virgo_malloc.c and its Makefile has been updated.

VIRGO Linux Kernel 4.1.5 - filesystem calls- testcases and logs:

  - added insmod script for VIRGO filesystem drivers
  - test_virgo_filesystem.c has been updated for syscall numbers in 4.1.5 VIRGO kernel
  - virgo_fs.c syscalls code has been updated for iov.iov_base kstrdup() - without this there are kernel panics in copy_from_iter(). kern.log
testlogs have been added, but there are heisen kernel panics. The virgo syscalls are executed but not written to kern.log due to these crashes.
Thus execution logs are missing for VIRGO filesystem syscalls.

1010. Commits as on 28,29 September 2015

VIRGO Linux Kernel 4.1.5 filesystem syscalls:

- Rewrote iov_base code with a separate iovbuf set to iov_base and strcpy()-ing the syscall command to iov_base similar to VIRGO
memory syscalls
- Pleasantly the same iovbuf code that crashes in memory syscalls works for VIRGO FS without crash.Thus both virgo_clone and virgo_filesystem
syscalls work without issues in 4.1.5 and virgo_malloc() works erratically in 4.1.5 which remains as issue.
- kern.log for VIRGO FS syscalls and virgofstest text file written by virgo_write() have been added to repository

VIRGO Linux 4.1.5 kernel memory syscalls:

- rewrote the iov_base buffer code for all VIRGO memory syscalls by allocating separate iovbuf and copying the message to it - this just replicates the virgo_clone syscall behaviour which works without any crashes mysteriously.
- did extensive repetitive tests that were frequented by numerous kernel panics and crashes
- The stability of syscalls code with 3.15.5 kernel appears to be completely absent in 4.1.5
- The telnet path works relatively better though
- Difference between virgo_clone and virgo_malloc syscalls despite having same kernel sockets code looks like a non-trivial bug and a kernel stability issue.
- kernel OOPS traces are quite erratic.
- Makefile path in testcase has been updated

1011. Commits as on 4 October 2015

VIRGO Linux Kernel 4.1.5 - Memory System Calls:

- replaced copy_to_user() with a memcpy()
- updated the testcase with an example VUID hardcoded.
- str_to_addr2() is done on iov_base instead of buf which was causing NULL parsing
- kern.log with above resolutions and multiple VIRGO memory syscalls tests -

malloc,get,set
- With above VIRGO malloc and set syscalls work relatively causing less number of random kernel panics
- return values of memory calls set to 0
- in virgo_get() syscall, memcpy() of iov_base is done to data_out userspace pointer
- kern.log with working logs for syscalls - virgo_malloc(), virgo_set(), virgo_get() but still there are random kernel panics
- Abridged kern.log for VIRGO Memory System Calls with 4.1.5 Kernel - shows example logs for virgo_malloc(), virgo_set() and virgo_get()

1012. Commits as on 14 October 2015

VIRGO Queue Workqueue handler usermode clause has been updated with 4.1.5 kernel paths and kingcobra in user mode is enabled for invoking KingCobra Cloud Perfect Forwarding.

1013. Commits as on 15 October 2015

- Updated VIRGO Queue kernel binaries and build generated sources
- virgo_queue.h has been modified for call_usermodehelper() - set_ds() and fd_install() have been uncommented for output redirection. Output redirection works but there are "alloc_fd: slot 1 not NULL!" errors at random (kern.log in kingcobra testlogs) which seems to be a new feature in 4.1.5 kernel. This did not happen in 3.7.8-3.15.5 kernels

---
1014. Commits as on 3 November 2015

- kern.log for VIRGO kernel_analytics+config drivers which export the analytics variables from /etc/virgo_kernel_analytics.conf kernel-wide and print them in config driver has been added to config/testlogs

---
1015. Commits as on 10 January 2016

NeuronRain VIRGO enterprise version 2016.1.10 released.

NeuronRain - AsFer commits for VIRGO - C++ and C Python extensions
- Commits as on 29 January 2016

1016. (FEATURE - DONE) Python-C++-VIRGOKernel and Python-C-VIRGOKernel boost::python and cpython implementations:

- It is a known idiom that Linux Kernel and C++ are not compatible.
- In this commit an important feature to invoke VIRGO Linux Kernel from userspace python libraries via two alternatives have been added.
- In one alternative, C++ boost::python extensions have been added to encapsulate access to VIRGO memory system calls - virgo_malloc(), virgo_set(), virgo_get(), virgo_free(). Initial testing reveals that C++ and Kernel are not too incompatible and all the VIRGO memory system calls work well though initially there were some errors because of config issues.
- In the other alternative, C Python extensions have been added that replicate boost::python extensions above in C - C Python with Linux kernel works exceedingly well compared to boost::python.
- This functionality is required when there is a need to set kernel analytics

configuration variables learnt by AsFer Machine Learning Code
dynamically without re-reading /etc/virgo_kernel_analytics.conf.
- This completes a major integration step of NeuronRain suite - request travel roundtrip to-and-fro top level machine-learning C++/python
code and rock-bottom C linux kernel - bull tamed ;-).
- This kind of python access to device drivers is available for Graphics Drivers already on linux (GPIO - for accessing device states)
- logs for both C++ and C paths have been added in cpp_boost_python_extensions/ and cpython_extensions.
- top level python scripts to access VIRGO kernel system calls have been added in both directories:
        CPython - python cpython_extensions/asferpythonextensions.py
        C++ Boost::Python - python
cpp_boost_python_extensions/asferpythonextensions.py
- .so, .o files with build commandlines(asferpythonextensions.build.out) for "python setup.py build" have been added
in build lib and temp directories.
- main implementations for C++ and C are in
cpp_boost_python_extensions/asferpythonextensions.cpp and
cpython_extensions/asferpythonextensions.c

Commits as on 12 February 2016

1017. Commits for Telnet/System Call Interface to VIRGO CPUPooling -> VIRGO Queue -> KingCobra
---
*) This was commented earlier for the past few years due to a serious kernel panic in previous kernel versions - <= 3.15.5
*) In 4.1.5 a deadlock between VIRGO CPUPooling and VIRGO queue driver init was causing following error in "use_as_kingcobra_service" clause :
        - "gave up waiting for virgo_queue init, unknown symbol push_request()"
*) To address this a new boolean flag to selectively enable and disable VIRGO Queue kernel service mode "virgo_queue_reactor_service_mode" has been added.
*) With this flag VIRGO Queue is both a kernel service driver and a standalone exporter of function symbols - push_request/pop_request
*) Incoming request data from telnet/virgo_clone() system call into cpupooling kernel service reactor pattern (virgo cpupooling listener loop) is treated as generic string and handed over to VIRGO queue and KingCobra which publishes it.
*) This resolves a long standing deadlock above between VIRGO cpupooling "use_as_kingcobra_service" clause and VIRGO queue init.
*) This makes virgo_clone() systemcall/telnet both synchronous and asynchronous - requests from telnet client/virgo_clone() system call can be either synchronous RPC functions executed on a remote cloud node in kernelspace (or) an asynchronous invocation through "use_as_kingcobra_service" clause path to VIRGO Queue driver which enqueues the data in kernel workqueue and subsequently popped by KingCobra.
*) Above saves an additional code implementation for virgo_queue syscall paths - virgo_clone() handles, based on config selected, incoming data passed to it either as a remote procedure call or as a data that is pushed to VIRGO Queue/KingCobra pub-sub kernelspace.

Prerequisites:

- insmod kingcobra_main_kernelspace.ko
- insmod virgo_queue.ko compiled with flag virgo_queue_reactor_service_mode=1
      (when virgo_queue_reactor_service_mode=0, listens on port 60000 for direct telnet requests)
- insmod virgo_cloud_test_kernelspace.ko
- insmod virgo_cloudexec.ko (listens on port 10000)

Schematic Diagram

VIRGO clone system call/telnet client ---> VIRGO cpupooling(compiled with use_as_kingcobra_service=1) ------> VIRGO Queue kernel service (compiled with virgo_queue_reactor_service_mode=1) ---> Linux Workqueue handler ------> KingCobra

---
1018. Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports
---
- Imported Kernel Analytics variables into CloudFS kernel module - printed in driver init()
- Module.symvers from kernel_analytics has been merged with CloudFS Module.symvers
- Logs for above has been added in cloudfs/test_logs/
- Makefile updated with correct fs path
- Copyleft notices updated

1019. Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports
---
- Kernel Analytics driver exported variables have been imported in CPU virtualization driver
- Module.symvers from kernel_analytics has been merged with Module.symvers in cpupooling
- kern.log for this import added to cpupooling/virgocloudexec/test_logs/

1020. Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports
---
- Imported kernel analytics variables into memory virtualization driver init() , exported from kernel_analytics driver
- build shell script updated
- logs added to test_logs/
- Module.symvers from kernel_analytics has been merged with memory driver Module.symvers
- Makefile updated

1021. Commits as on 15 February 2016 - Kernel Analytics - VIRGO Linux Kernelwide imports
---
- Imported kernel analytics variables into VIRGO Queueing Driver
- logs for this added in test_logs/
- Makefile updated
- Module.symvers from kernel_analytics has been merged with Queueing driver's Module.symvers
- .ko, .o and build generated sources

Commits as on 16,17 February 2016

1022. (FEATURE-DONE) Socket Buffer Debug Utility Function - uses linux skbuff facility
---
- In this commit a multipurpose socket buffer debug utility function has been added in utils driver and exported kernelwide.
- It takes a socket as function argument does the following:
      - dereference the socket buffer head of skbuff per-socket transmit data queue
      - allocate skbuff with alloc_skb()
      - reserve head room with skb_reserve()
      - get a pointer to data payload with skb_put()
      - memcpy() an example const char* to skbuff data
      - Iterate through the linked list of skbuff queue in socket and print headroom and data pointers
      - This can be used as a packet sniffer anywhere within VIRGO linux network stack
- Any skb_*() functions can be plugged-in here as deemed necessary.
- kern.log(s) which print the socket internal skbuff data have been added to a new testlogs/ directory
- .cmd files generated by kbuild

1023. (FEATURE-DONE) Commits as on 24 February 2016
---
skbuff debug function in utils/ driver:
(*) Added an if clause to check NULLity of skbuff headroom before doing skb_alloc()
(*) kern.log for this commit has been added testlogs/
(*) Rebuilt kernel objects and sources

Commits as on 29 February 2016

---
771. (FEATURE-DONE) Software Analytics - SATURN Program Analysis added to VIRGO Linux kernel drivers - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
---
- SATURN (saturn.stanford.edu) Program Analysis and Verification software has been
integrated into VIRGO Kernel as a Verification+SoftwareAnalytics subsystem
- A sample driver that can invoke an exported function has been added in drivers
- saturn_program_analysis
- Detailed document for an example null pointer analysis usecase has been created in virgo-docs/VIRGO_SATURN_Program_Analysis_Integration.txt
-
linux-kernel-extensions/drivers/virgo/saturn_program_analysis/saturn_program_analysis_trees/error.txt is the error report from SATURN
- SATURN generated preproc and trees are in linux-kernel-extensions/drivers/virgo/saturn_program_analysis/preproc and linux-kernel-extensions/drivers/virgo/saturn_program_analysis/saturn_program_analysis_trees/

1024. Commits as on 10 March 2016

- SATURN analysis databases (.db) for locking, memory and CFG analysis.
- DOT and PNG files for locking, memory and CFG analysis.

- new folder saturn_calypso_files/ has been added in saturn_program_analysis/ with new .clp files virgosaturncfg.clp and virgosaturnmemory.clp
- SATURN alias analysis .db files

1025.(FEATURE-DONE) NEURONRAIN - ASFER Commits for VIRGO - CloudFS systems calls integrated into Boost::Python C++ and Python CAPI invocations

---

AsFer Commits as on 30 May 2016

VIRGO CloudFS system calls have been added (invoked by unique number from syscall_32.tbl) for C++ Boost::Python interface to
VIRGO Linux System Calls. Switch clause with a boolean flag has been introduced to select either VIRGO memory or filesystem calls.
kern.log and CloudFS textfile Logs for VIRGO memory and filesystem invocations from AsFer python have been committed to testlogs/

1026. AsFer Commits as on 31 May 2016

Python CAPI interface to NEURONRAIN VIRGO Linux System Calls has been updated to include File System open, read, write primitives also.
Rebuilt extension binaries, kern.logs and example appended text file have been committed to testlogs/. This is exactly similar to
commits done for Boost::Python C++ interface. Switch clause has been added to select memory or filesystem VIRGO syscalls.

(BUG - STABILITY ISSUES) Commits - 25 July 2016 - Static Analysis of VIRGO Linux kernel for investigating heisencrashes

---

Initial Documentation for Smatch and Coccinelle kernel static analyzers executed on VIRGO Linux kernel - to be updated
periodically with further analysis.

(BUG - STABILITY ISSUES) Commits - 1 August 2016 - VIRGO Linux Stability Issues - Ongoing Random oops and panics investigation

---

1. GFP_KERNEL has been replaced with GFP_ATOMIC flags in kmem allocations.
2. NULL checks have been introduced in lot of places involving strcpy, strcat, strcmp etc., to circumvent
buffer overflows.
3. Though this has stabilized the driver to some extent, still there are OOPS in unrelated places deep
with in kernel where paging datastructures are accessed - kmalloc somehow corrupts paging
4. OOPS are debugged via gdb as:
        4.1 gdb ./vmlinux /proc/kcore
        or
        4.2 gdb <loadable_kernel_module>.o
   followed by
        4.3 l *(address+offset in OOPS dump)

5. kern.log(s) for the above have been committed in tar.gz format and have numerous OOPS occurred during repetitive telnet and syscall invocation(boost::python C++) invocations of virgo memory system calls.
6. Paging related OOPS look like an offshoot of set_fs() encompassing the filp_open VFS calls.

(BUG-STABILITY ISSUES) Commits - 26 September 2016 - Ongoing Random Panic investigation
---
Further analysis on direct VIRGO memory cache primitives telnet invocation - problems are similar
to Boost::Python AsFer VIRGO system calls invocations.

(BUG-STABILITY ISSUES) Commits - 27 September 2016 - Ongoing Random Panic investigation
---
Analysis of VIRGO memory cache primitives reveal more inconsistencies in cacheline flushes between CPU and GPU.

1027. Commits - 20 March 2017 and 21 March 2017 - VIRGO Linux 64-bit build based on 4.10.3 kernel
---
*) moved virgoeventnetclient_driver_build.sh to virgoutils_driver_build.sh in utils/ driver
*) Updated VIRGO Linux Build Steps for 4.10.3
*) New repository has been created for 64-bit VIRGO Linux kernel based on 4.10.3 mainline kernel in GitHub and imported in SourceForge:
	https://github.com/shrinivaasanka/virgo64-linux-github-code
	https://sourceforge.net/p/virgo64-linux/
*) Though it could have been branched off from existing VIRGO repository (32-bit) which is based on 4.1.5 mainline kernel, creating a separate repository for 64-bit 4.10.3 VIRGO kernel code was simpler because:
	- there have been directory path changes for syscall entries in 4.10.3 and some other KBuild entities
	- Some script changes done for 4.1.5 in modpost and vmlinux phases are not required
	- having two VIRGO branches one with 4.1.5 code and 32-bit driver .ko binaries and other with 4.10.3 code and 64-bit driver .ko binaries could be unmanageable and commits could go into wrong branch
	- 4.10.3 64-bit VIRGO kernel build is still in experimental phase and it is not known if 64-bit 4.10.3 build solves earlier panic problems in 4.1.5
	- If necessary one of these two repositories could be made branch of the other later

---
1028. Commits - 27 March 2017 Ongoing analysis of VIRGO 64 bit linux kernel based on 4.10.3 kernel mainline
---
*) Prima facie, 64 bit kernel is quite finicky and importunate compared to 32 bit and 64 bit specific idiosyncrasies are to the fore.
*) During the past 1 week, quite a few variants of kernel and drivers builds were tried with KASAN enabled and without KASAN (Google Kernel Address

Sanitizer)
*) KASAN shows quite huge number of user memory accesses which later translate to panics.
*) Most nagging of these was kernel_recvmsg() panic.
*) Added and updated skbuff socket debug utility driver with a new debug function and to print more fields of skbuff
*) KASAN was complaining about _asan_load8 (loading 8 userspace bytes)
*) All erroneous return data types in VIRGO mempool ops structure have been corrected in VIRGO headers
*) all type casts have been sanitized
*) Changed all kernel stack allocations to kernel heap kzallocs
*) This later caused a crash in inet_sendmsg in kernel_sendmsg()
*) gdb64 disassemble showed a trapping instruction:
testb $0x6,0x91(%14) with corresponding source line:
sg = !!(sk->sk_route_caps & NETIF_F_SG)
in tcp_sendmsg() (net/ipv4/tcp.c)
*) changed kernel_sendmsg() to sock->ops->sendmsg()
*) These commits are still ongoing analysis only.
*) Screenshots for these have been added to debug-info/

1029. Continued analysis of VIRGO 64-bit linux kernel built on 4.10.3 mainline - Commits - 30 March 2017
---
*) Previous commit was crashing inside tcp_sendmsg()
*) GDB64 disassembly shows NULL values for register R12 which is added with an offset 91 and is an operand in testb
*) Protected all kernel_sendmsg() and kernel_recvmsg() in both system calls side and drivers side with
      oldfs=get_fs(), set_ds(KERNEL_DS) and set_fs(oldfs)
blocks without which there are random kernel_sendmsg and kernel_recvmsg hangs
*) Removed init_net and sock_create_kern usage everywhere and replaced them with sock_create calls
*) Tried MSG_FASTOPEN flags but it does not help much in resolving tcp_sendmsg() NULL pointer dereference issue. MSG_FASTOPEN just
speedsup the message delivery by piggybacking the message payload before complete handshake is established(SYN, SYN-ACK, SYN-ACK-ACK) in
SYN-ACK itself. But eventually it has to be enabled as fast open is becoming a standard.
*) Kasan reports have been enabled.
*) Added more debug code in skbuff debug utility functions in utils driver to check if sk->prot is a problem.
*) Replaced kernel_sendmsg with a sock->ops->sendmsg() in mempool sendto function which otherwise crashes in tcp_sendmsg().
*) With sock->ops->sendmsg() systemcalls <-----> drivers two-way request-reply works but still there are random -32 (broken pipe) and -104
(Connection Reset by Peer) errors
*) Logs for working sys_virgo_malloc() call with correctly returned VIRGO Unique ID for memory allocated has been committed to test_logs in
virgocloudexecmempool
*) sock->ops->sendmsg() in mempool driver sendto function requires a MSG_NOSIGNAL flag which prevents SIGPIPE signal though not fully
*) Reason for random broken pipe and connection reset by peer errors in mempool sendto is unknown. Both sides have connections open and
there is no noticeable traffic.
*) While socket communications in 32 bit VIRGO kernel syscalls and drivers work with no issues, why 64-bit has so many hurdles is puzzling.
Reasons could be 64 bit address alignment issues, 64 bit specific #ifdefs in kernel code flow, major changes from 4.1.5 to 4.10.3 kernels etc.,
*) NULL values for register R12 indicate already freed skbuff data which are accessed/double-freed. Kernel TCP engine has a circular linked list of skbuff write queue which is iterated in skbuff utils driver debug functions.

*) TCP engine clones the head data of skbuff queue, transmits it and waits for an ACK or timeout. Data is freed only if ACK or timeout occurs.
And head of the queue is advanced to next element in write queue and this continues till write queue is empty waiting for more messages.
*) If ACK is not received, head data is cloned again and retransmitted by sequence number flow control.

1030. Continued Analysis of VIRGO 64 bit based on 4.10.3 linux kernel - Commits - 1 April 2017, 3 April 2017
---
*) kernel_sendmsg() has been replaced with sock->ops->sendmsg() because kernel_sendmsg() is quite erratic in 4.10.3 64 bit
*) There were connection reset errors in system calls side for virgo_malloc/. This was probably because
sock->ops->sendmsg() requires MSG_DONTWAIT and MSG_NOSIGNAL flags and sendmsg does not block.
*) sock release happens and virgo_malloc syscalls receives -104 error
*) Temporarily sock_release has been commented. Rather socket timeout should be relied upon which should
do automatic release of socket resources
*) Similar flags have been applied in virgo_malloc syscalls too.
*) Logs with above changes do not have reset errors as earlier.
*) virgo set/get still crashes because 64 bit id is truncated which would require data type changes for
64 bit
*) test_virgo_malloc test case has been rebuilt with -m64 flag for invocation of 64 bit syscalls by
numeric ids

1031. Continued Analysis of VIRGO 64 bit 4.10.3 kernel - commits - 10 April 2017
---
*) There is something seriously wrong with 4.10.3 kernel sockets in 64 bit build VIRGO send/recv messages and even accept/listen too.
*) All kernel socket functionalities which work well in 4.1.5 32 bit VIRGO , have random hangs, panics in 4.10.3 VIRGO64 build mostly
in inet_recvmsg/sendmsg code path
*) KASAN shows attempts to access user address which occurs despite set_fs(KERNEL_DS)
*) Crash stack is similar to previous crashes in tcp_sendmsg()
*) Tried different address and protocol families for kernel socket accept (TCP,UDP,RAW sockets)
*) With Datagram sockets, kernel_listen() mysteriously fails with -95 error in kernel_bind(operation not supported)
*) With RAW sockets, kernel_listen() fails with -93 error for AF_PACKET (protocol not supported)
*) tcpdump pcap sniffer doesn't show anything unruly.
*) This could either be a problem with kernel build (unlikely), Kbuild .config or could have extraneous reasons. But .config for
4.1.5 and 4.10.3 are similar.
*) Only major difference between 4.10.3 and 4.1.5 is init_net added in sock_create_kern() internally
*) datatype of VIRGO Unique ID has been changed to unsigned long long (_u64)
*) tried with INADDR_LOOPBACK in place of INADDR_ANY
*) also tried with disabled multi(homing) in /etc/hosts.conf
*) Above random kernel socket hangs occur across all VIRGO system calls and drivers transport.
*) Utils kernel socket client to EventNet kernel service also has similar inet_recvmsg/inet_sendmsg panic problems.

1032. Commits - 11 April 2017 - EventNet and Utils Drivers 64bit
---
*) EventNet driver works in 64 bit VIRGO Linux
*) An example eventnet logging with utils virgo_eventnet_log() works now without tcp_sendmsg() related stalls in previous builds
) Return Datatypes for all EventNet operations have been sanitized (struct socket was returned as int in 32 build and reinterpret-cast to
struct socket*. This reinterpret cast does not work in 64 bit) in eventnet header.
*) utils eventnet log in init() has been updated with a meaningful edge update message
*) kern.log for this has been added to eventnet/testlogs

1033. Commits - 17 April 2017 - VIRGO64 Memory, CPU, FileSystem, EventNet kernel module drivers
---
*) telnet requests to VIRGO memory(kernelmemcache), cpu and filesystem modules work after resolving issues with return value types
*) commented le32_to_cpu() and print_buffer() which was suppressing lot of log messages.
*) VIRGO <driver> ops structures have been updated with correct datatypes.
) reinterpret cast of struct socket to int has been completely done away with which could have caused 64bit specific panics
*) lot of kern.log(s) and screen captures have been added for telnet requests in testlogs/ of respective <driver> directory
*) Prima facie 64bit telnet requests to VIRGO module listeners are relatively stabler than 32bit
*) Previous code changes should be relevant to 32 bit VIRGO kernel too.
*) tcp_sendmsg()/tcp_recvmsg() related hangs could be mostly related to corrupted skbuff queue within each socket.
*) This is because replacing kernel_<send/recv>msg() with sock_<send/recv>msg() causes return value to be 0 while
socket release crashes within skbuff related kernel functions.
*) To make socket state immutable, in VIRGO memory driver header files, client socket has been declared as const type.

1034. Commits - KingCobra 64 bit and VIRGO Queue + KingCobra telnet requests - 17 April 2017
---
*) Rebuilt KingCobra 64bit kernel module
*) telnet requests to VIRGO64 Queueing module listener driver are serviced by KingCobra servicerequest
*) Request_Reply queue persisted for this VIRGO Queue + KingCobra routing has been committed to c-src/testlogs.
*) kern.log for this routing has been committed in VIRGO64 queueing directory
) Similar to other drivers struct socket reinterpret cast to int has been removed and has been made const in queuesvc kernel thread

---
1035. Commits - VIRGO64 system calls - kernel module listeners - testcases and system calls updates - 18 April 2017
---

*) All testcases have been rebuilt
*) VIRGO kernel memcache,cpu and filesystem system calls have been updated with set_fs()/get_fs() blocks for kernel_sendmsg()
and kernel_recvmsg()
*) Of these virgo_clone() system call testcase (test_virgo_clone) works flawlessly and there are no tcp_sendmsg()/tcp_recvmsg() related
kernel panics.
*) VIRGO memcache and filesystem system call testcases have usual tcp_sendmsg()/tcp_recvmsg() despite the kernel socket code
being similar to VIRGO clone system call
*) Logs for VIRGO clone system call to CPU kernel driver module have been committed to virgo_clone/test/testlogs

---
1036. Commits - VIRGO64 Kernel MemCache and FileSystem system calls to VIRGO Memory and FileSystem Drivers - 19 April 2017
---
*) Changed iovec in virgo_clone.c to kvec
*) test_virgo_filesystem.c and test_virgo_malloc.c VIRGO system calls testcases have been changed with some additional printf(s) in userspace
*) virgo_malloc.c has been updated with BUF_SIZE in iov_len and memset to zero initialize the buffer. tcp_sendmsg()/tcp_recvmsg() pair were
getting stuck in copy_from_iter_full() memcpy with a NULL Dereference. memcpy() was reading past the buffer bound causing an overrun. strlen()
didnot work for iov_len.
*) virgo_fs.c virgo_write() memcpy has been changed back to copy_from_user() thereby restoring status quo ante (commented more than 3 years ago
because of a kernel panic in older versions of 32 bit VIRGO kernel)
*) Logs for VIRGO kmemcache and filesystem system calls have been committed to respective system call directories.
*) With this all VIRGO64 functionalities work in both telnet and system calls requests routes end-to-end from clients to kernel modules with
kernel sockets issues resolved fully.
*) Major findings are:
- VIRGO 4.10.3 64 bit kernel is very much stable compared to 32 bit 4.1.5 kernel
- there are no i915 related errors which happened in VIRGO 32 bit 4.1.5 kernel
- Repetitive telnet and system calls requests to VIRGO modules are stable and there are no kernel panics like 4.1.5 32 bit kernel.
- Google Kernel Address Sanitizer is quite helpful in finding stack overruns, null derefs, user memory accesses etc.,
- 64 bit kernel is visibly faster than 32 bit.
- Virgo Unique ID is now extended to 2^64 with unsigned long long.

1037. Commits - VIRGO64 memory and filesystem calls to memory and filesystem drivers requests routing - 20 April 2017
---
*) Changed return value of virgo_cloud_free_kernelspace() to a string literal "kernel memory freed"
*) Logs for VIRGO64 memory and filesystem calls to memory and filesystem drivers requests routing have been committed in test_logs of
both driver directories

1038. Commits - 27 April 2017

Residual logs for VIRGO 64 bit 4.10.3 kernel committed.

1039. Commits - 25 May 2017
---
*) Changed LOOPBACK to INADDR_ANY for VIRGO64 kernel memcache listen port
*) All VIRGO64 RPC, kernel memcache, cloud filesystem primitives have been retested
*) VIRGO64 mempool binaries have been rebuilt

1040. Commits - 31 August 2017 - NeuronRain ReadTheDocs Documentation - VIRGO64 System calls and Drivers (http://neuronrain-documentation.readthedocs.io/en/latest/)

(*) New directory systemcalls_drivers/ has been added to virgo-docs/ and representative VIRGO64
system calls and drivers functionality logs have been committed for demonstration purpose.
(*) VIRGO64 cloudfs driver has been rebuilt after changing virgofstest.txt file creation filp_open() call
(*) Screenshots and logs for VIRGO64 Clone, Kernel MemCache and Cloud FS SystemCalls-Drivers interaction, socket transport debug messages and Kernel Address Sanitizer have been committed in virgo-docs/systemcalls_drivers

---
1041. Commits - 23 September 2017 - Major VIRGO mainline kernel version Upgrade for Kernel Transport Layer Security - 4.10.3 to 4.13.3

(*) Recently released mainline kernel version 4.13 integrates SSL/TLS into kernelspace- KTLS - for the first time.
(*) KTLS is a standalone kernel module af_ktls (https://github.com/ktls) implemented by RedHat and Facebook for optimizing SSL handshake
within kernelspace itself and reduce userspace-kernelspace switches.
(*) sendfile() system call in linux which is used for file transmission (combining read+write) from one fd to another uses this
KTLS optimization in kernelspace in af_ktls codebase (af_ktls tool)
(*) VIRGO Linux kernel fork-off requires this kernelspace TLS functionality to fully secure traffic from system call client to remote
 cloud node's kernel module listeners
(*) Hence VIRGO64 linux kernel mainline base is urgently upgraded from 4.10.3 to 4.13.3
(*) All system calls and kernel module code in VIRGO64 now have #include(s) for tls.h and invoke kernel_setsockopt() on the client-server
kernelspace sockets for SOL_TLS and TLS_TX options and have been rebuilt.
(*) VIRGO64 RPC clone/kmemcache/cloudfs system calls to kernel module listeners have been tested with this new KTLS socket option,
on rebuilt VIRGO64 kernel overlay-ed on 4.13.3 64-bit linux kernel
(*) 4.13 mainline kernel also has SMB CIFS bug fixes for recent malware attacks (WannaCry etc.,) which further ensures security of
VIRGO64 linux fork-off kernelspace traffic.
(*) New buildscript for 4.13.3 linux kernel has been committed
(*) testlogs for VIRGO64 system calls and driver listeners KTLS transport have been committed in
virgo-docs/systemcalls_drivers/kern.log.VIRGO64_SystemCalls_Drivers.4.13.3_KTLS_kernelsockets.22September2017
(*) After this upgrade, complete system calls to driver listener traffic is SSL enabled implicitly.
(*) Updated kernel object files for 4.13.3 build are part of this commit.

```
--------------------------------------------------------------------------------
------------
1042. Commits - Remnant commits for 4.13.3 upgrade - 24 September 2017
--------------------------------------------------------------------------------
------------
Updated init.h and syscalls.h headers for virgo system calls

--------------------------------------------------------------------------------
------------------------------------------------------------
1043. Commits - VIRGO64 4.13.3 KTLS Upgrade - System Calls-Driver Listeners End-
to-End encrypted traffic testing - 25 September 2017
--------------------------------------------------------------------------------
------------------------------------------------------------
(*) VIRGO64 CPU/KMemCache/CloudFS system calls have been invoked by userspace
testcases and all primitives work after 4.13.3 KTLS upgrade
(*) Some small modifications to system calls code have been made and rebuilt to
remove redundant iovbuf variables in printk(s)
(*) test_virgo_filesystem.c testcase has been updated and rebuilt
(*) kern.log(s) for CPU/KMemCache/CloudFS systemcalls to driver listeners
invocations have been committed to respective system call
directories
(*) virgofstest.txt written to by virgo_write() has also been committed. But a
weird behaviour is still observed similar to previous linux kernel versions
(4.1.5 and 4.10.3): Repetitive invocations are required to flush the filesystem
buffer to force write the file.
(*) No DRM GEM i915 panics are observed and stability of VIRGO64 + 4.13.3 linux
kernel is more or equal to VIRGO64 + 4.10.3 linux kernel

--------------------------------------------------------------------------------
------------------------------------------------------------
1044. Commits - VIRGO64 VIRGO_KTLS branch creation and rebase of master to
previous commit - 30 September 2017
--------------------------------------------------------------------------------
------------------------------------------------------------
(*) New branch VIRGO_KTLS has been created after previous commit on 25 September
2017 and all 5 commits after 25 September 2017 till
28 September 2017 have been branched to VIRGO_KTLS (which has the #ifdef for
crypto_info, reads from /etc/virgo_ktls.conf and a new ktls
driver module)
(*) Following are the commit hashes and commandlines in GitHub and SourceForge:
               git branch -b VIRGO_KTLS
               git branch master
               git rebase -i <SHA1_on_25September2017>
               git rebase --continue
               git commit --amend
               git push --force
--------------------------------------------------------------------------------
-------------
1958  git checkout -b
1959  git checkout -b VIRGO_KTLS
1960  ls
1961  git checkout VIRGO_KTLS
1962  git push origin VIRGO_KTLS
1963  git status
1964  git checkout
1965  git checkout -b
1966  git branch
1967  git branch master
1968  git branch -h
1969  git branch
1970  git checkout master
1975  git checkout -b
1976  git checkout -b VIRGO_KTLS
1979  git push origin VIRGO_KTLS
```

```
1990  git rebase -i bb661e908cba2a5357414e89166f29086a28bdf0
1991  git status
1992  git rebase -i bb661e908cba2a5357414e89166f29086a28bdf0
1996  git rebase --continue
1997  git commit --amend
2019  git rebase -i bb661e908cba2a5357414e89166f29086a28bdf0
2029  git rebase --continue
2037  git push --force
2091  git rebase -i bb661e908cba2a5357414e89166f29086a28bdf0
2092  git rebase --continue
2093  git commit --amend
2094  git push --force
2110  git branch
2111  git branch master
2112  git checkout master
2113  git branch
2114  git rebase -i e76b4089633223f610fddc0e0eaff8c2cef8b9f1
2115  git commit --amend
2116  git rebase --continue
2117  git push --force
```

KTLS in 4.13.3 has support for only private symmetric encryption. It does not support Public Key Encryption yet. Since this might take a while mainstream VIRGO64 code might change a lot for other features. Therefore, VIRGO_KTLS specific crypto_info code has been branched off and would parallelly evolve if PKI features are available on KTLS in next versions of Linux kernel.

1045. Commits - 1 October 2017

---

kern.log(s) for VIRGO64 systemcalls-driver 4.13.3 64-bit upgrade tests on master branch after reversal and rebase of master HEAD yesterday for branching to VIRGO_KTLS. There is a weird General Protection Fault in intel atomic commit work not seen thus far. Also -32 and -107 socket errors are frequent after reversal though code remains same. But all virgo clone/kmemcache/ fs systemcalls functionalities work in invocations after GPF.

1046. Commits - VIRGO64 Utils and EventNet Drivers Update for tcp_sendmsg() stack out-of-bounds error - 3 October 2017

(*) Utils Generic Socket Client function virgo_eventnet_log() for EventNet kernel module listener was repeatedly failing in kernel_connect() emitting -32 and -107 errors.
(*) kernel_connect() was guarded by set_fs() and get_fs() memory segment routines to prevent any memory corruption. After this stack out-of-bounds error was reported by kernel address sanitizer in tcp_sendmsg() and copy_from_iter_full() implying the message buffer was not properly read within kernel.
(*) After replacing strlen(buf) by BUF_SIZE in msg flags before kernel_connect() stack out-of-bounds error has been remedied and Utils to EventNet virgo_eventnet_log() doesn't crash in tcp_sendmsg()
(*) kern.log for this has been committed in drivers/virgo/utils/testlogs/
(*) Both eventnet and utils drivers have been rebuilt

1047. VIRGO64 system calls-drivers on linux kernel 4.13.3 - miscellaneous bugfixes - 5 October 2017

```
--------------------------------------------------------------------------------
------------------------------------------------
(*) kernel_setsockopt() for KTLS has been commented in all system calls and
drivers because KTLS functionality has been branched to
VIRGO_KTLS
(*) In virgo_clone.c, iov.iov_len has been set to BUF_SIZE
(*) kernel_connect() has been guarded by set_fs()/get_ds() in VIRGO64 system
call clients
(*) test_virgo_malloc.c testcase has been updated
(*) There was a weird problem in in4_pton(): sin_addr.saddr was not set
correctly from string IP address and this was randomly occurring only in
virgo_set()
(*) in4_pton() is implemented in net/core/utils.c and reads the string IP
address digits and sums up the ASCII values to hex representation of the
address. Bitwise operations within this functions were failing randomly.
(*) Repeated builds were done trying different possible fixes but didn't work
e.g casting saddr to (u8*)
(*) There is an alternative in_aton() function which takes only String IP
address and returns address as __be32
(*) After in_aton() in virgo_set() random faulty address conversion does not
occur - in_aton() is differently implemented compared to in4_pton()
(*) msg_hdr has been initialized to NULL in virgo_set()
(*) Lot of debug printk()s have been added
(*) kern.log (.tar.gz) for RPC clone/KMemCache/Filesystem systemcalls-driver has
been committed to virgo-docs/systemcalls_drivers
(*) VIRGO Linux build steps have been updated for example commandlines to
overlay mainline kernel tree by VIRGO64 source

commit 4e6681ade4ddbf1bed17f7c115b59a5ebf884256
Author: K.Srinivasan <ka.shrinivaasan@gmail.com>
Date:   Fri Oct 6 11:36:15 2017 +0530

--------------------------------------------------------------------------------
------------------------------------------------
1048. VIRGO64 Queueing Kernel Module Listener - KingCobra64 - 4.13.3 - 6 October
2017
--------------------------------------------------------------------------------
------------------------------------------------
(*) telnet client connection to VIRGO64 Queue and a subsequent workqueue routing
(pub/sub) to KingCobra64 has been tested on 4.13.3
(*) TX_TLS socket option has not been disabled and is a no-op because it has no
effect on the socket.
(*) REQUEST_REPLY.queue for this routing from VIRGO64 queue and persisted by
KingCobra64 has been committed to KingCobra64 repositories in GitHub and
SourceForge

commit d4e95b58474838d65da9c69944c6287acbdfe72c
Author: K.Srinivasan <ka.shrinivaasan@gmail.com>
Date:   Fri Oct 6 11:05:21 2017 +0530

--------------------------------------------------------------------------------
------------------------------------------------
1049. VIRGO64 System Calls to Drivers and Telnet Client to Drivers on 4.13.3
linux kernel - master branch (after KTLS has been reverted
and branched to VIRGO_KTLS) - test case logs - 6 October 2017
--------------------------------------------------------------------------------
------------------------------------------------
(*) VIRGO64 System calls - Clone, KMemCache and Filesystem system call
primitives to Driver listeners invocations have been been tested
by respective test_<systemcall> unit testcases
(*) VIRGO64 Telnet Clients to Driver listeners invocations have been tested by
telnet connections
(*) Master branches in SourceForge and GitHub VIRGO64 do not have KTLS
provisions. Only VIRGO_KTLS branch has crypto_info and setsockopt()
```

for TX_TLS for kernel sockets.
(*) It has been already mentioned in NeuronRain Documentation in https://neuronrain-documentation.readthedocs.io/en/latest/ on securing VIRGO cloud nodes in the absence of KTLS - most obvious solution is to install VPN client-servers in all nodes which create Virtual IPs
on a secure tunnel (e.g OpenVPN).
(*) VIRGO64 system call clients and driver listeners should read these Virtual IPs from /etc/virgo_client.conf and /etc/virgo_cloud.conf
and cloud traffic is confined to the VPN tunnel.

1050. VIRGO64 SystemCalls-Drivers endtoend invocations unit case tests - on 4.13.3 - VIRGO64 main branch - 11 October 2017

(*) VIRGO64 systemcalls have been invoked from unit test cases (test_<system_call>) in a loop of few hundred iterations
(*) No DRM GEM i915 panics or random crashes are observed and stability is good
(*) This is probably the first loop iterative testing of VIRGO system calls and drivers.
(*) Kernel logs for this have been committed to virgo-docs/systemcalls_drivers directory.
(*) Note on concurrency: Presently mutexing within system calls have been commented because in past linux versions mutexing within kernel was causing strange panic issues. As a design choice and feature-stability tradeoff (stability is more important than introducing additional code) mutexing has been lifted up to userspace. It is upto the user applications invoking the system calls to synchronize multiple user threads invoking VIRGO64 system calls i.e VIRGO64 system calls are not re-entrant. This would allow just one kernel thread (mapped 1:1 to a user thread)
to execute in kernel space. Mostly this is relevant only to kmemcache system calls which have global in-kernel-memory address translation tables and next_id variable. VIRGO clone/filesystem calls do not have global in-kernel-memory datastructures.

1051. VIRGO64 SystemCalls-Drivers concurrent invocations - 2 processes having shared mutex - 14 October 2017

(*) VIRGO64 systemcalls are invoked in a function which is called from 2 processes concurrently
(*) Mutexes between the processes are PTHREAD_PROCESS_SHARED attribute set.
(*) test_virgo_malloc.c unit testcase has been enhanced to fork() a process and invoke systemcalls in a function for 100 iterations each
(*) Logs for the Virgo Unique IDs malloc/set/get/free in the systemcalls side and kern.logs for the drivers have been committed to test/testlogs/
(*) No DRM GEM i915 crashes were observed
(*) test_virgo_malloc.c testcase demonstrates the coarse grained critical section lock/unlock for kmemcache systemcalls and is a template
that should be followed for any userspace application.

---
775. (FEATURE) VIRGO64 Kernel Analytics - Streaming Implementation - 13 December 2017 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
---
(#) Presently kernel analytics config have to be read from a file storage. This

is a huge performance bottleneck when frequency of
analytics variables written to is realtime. For example autonomous
vehicles/drones write gigabytes of navigation data in few minutes.
(#) Because of this /etc/virgo_kernel_analytics.conf grows tremendously. File I/
O in linux kernel module is also fragile and not recommended.
(#) Previous latency limitations necessitate an alternative high performance
analytics config variable read algorithm
(#) This commit introduces new streaming kernel analytics config reading
function - It connects to a kernel analytics streaming IP address
on hardcoded port 64000 and reads analytics key-value pairs in an infinite loop.
(#) These read key-value pairs are stored in a kernel global ring buffer
exported symbol (by modulus operator). Because of circular buffer, older kernel
analytics variables are overwritten repetitively.
(#) kernel socket message flags are set to MSG_MORE | MSG_FASTOPEN |
MSG_NOSIGNAL for high response time. MSG_FASTOPEN works with no panic
in 4.13.3 64-bit which was a problem in previous kernel versions.
(#) kern.log for this has been committed to kernel_analytics/testlogs/
(#) include/linux/virgo_kernel_analytics.h header file has been updated for
declarations related to streaming analytics.
(#) Webserver used for this is netcat started on port 64000 as:

```
nc -l 64000
>k1=v1
>k2=v2
...
```


1052. VIRGO64 Kernel Analytics - Reading Stream of Analytic Variables made a
kernel thread - 13 December 2017

(#) This is sequel to previous commit for Stream reading Kernel Analytics
variables over a network socket
(#) read_streaming_virgo_kernel_analytics_conf() function is invoked in a
separate kernel thread because module init is blocked
(#) VIRGO64 config module was loaded and exported kernel analytics variables
read over socket by previous spin-off kernel thread are
imported in VIRGO64 config init.
(#) kern.log for this has been committed to testlogs/
(#) Pre-requisite: Webservice serving kernel_analytics variables must be started
before kernel_analytics module is loaded in kernel.
(#) By this a minimum facility for live reading analytics anywhere on cloud (it
can be userspace or kernelspace) and exporting them
to modules on a local cloud node kernel has been achieved - ideal for cloud-
analytics-driven IoT

1053. VIRGO64 System Calls - Drivers - Kernel Analytics Streaming - on 4.13.3
kernel - 15 December 2017

(#) VIRGO64 System Calls to Drivers invocations on 4.13.3 kernel have been
executed after enabling streaming kernel analytics
(#) VIRGO64 RPC/KMemCache/CloudFS Drivers import, streamed variable-value pairs
exported from kernel_analytics read from netcat webservice
(#) VIRGO64 KMemCache testcase has 2 concurrent processes invoking kememcache
systemcalls in a loop.
(#) kern.log for this has been committed to virgo-docs/systemcalls_drivers
(#) virgofstest.txt written by CloudFS systemcalls-drivers invocation is also
committed to virgo-docs/systemcalls_drivers

1054. VIRGO64 system calls and drivers - Quadcore 64bit - Known Issues - documentation only - 26 May 2019
---
1. VIRGO64 system calls to drivers interactions so far have been tested only on dual core 64 bit architecture.
2. In quad core 64 bit there have been random -32,-107, -101 errors in kernel_connect() from system call clients to driver services in almost all three types of VIRGO64 system calls - clone/kmemcache/filesystem - to respective drivers
3. These errors do not occur if following in4_pton() invocation is changed to in_aton() before kernel_connect() in system call clients and kernel is rebuilt with following change before kernel_connect():

```
       /*in4_pton(vaddr->hstprt->hostip, strlen(vaddr->hstprt->hostip),
(u8*)&sin.sin_addr.s_addr, '\0',NULL);*/
      sin.sin_addr.s_addr=in_aton(vaddr->hstprt->hostip);
```

4. Since this problem occurs erratically and only on quadcore 64-bit and reasons for these random -32,-101,-107 errors are still unknown, no commit for this code change has been made and this issue is left as documented known issue.
5. Most likely the u8* cast causes client socket address corruption.
6. Because of random -32,-101,-107 errors, in quadcore, system calls sometimes do not transmit client side commandline primitive strings to driver services.

---
773. (FEATURE) Linux Kernel 5.1.4, PXRC Drone/UAV/Flight Controller, UVC Video WebCam driver, Kernel Analytics, NeuronRain Usecases - 28 May 2019 - this section is an extended draft on respective topics in NeuronRain AstroInfer design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
---
1. Two analytics usecases mentioned in NeuronRainUsecases.txt in NeuronRain AsFer asfer-docs/ have been illustrated in this commit by 2 example drivers for PXRC Drone controller Driver and UVC Video WebCam Driver.
2. PXRC Phoenix RC flight controller is part of linux kernel from 4.17 and kernel major version has been bumped to 5.x.x recently.
3. Linux kernel 5.1.4 has been built by a new build script - buildscript_5.1.4.sh.
4. Linux kernel 5.1.4 has recent versions of PXRC drone controller driver and a UVC video webcam driver (http://www.ideasonboard.org/uvc/faq/)
5. New directory linux-kernel-5.1.4-extensions/ has been created for VIRGO64 code built on kernel mainline version 5.1.4. No branch is created for version 5.1.4 because pxrc driver is part of kernel only from 4.17 while code in linux-kernel-extensions/ is based on kernel 4.13.3 for dual core 64-bit architecture.
6. New VIRGO64 build on 5.1.4 kernel is necessary only for PXRC, UVC and kernel_analytics drivers while other VIRGO64 drivers have not been ported to 5.1.4 and are still on 4.13.3 kernel.
7. Two drivers for PXRC and UVC Webcam in 5.1.4 have been committed under linux-kernel-5.1.4-extensions/drivers/media/usb/uvc and linux-kernel-5.1.4-extensions/drivers/input/joystick
8. VIRGO64 kernel_analytics driver for 5.1.4 has been committed under linux-kernel-5.1.4-extensions/drivers/virgo/kernel_analytics.
9. Porting VIRGO64 kernel_analytics driver from 4.13.3 to 5.1.4 required changing vfs_read() of /etc/virgo_kernel_analytics.conf to kernel_read() in config file read.
10. Drivers code for PXRC is in drivers/input/joystick/pxrc.c has been instrumented with few printk() statements that print the virgo_kernel_analytics_conf array variable-value pairs exported by VIRGO64 kernel_analytics driver. Kernel analytics variables are

imported by #include of virgo_config.h
11. Drivers code for UVC Video WebCam is in drivers/media/usb/uvc. File drivers/media/usb/uvc/uvc_video.c has been instrumented with lot of uvc_trace() statements which print kernel_analytics driver exported analytics variable "match" and its boolean value. Kernel analytics variables are imported by #include of virgo_config.h. Variable "match" has been set to "True" assuming a face recognition or retinal scan match by userspace analytics and /etc/virgo_kernel_analytics.conf has been written to.
12. UVC Video WebCam traces are enabled by:
echo 0xffff > /sys/module/uvcvideo/parameters/trace
13. Example kern.log(s) for PXRC and UVC drivers are committed under drivers/input/joystick/testlogs/kern.log.pxrc.28May2019 and drivers/media/usb/uvc/kern.log.uvcvideo.28May2019 which show UVC traces printing the imported kernel analytics variable "match=True" and PXRC driver registration. No other PXRC traces are printed because of lack of drones and drone licensing dependency
14. Both UVC and PXRC drivers, VIRGO64 kernel_analytics driver and linux kernel 5.1.4 have been built on quadcore 64-bit architecture.
15. This example import of VIRGO64 kernel analytics variables into PXRC drone and UVC webcam drivers demonstrate passing of userspace analytics information to kernel for suitable action.
16. Driver build shell scripts have been committed to UVC and PXRC driver directories.

---
774. (FEATURE) Concurrent Managed Workqueue(CMWQ), VIRGO64 Queueing and KingCobra64 messaging - 12 June 2019 - this section is an extended draft on respective topics in NeuronRain AstroInfer design and KingCobra design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt,https://github.com/shrinivaasanka/kingcobra64-github-code/blob/master/KingCobraDesignNotes.txt
---
1. Existing workqueue underneath VIRGO64 queueing and requests routed by it to KingCobra64 messaging are old legacy workqueues which
have been revamped to Concurrent Managed Workqueue which supports concurrent messaging and lot of other options in queue creation.
2. create_workqueue() in VIRGO64 Queueing has been changed to alloc_workqueue() of Concurrent Managed Workqueue.
3. VIRGO64 Queueing request routing to KingCobra64 messaging has been tested with CMWQ and queueing log and kingcobra64 Request-Reply
Queue have been committed to respective testlogs of the drivers
4. reading from stream has been disabled in virgo_kernel_analytics.h
5. Reference - CMWQ documentation - https://www.kernel.org/doc/html/v4.11/core-api/workqueue.html
6. Byzantine Fault Tolerance in KingCobra64 persisted queue can be made available by performant CMWQ and routing to the Replicas of REQUEST_REPLY.queue by any of the practical BFT protocols available.
7. Most important application of CMWQ based VIRGO64-KingCobra64 is in the context of kernelspace hardware messaging in IoT,Drones and other analytics driven embedded systems.
8. An example usecase which is a mix of sync and async I/O in kernelspace:
(*) Analytics Variables computed by userspace machine learning are read over socket stream by kernel_analytics driver and
exported kernelwide
(*) Some interested Drone driver in kernel (example PXRC) reads the analytics variables synchronously and sends reply messages asynchronously to VIRGO64 Queuing driver over kernel sockets.
(*) VIRGO Queuing routes the queued messages to KingCobra64 driver

Srinivasan Kannan (alias) Ka.Shrinivaasan (alias) Shrinivas Kannan
http://sites.google.com/site/kuja27

```
--------------------------------------------------------------------------------
----------
KingCobra - A Research Software for Distributed Request Service on Cloud with
Arbiters

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

#------------------------------------------------------------------------------
------------------------
#Copyleft (Copyright+):
#Srinivasan Kannan
#(also known as: Ka.Shrinivaasan, Shrinivas Kannan)
#Ph: 9791499106, 9003082186
#Krishna iResearch Open Source Products Profiles:
#http://sourceforge.net/users/ka_shrinivaasan,
#https://github.com/shrinivaasanka,
#https://www.openhub.net/accounts/ka_shrinivaasan
#Personal website(research): https://sites.google.com/site/kuja27/
#emails: ka.shrinivaasan@gmail.com, shrinivas.kannan@gmail.com,
#kashrinivaasan@live.com
#------------------------------------------------------------------------------
------------------------
*******************************************************************************
*********/

Theoretical Interludes, Design Notes and ToDo (long term with no deadline)
--------------------------------------------------------------------------

[This is a major research oriented subsystem of NeuronRain and inspired by COBRA
project (done by the author in his BE (1995-1999) along with few other esteemed
classmates.
http://sourceforge.net/projects/acadpdrafts/files/Excerpts_Of_PSG_BE_FinalProjec
t_COBRA_done_in_1999.pdf/download.)]

[KingCobra though a misnomer is expanded as ClOud With ARBiters MimicKING
containing the anagram]

1. (THEORY) There is a cloud of nodes which execute a set of services from
randomly created clients.

2. (THEORY) This cloud could be on iCloud (AsFer+USBmd+VIRGO) platform or any
other opensource cloud platforms like Hadoop Cluster.

3. (THEORY) The Clients are publishers of Service requests which are of many
types -  miscellaneous types of Service that could be dynamically added through
other kernel modules and invoked through a switch-case or embedded in function
itself. Identified by unique id(s) for different types of services (for example
Problem reports, Suggestions etc.,)

4. (THEORY) The Services on the Cloud are Subscribers to these requests of
```

specific type. Thus this is the conventional publisher-subscriber model.

5. (THEORY) The requests flow through cloud using a workqueue (which could be a lowlevel Linux workqueue or VIRGO queue or some other queuing middleware software like ActiveMQ). The publishers enqueue and Subscribers dequeue the requests.

6. (THEORY) The difference is that the Cloud has nodes that "deceive" or "corrupt".

7. (THEORY) Service requests - are published by the clients in the need of a service which could be defined by markup file. These requests are scheduled and routed by the middleware to competent authority which services it (with or without timeframe) and replies to the client.

8. (THEORY) Problem reports - are published by clients which are "dissatisfied" by the quality of service by the cloud. These are analyzed by "arbiters" in the cloud which find the faulting node(s) and take action. This allows manual intervention but minimizes it.

9. (THEORY) Suggestions - are enhancement requests sent by clients and require manual intervention.

10. (THEORY) Cloud nodes have a Quality of Service metric calculated by a model.

11. (THEORY) The cloud has a reporting structure of nodes - either as a graph or tree. The graph is dynamically reorganized by weighting the Quality of Service of each node.

12. (THEORY) The difficult part of the above is using Arbiters to find "faulty" nodes based on problem reports from clients.

13. (THEORY) Brewer's CAP conjecture proved by [GilbertLynch] as a theorem (still debated) states that only 2 of the 3 (Consistency of data, Avaliability of data and Partition tolerance when some nodes or messages are lost) can be guaranteed and not all 3 are simultaneously achievable.

14. (THEORY) CAP theorem does not seem to apply to the above faulty scenario with corrupt nodes under Consistency or Availablity or Partition Tolerance. This is because a corrupt node can have any 2 of the 3 - it can give consistent data, is available with success response or can make the cloud work with missing data in partition tolerance but yet can "corrupt" the cloud. Probably this needs to be defined as a new attribute called Integrity.

15. (THEORY) As "corruption" is more conspicuous with monetary element, if above services are "charged" with a logical currency (e.g. bitcoin), then corruption in cloud is defineable approximately as (but not limited to)- "Undue favour or harm meted out to a client not commensurate with the charge for the service (or) unreasonable extra logical currency demanded to execute the service of same quality (or) deliberate obstruction of justice to a client with malevolent and unholy collusion with other cloud nodes with feigned CAP".

16. (THEORY) Identifying criminal nodes as in (15) above seems to be beyond the ambit of CAP. Thus CAP with Integrity further places a theoretical limit on "pure" cloud. If Integrity is viewed as a Byzantine problem with faulty or corrupt processes in a distributed system, and if resilience factor is rf (expected number of faulty nodes), then most algorithms can ensure a "working" cloud only if resilience is ~30% or less (3*rf+1) of the total number of cloud nodes. Probably this could apply to Integrity also that places a limit of 30% on "corrupt nodes" for the Cloud to work with sanity. Translating this to a Governance problem, a corruption-free administration is achievable with a maximum limit of 30% "corrupt" elements.

17. (THEORY-ONGOING) Analytics on the Problem reports sent to the cloud queue

give a pattern of corrupt nodes. Intrinsic Merit ranking with Citation graph maxflow considering cloud as a flow network where a node positively or negatively cites or "opines" about a node, as mentioned in http://arxiv.org/abs/1006.4458 (author's Master's thesis) and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf(published by the author during PhD) give a p2p ranking of cloud nodes that can be used for analysis though may not be reliable fully. AsFer has bigdata analytics functionality that fits well to this point to analyse the problem reports with machine learning algorithms and set the key-value pairs that are read by VIRGO kernel_analytics module and exported kernelwide. The persisted REQUEST_REPLY.queue with the logged request-reply IPs and timestamps can be mined with AsFer bigdata capability (e.g. Spark)

18. (THEORY) Policing the cloud nodes with arbiters - This seems to be limited by CAP theorem and Integrity as above. Also this is reducible to perfect inference problem in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt and drafts in https://sites.google.com/site/kuja27/

19. (THEORY) Brooks-Iyengar algorithm for sensors in all cloud nodes is an improved Byazantine Fault Tolerant algorithm.

20. (THEORY) BitCoin is a Byzantine Fault Tolerant protocol.

21. (THEORY) Byzantine Fault Tolerance in Clouds is described in http://www.computer.org/csdl/proceedings/cloud/2011/4460/00/4460a444-abs.html, http://www.eurecom.fr/~vukolic/ByzantineEmpire.pdf which is more on Cloud of Clouds - Intercloud with cloud nodes that have malicious or corrupt software. Most of the key-value(get/set) implementations do not have byzantine nodes (for example CAP without Byzantine nodes in Amazon Dynamo: http://www.eurecom.fr/~michiard/teaching/slides/clouds/cap-dynamo.pdf)

22. (THEORY) Related to point 18 - The problem of fact finding or fault finding using a cloud police has the same limitation as the "perfect inference" described in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt. "Money trail" involving the suspect node in point 28 is important to conclude something about the corruption. In real world tracking money trail is a daunting task. In cloud that abides by CAP, missing messages on trail can prevent reaching a conclusion thereby creating benefit-of-doubt. Also fixing exact value for a transaction that involves MAC currency message is undecidable - a normative economics problem can never be solved by exact theoretical computer science.

23. (THEORY) Reference article on cloud BFT for Byzantine, Corrupt brokers - Byzantine Fault-Tolerant Publish/Subscribe: A Cloud Computing Infrastructure (www.ux.uis.no/~meling/papers/2012-bftps-srdsw.pdf)

---

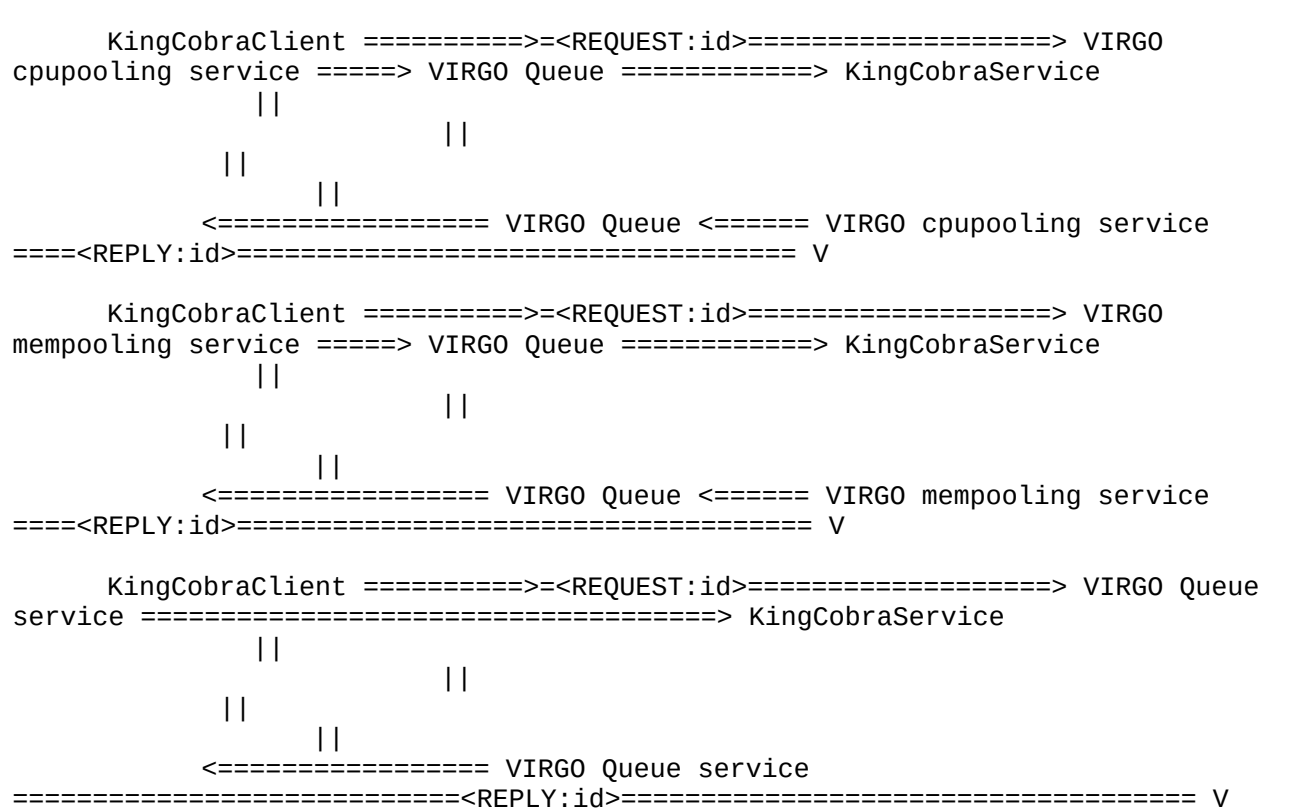
24. KingCobra messaging request-response design - options

24a. Implementing a message subscription model in kernelspace where clients publish the message that is queued-in to subscribers' queue (Topic like implementation - use of ActiveMQ C implementation if available).

24b. (DONE-minimal implementation) At present a minimum kernelspace messaging system that queues remote request and handles through workqueue handler is in place. This responds to the client once the Kingcobra servicerequest function finishes processing the request(reply_to_publisher() in KingCobra driver). Unlike the usual messaging server, in which client publishes messages of a particular type that are listened to by interested clients, one option is to continue the status-quo of KingCobra as a peer-to-peer messaging system. Thus every VIRGO node is both a kernelspace messaging client and server that can both

publish and listen. Every message in the cloud can have a universally unique id assigned by a timestamp server - https://tools.ietf.org/html/rfc4122 (similar to bitcoin protocol) so that each message floating in the cloud is unique across the cloud (or) no two messages on the VIRGO cloud are same. The recipient node executing kingcobra_servicerequest_kernelspace() parses the unique-id (example naive unique-id is <ip-address:port>#localtimestampofmachine which is a simplified version of RFC4122) from the incoming remote request and responds to the remote client through kernel socket connection that gets queued-in the remote client and handled similar to incoming remote request. To differentiate request and response-for-request response messages are padded with a string "REPLY:<unique-id-of-message>" and requests are padded with "REQUEST:<unique-id-of-message>". This is more or less similar to TCP flow-control with SEQ numbers but state-less like UDP. Simple analogy is post-office protocol with reference numbers for each mail and its reply. Thus there are chronologically two queues: (1) queue at the remote VIRGO cloud service node for request (2) queue at the remote client for response to the request sent in (1). Thus any cloudnode can have two types of messages - REQUEST and REPLY. Following schematic diagram has been implemented so far.

24c.(DONE) KingCobra - VIRGO queue - VIRGO cpupooling , mempooling and queue service drivers interaction schematic diagram:



24d. (ONGOING) kingcobra_servicerequest_kernelspace() distinguishes the "REQUEST" and "REPLY" and optionally persists them to corresponding on-disk filesystem. Thus a disk persistence for the queued messages can either be implemented in 1) VIRGO queue driver 2) workqueue.c (kernel itself needs a rewrite (or) 3) KingCobra driver. Option (2) is difficult in the sense that it could impact the kernel as-a-whole whereas 1) and 3) are modularized. At present Option 3 persistence within KingCobra driver has been implemented.

24e.Above option 24b implements a simple p2p queue messaging in kernel. To get a

Topic-like behaviour in VIRGO queue might be difficult as queue_work() kernel function has to be repeatedly invoked for the same work_struct on multiple queues which are subscribers of that message in AMQ protocol. Moreover creating a queue at runtime on need basis looks difficult in kernel which is usually done through some CLI or GUI interface in ActiveMQ and other messaging servers.

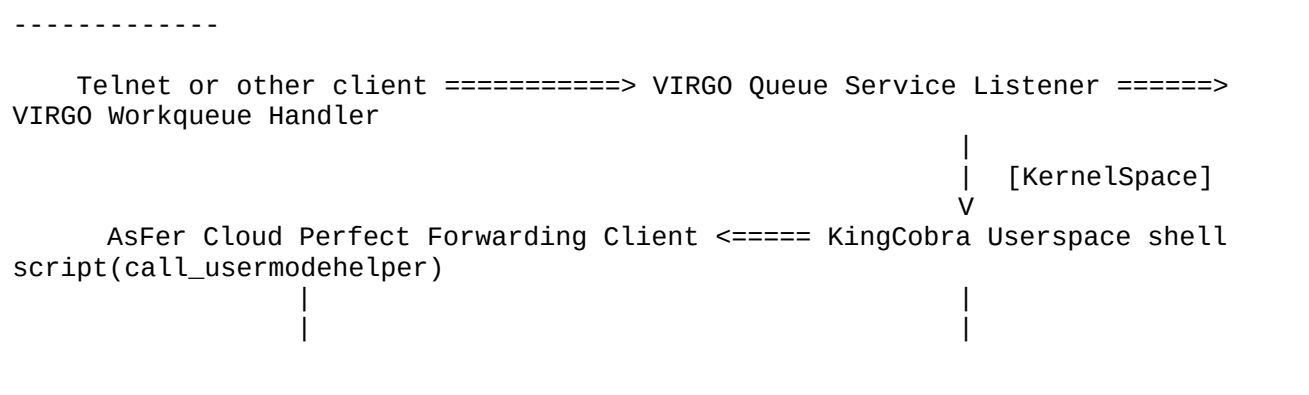
25. (ONGOING) For the timestamp service, EventNet described in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt is a likely implementation choice. AsFer already has a primitive text files based EventNet graph implementation in place. Periodic topological sort (quite expensive) of EventNet gives logical ordering and thus a logical timestamp of the cloud events.

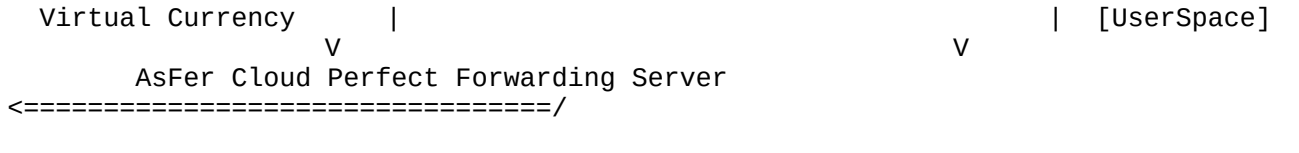
26. (THEORY - ONGOING Implementation) MESSAGE-AS-CURRENCY PROTOCOL: If each message payload is also construed as a currency carrier, each message id can be mapped to a unique currency or a coin with fixed denomination. This is similar to each currency note having a serial number as unique id. Uniqueness is guaranteed since there can be only one message (or coin) with that id on the cloud. This simulates a scenaio - "Sender of the message pays the Receiver with a coin having the unique id and Receiver acknowledges receipt". This is an alternative to BitCoin protocol. Double spending is also prohibited since at any point in time the message or "coin" with unique id can be sent by only one node in the cloud. Unique Cloudwide Timestamp server mimicks the functionality of "Mint". There is a difference here between conventional send-receive of messages - Once a message is sent to remote cloud node, no copy of it should exist anywhere in the cloud. That is, every MAC currency message is a cloudwide singleton. In pseudocode this is expressible as:

```
      m1=MAC_alloc(denomination)
      m2=m1 (---- this is disallowed)
```

Linux kernel allocation functions - kmalloc() - have a krefs functionality for reference counting within kernel. Refcount for MAC message can never exceed 1 across cloud for above singleton functionality - this has to be a clause everywhere for any unique MAC id. This requires a cloudwide krefs rather. Buyer decrements cloudwide kref and Seller increments it. In C++ this is done by std::move() and often required in "Perfect Forwarding" - http://thbecker.net/articles/rvalue_references/section_07.html - within single addressspace. By overloading operator=() with Type&& rvalue reference, the necessary networking code can be invoked that does the move which might include serialization. But unfortunately C++ and Linux kernel are not compatible. The Currency object has to be language neutral and thus Google Protocol Buffers which have C,C++,Java, Python .proto files compilers support might be useful but yet the move semantics in Kernel/C is non-trivial that requires cloudwide transactional kernel memory as mentioned in (31) below. A C++ standalone userspace client-server cloud object move implementation based on std::move() over network of Protocol Buffer Currency Objects has been added to AsFer repository at - http://sourceforge.net/p/asfer/code/HEAD/tree/cpp-src/cloud_move which can optionally be upcall-ed to userspace from VIRGO and KingCobra drivers. This C++ implementation is invoked in userspace with call_usermodehelper() from VIRGO Queue Messaging via the kernel workqueue handler.

26.1 Schematic Diagram for Cloud Perfect Forwarding with AsFer+VIRGOQueue+KingCobraUserspace:



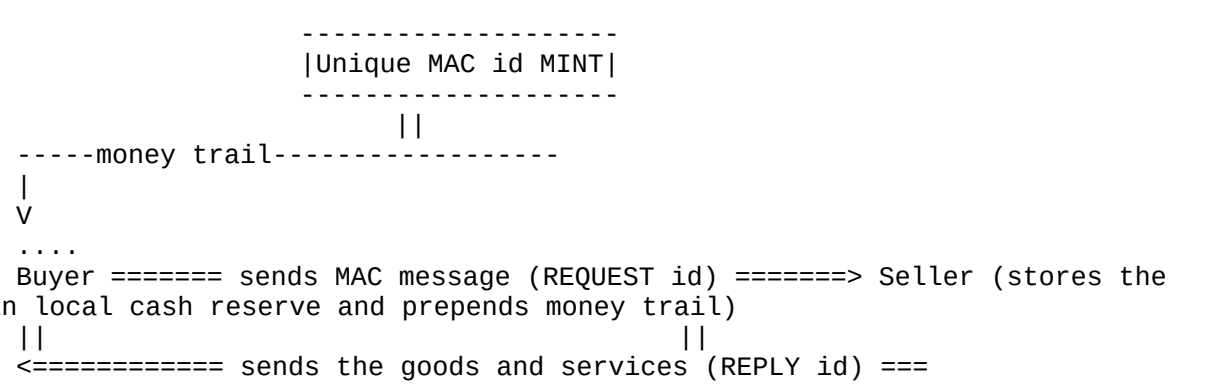


References:

26.2 An example distributed transactional memory implementation in cloud - http://infinispan.org/tutorials/simple/tx/ and http://www.cloudtm.eu/ - these are in userspace cloud (C++ and Java) and may not have cloud move functionality - move has to be simulated in a transaction: replicate data, delete in one endpoint and create in other endpoint

27. (THEORY) SIMULATING A VIRTUAL ECONOMY with above MAC protocol (Message-as-currency): If each message sent is considered as "money" element and cloud nodes and clients are the consumers and producers of "electronic money", the timestamp "Mint" becomes a virtual Federal Reserve or Central Bank that controls the "electronic money" circulation in the cloud. Infact any REPLY messages could be mapped to a Service a client derives by s(p)ending the REQUEST "money message". Thus value(REQUEST) should equal value(REPLY) where value() is a function that measures the value of a money denomination and the value of goods and/or services for that money. For example Rs.10000 or $10000 has no meaning if it doesn't translate into a value (analogy: erstwhile Gold Standard).When the value() function gets skewed phenomena like Inflation arise. Thus above model could also have a notion of value() and "electronic money inflation". Thus any "message money" with a unique id assigned by the cloud unique id(or logical timestamp) server can exist at most in only one node in the cloud. Money trail can be implemented by prefixing a header to the incoming message money in each cloud node that receives the money which traces the "path" taken. Cloud has to implement some Byzantine Fault Tolerant protocol. The value() function to some extent can measure the "deceit" as above. When a Buyer and Seller's value() functions are at loggerheads then that is starting point of "cloud corruption" at either side and might be an undecidable problem.

28. (THEORY) TRADING WITH ABOVE KINGCOBRA MAC protocol - somewhat oversimplified:



In the above schematic, money with unique id in cloud reaches a buyer after many buyer-seller transitions called "money trail". The MAC currency is prefixed by each node to create a chain. Buyer then sends a request to the seller through MAC virtual currency and seller replies with goods and services. Seller prepends the money trail chain. When a transaction occurs the whole cloud need not be notified about it except only buyer and seller. MAC Mint could create a bulk of money denominations and circulate them in cloud economy.

References:

28.1 Price fixing for items in Buyer-Seller-Trader networks - Trading Networks - Market Equilibrium and Walrasian Model of Price fixing - http://www.cs.cornell.edu/~eva/traders.pdf

28.2 Algoithmic Game Theory - Market Equilibrium for Price - Equilibrium is a strategic standoff - both players can't better their own present by changing strategies e.g Buyer-Sellers are market players and equilibrium price is the one where both buyer and seller can't gain by varying it - http://www.cis.upenn.edu/~mkearns/nips02tutorial/nips.pdf. Buyer-Seller payoff matrix picturises the bargaining problem.
28.3 Price-setting in Trading networks - Chapter 11 - https://www.cs.cornell.edu/home/kleinber/networks-book/networks-book.pdf

29. (THEORY) VALUE FOR ELECTRONIC MONEY: How is above MAC money earned - This again requires linking value to money (as money is not a value by itself and only a pointer to valuable item or resource). Thus any buyer can "earn" MAC money by something similar to a barter.

30. (THEORY and IMPLEMENTATION) FIXING VALUE FOR MAC MONEY: To delineate corruption as discussed in 27 above with value() disparity between MAC money and REPLY goods and services, an arbiter node in cloud has to "judge" the value of MAC sent by buyer and goods and services from seller and proclaim if corruption exists. Thus value() function itself has to be some kind of machine learning algorithm. This is related to or same as points 12 to 23 above. For example, while buying an item for few million bucks, value() has to take as input the description of the item and calculate the value "ideally" which is difficult. Because there are no perfect references to evaluate and only a weighted average of available market price range has to be taken as a reference which is error-prone. value() function can be recursively defined as("Reductionism"):

```
      ------------------------------------------------------------------------------------------------------------
      |value(i) = summation(value(ingredients of i)) + cost(integrating the ingredients to create item i) |
      ------------------------------------------------------------------------------------------------------------
```

Obviously the above recursion combinatorially explodes into exponential number of nodes in the recursion tree. Ideally recursion has to go deep upto quarks and leptons that makeup the standard model. If for practical purposes, recursion depth is restricted to t then size of value() tree is O(m^t) where m is average number of ingredients per component. Hence any algorithm computing the value() recursion has to be exponential in time. Computation of value() in the leaf nodes of the recursion is most crucial as they percolate bottom-up. If leaf nodes of all possible items are same (like quarks and leptons making up universe) then such atomic ingredient has to have "same" value for all items. Only the integration cost varies in the levels of the tree.For infinite case, value() function is conjectured to be undecidable - probably invoking some halting problem reduction .But above value() function could be Fixed Parameter Tractable in parameter recursion depth - t but yet could only be an approximation. A Turing machine computing value() function exactly might loop forever and thus Recursively Enumerable and not Recursive. A CVXPY implementation for Pricing Market Equilibrium has been implemented in KingCobra.

31. (THEORY) Buyer-Seller and MAC electronic money transaction schematic:

```
      Buyer        A-------<id><refcnt:0>----------------------> Seller <id><refcnt:1> (increments refcnt)
      (<id><refcnt:1>  |
       <id><refcnt:0>  |
      after decrement  |
      refcnt           |
      )----------------->
```

Above has to be transactional (i.e atomic across cloud nodes)

32. (THEORY) MAC protocol reaper

Reaper thread in each cloud node harvests the zero refcounted allocations and invokes destructors on them. Same MAC id cannot have kref count of 1 or above

in more than one cloud node due to the transaction mentioned previously.

33. (THEORY) Cloud Policing With Arbiters - Revisited:

When a suspect node is analyzed when a complaint problem is filed on it, (1) it is of foremost importance on how flawless is the arbiter who investigates on that and is there a perfect way to choose a perfect arbiter. In the absence of the previous credibility of entire cloud judiciary is blown to smithereens and falls apart. (2) Assuming a perfect arbiter which is questionable, next thing is to analyze the credibility of the node who sulked. This is nothing but the Citation problem in http://arxiv.org/abs/1106.4102 and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf where a node can positively or negatively cite another node a "Societal Norm" which can be faulted and citations/opinions could be concocted with malafide intent(perjury). This is rather a generalization of PageRank algorithm with negative citations. Thus Perfect Cloud Arbitration could be an unsolvable problem. (3) Even if both arbiter and complainant are perfect which is again questionable, there are still loopholes - lack of evidences or implicated witnesses might portray a negative impression of a positive node. Thus there are 3 tiers of weakenings in cloud arbitration and there could be more. P(Good) series in https://sites.google.com/site/kuja27/ precisely addresses this problem.

34. (THEORY) MAC Money Flow as MaxFlow problem:

Transactions happening in a cloud are edges between the nodes involved (buyer and seller). Thus it creates a huge directed graph. Flow of money in this graph can be modelled as Flow network. Minimum Cut of this graph shows crucial nodes in the graph which play vital role in cloud economy removal of which paralyzes the cloud (could be Central bank and other financial institutions). This graph has bidirectional edges where one direction is for money and the opposite direction is for Goods and Services. In the Flow network sum of flows is zero. But in the Money Flow Network each node is having a cash reserve ratio (CRR) due to commercial transactions which is confidential and privy to that node only and thus sum of flows can not be zero. Hub nodes in the Money Flow graph which can be obtained by getting k-core or D-core of the graph by some graph peeling algorithms are crucial nodes to the economy that contribute to Money circulation.

35. (THEORY) Cycles and components in above MAC Money Flow Graph:

Above graph of money transactions could be cyclic which implies a supply chain. Strongly connected components of this graph are most related nodes that are in same industry.

36. (THEORY) STOCK TRADING:

One of the component in above MAC Money Flow Graph of cloud could be a virtual Stock Exchange. Based on the financial and securities transactions of constituent organizations in the graph, index of the exchange varies.

37. (THEORY) Analysis of Poverty and Alleviation through above money flow graph:

Weights of the edges of money flow graph are the denominations of the transaction. Thus high value edges and low value edges divide the Graph logically into Rich and Poor strata(Bourgeoisie and Proletariat subgraphs). Equitable graph is the one which does not have too much of value difference between Rich and Poor sets of edges - a utopian to achieve. Mathematically, it is an optimization LP problem that seeks to minimize sum(RichEdges)-sum(PoorEdges) or Sum(RichVertices) - sum(PoorVertices) - without harming either - to be precise. This requires money flow to be programmed to find a feasible solution to this LP subject to constraints like work-pay parity etc.,(there could be more variables and constraints to this LP) . Due to CRR above Vertices also can be Rich and Poor in addition to Rich Edges and Poor Edges.

38.(THEORY) Demand and Supply and Value() function:

Alternative to the recursive definition of value() function above can be done through Demand and Supply - more the demand and less the supply, price increases and vice-versa. This is quite subjective compared to absolute recursive definition above. To simulate demand and supply, the weights of the money edges (-> direction) in the bidirectional graph change and fluctuate dynamically over time for unchanging weights of the Goods and Services edges (<- direction) between any pair of Buyer-Seller vertices. This makes Money and G&S Flow graph a Dynamic Graph with edge weight update primitive.

39.(THEORY) Hidden or Colored Money:

In an ideal Cloud with only MAC currencies, colored money can co-exist if (not limited to) some money trails are missing, due to "cloud corruption", systemic failure, hardware and network issues etc.,. Probably this is the direct consequence of CAP theorem and can be conjectured to be undecidable. Hidden money is to some extent dependent on quantity of net flow (if non-zero) and how much of this net flow is contributed by Rich vertices and Edges. Money Circulation with Colored money can be formulated as Network Flow Problem with Time horizon and storage at nodes. Time horizon implies a certain flow has to happen before a stop time. Flow conservation is affected by this Dynamic Flow because of storage at nodes and money entering a node need not be equal to money leaving. Thus not all Hidden/Colored money is illegal in theoretical terms. As mentioned in 39.1, storage is simulated with a closed loop at nodes so that flow conservation is not seemingly violated. Profiteering is achieved by money flows over time in financial markets by assigining a multiplicative factor at each edge which accrues through a cycle and comes back to start node in cycle with a magnification in value. BlockChain techniques maintain ledgers which record all transactions globally thus decimating hidden unaccounted wealth if any. KingCobra experimental MAC currency relies on unique global identifier and global refcounts with atomic cloud transactions in linux kernel. Money Flow Graph mentioned in 34-39 has striking resemblance to Money Flow Markets already studied in Algorithmic Game Theory(AGT). 39.3 primarily devotes to Pricing and Equilibrium of Edges in Money Flow Graphs and not much on Colored Money Flow. An algorithm to find colored money flow could be a major advance in AGT. Prima facie there exists no zero-knowledge blackbox proof algorithm to find colored money because all storage data is a prerequisite which is impossible to know. Money trail for MAC currency described in 22,27,28 requires tracking of currencies. There are recent dollar and euro bills issued with Radio Frequency ID tags (RFID).

Total storage of money in Flow Market Graph = | Incoming money flow at Source - Incoming money flow at Sink | i.e Flow conservation is no longer obeyed.

There is a special vertex in Money Flow Market designated as Direct Taxation Hub which has incoming direct tax money flow edges from all other vertices in Flow market.Colored money is then approximately the taxed storage money estimated above minus the net flow of money received at Taxation Hub.

Colored Money = | Total storage money * Direct Taxation rate - Incoming Flow at Direct Taxation Hub Node|
Colored Money = | Incoming money flow at Source * Direct Taxation rate - Incoming money flow at Sink * Direct Taxation rate - Incoming Flow at Direct Taxation Hub Node |

Previous is an approximate naive zero-knowledge estimation of Colored money in Money Flow Market. This assumes that there is always a sink in Money Flow Market which is not necessarily valid unless notes are returned to mint. Satellite RFID Tracking technologies though invasive and intrusive like previous can present a rough figure of total money circulation in Source and Sink.

Above estimate is for direct taxation and resultant evasion. For indirect taxes

(on Goods and Services etc.,), there is no direct impact on the money component and only G&S are affected. Combining Goods,Services and Income into one (direct+indirect) taxation could have a negative effect on Colored money because incentive to hide money no longer exists. For example, if only Goods and Services consumed by high-income brackets are preferentially taxed at high percentage replacing tax on income, income is also indirectly taxed in addition to G&S. Thus there is no necessity to tax income and any advantage of such direct taxation is indirectly usurped by preferential G&S taxation. Previous zero-knowledge estimate then becomes:

Colored Money = | Total Goods and Services * Taxation rate - Incoming Flow at Indirect Taxation Hub Node|
Colored Money = | GDP^2 * Tax-to-GDP ratio - Incoming Flow at Indirect Taxation Hub Node| where GDP is assumed to be equal to Total Goods and Services.

References:

39.1 Network Flows over time over Storage Area Networks (SAN) - https://hal.inria.fr/inria-00071643/document
39.2 Network Flow - [GoldbergTardosTarjan] - http://www.cs.cornell.edu/~eva/network.flow.algorithms.pdf
39.3 Algorithmic Game Theory - Flow Markets - [TimRoughGarden] - http://theory.stanford.edu/~tim/books.html
39.4 RFID tagged currencies - $100 bill - http://www.businessinsider.in/New-Smart-Paper-Could-Put-An-End-To-Dark-Money/articleshow/21134569.cms
39.5 Cons of RFID currencies - http://www.prisonplanet.com/022904rfidtagsexplode.html
39.6 Mechanism Design and Machine Learning - https://www.cs.cmu.edu/~mblum/search/AGTML35.pdf - Design of algorithms for maximizing gain in auctions involving sellers and buyers
39.7 Financial and Economic Networks - https://supernet.isenberg.umass.edu/bookser/innov-ch1.pdf

---
Commits as on 1 March 2014

Example java Publisher and Listeners that use ActiveMQ as the messaging middleware have been committed to repository for an ActiveMQ queue instance created for KingCobra. For multiple clients this might have to be a Topic rather than Queue instance. Request types above and a workflow framework can be added on this. This will be a JMS compliant implementation which might slow down compared to a linux workqueue or queue implementation being done in VIRGO.

Commits as on 17 March 2014

KingCobra userspace library and kernelspace driver module have been implemented that are invoked 1) either in usermode by call_usermodehelper()
2) or through intermodule invocation through exported symbols in KingCobra kernel module, by the workqueue handler in VIRGO workqueue implementation.

Commits as on 22 March 2014

Minimalistic Kernelspace messaging server framework with kernel workqueue,handler and remote cloud client has been completed - For this VIRGO clone cpupooling driver has been added a clause based on a boolean flag, to direct incoming request from remote client to VIRGO linux workqueue which is popped by workqueue handler that invokes a servicerequest function on the KingCobra kernel module. (Build notes: To remove any build or symbol errors, Module.symvers from VIRGO queue has to be copied to VIRGO cloudexec and built to get a unified VIRGO cloudexec Module.symvers that has exported symbol definitions for push_request()). End-to-end test with telnet path client sending a request to VIRGO cloudexec service, that gets queued in kernel workqueue,

handled by workqueue handler that finally invokes KingCobra service request function has been done and the kern.log has been added to repository at drivers/virgo/queuing/test_logs/

Commits as on 29 March 2014

Initial commits for KingCobra Request Response done by adding 2 new functions parse_ip_address() and reply_to_publisher() in kingcobra_servicerequest_kernelspace()

Commits as on 30 March 2014

Both VIRGO cpupooling and mempooling drivers have been modified with use_as_kingcobra_service boolean flag for sending incoming remote cloud node requests to VIRGO queue which is serviced by workqueue handler and KingCobra service as per the above schematic diagram and replied to.

Commits as on 6 April 2014

Fixes for REQUEST and REPLY headers for KingCobra has been made in virgo_cloudexec_mempool recvfrom() if clause and in request parser in KingCobra with strsep(). This has been implemented only in VIRGO mempool codepath and not in VIRGO clone.

Commits as on 7 April 2014

New function parse_timestamp() has been added to retrieve the timestamp set by the VIRGO mempool driver before pushing the request to VIRGO queue driver

---
Commits as on 29 April 2014

Intial commits for disk persistence of KingCobra request-reply queue messages have been done with addition of new boolean flag kingcobra_disk_persistence. VFS calls are used to open and write to the queue.

Commits as on 26 August 2014

KingCobra driver has been ported to 3.15.5 kernel and bugs related to a kernel_recvmsg() crash, timestamp parsing etc., have been fixed. The random crashes were most likely due to incorrect parameters to filp_open() of disk persistence file and filesystem being mounted as read-only.

Version 14.9.9 release tagged on 9 September 2014

Version 15.1.8 release tagged on 8 January 2015

Commits as on 17 August 2015

KingCobra + VIRGO Queuing port of Linux Kernel 4.1.5 :
 - changed the REQUEST_REPLY.queue disk persisted queue path to /var/log/kingcobra/REQUEST_REPLY.queue
 - kernel built sources, object files
 - kern.log with logs for telnet request sent to VIRGO queue driver, queued in kernel work queue and handler invocation for the

KingCobra service request kernel function for the popped request; disk persisted /var/log/kingcobra/REQUEST_REPLY.queue

---
Commits as on 14 October 2015

AsFer Cloud Perfect Forwarding binaries are invoked through call_usermodehelper() in VIRGO queue. KingCobra commands has been updated with a clause for cloud perfect forwarding.

Commits as on 15 October 2015

- Updated KingCobra module binaries and build generated sources
- kingcobra_usermode_log.txt with "not found" error from output redirection (kingcobra_commands.c). This error is due to need for absolute path. But there are "alloc_fd: slot 1 not NULL!" after fd_install() is uncommented in virgo_queue.h call_usermodehelper() code. The kern.log with these errors has been added to testlogs
- kingcobra_commands.c has been changed to invoke absolute path executable. With uncommenting of fd_install and set_ds code in virgo_queue the return code of call_usermodehelper() is 0 indicating successful invocation

---
Commits as on 10 January 2016

NeuronRain KingCobra enterprise version 2016.1.10 released.

NEURONRAIN VIRGO Commits for virgo_clone()/telnet -> VIRGO cpupooling -> VIRGO Queue -> KingCobra
- as on 12 February 2016

VIRGO commit:
https://github.com/shrinivaasanka/virgo-linux-github-code/commit/72d9cfc90855719542cdb62ce40b798cc7431b3d

Commit comments:

Commits for Telnet/System Call Interface to VIRGO CPUPooling -> VIRGO Queue -> KingCobra
---
*) This was commented earlier for the past few years due to a serious kernel panic in previous kernel versions - <= 3.15.5
*) In 4.1.5 a deadlock between VIRGO CPUPooling and VIRGO queue driver init was causing following error in "use_as_kingcobra_service" clause :
  - "gave up waiting for virgo_queue init, unknown symbol push_request()"
*) To address this a new boolean flag to selectively enable and disable VIRGO Queue kernel service mode "virgo_queue_reactor_service_mode" has been added.
*) With this flag VIRGO Queue is both a kernel service driver and a standalone exporter of function symbols - push_request/pop_request
*) Incoming request data from telnet/virgo_clone() system call into cpupooling kernel service reactor pattern (virgo cpupooling listener loop) is treated as generic string and handed over to VIRGO queue and KingCobra which publishes it.
*) This resolves a long standing deadlock above between VIRGO cpupooling "use_as_kingcobra_service" clause and VIRGO queue init.
*) This makes virgo_clone() systemcall/telnet both synchronous and asynchronous - requests from telnet client/virgo_clone() system call can be either synchronous RPC functions executed on a remote cloud node in kernelspace (or) an asynchronous invocation through "use_as_kingcobra_service"

clause path to VIRGO Queue driver which enqueues the data in kernel workqueue and subsequently popped by KingCobra.
*) Above saves an additional code implementation for virgo_queue syscall paths - virgo_clone() handles, based on config selected, incoming
data passed to it either as a remote procedure call or as a data that is pushed to VIRGO Queue/KingCobra pub-sub kernelspace
*) Kernel Logs and REQUEST_REPLY.queue for above commits have been added to kingcobra c-src/testlogs/

Commits - KingCobra 64 bit and VIRGO Queue + KingCobra telnet requests - 17 April 2017
---
*) Rebuilt KingCobra 64bit kernel module
*) telnet requests to VIRGO64 Queueing module listener driver are serviced by KingCobra servicerequest
*) Request_Reply queue persisted for this VIRGO Queue + KingCobra routing has been committed to c-src/testlogs.
*) kern.log for this routing has been committed in VIRGO64 queueing directory
) Similar to other drivers struct socket reinterpret cast to int has been removed and has been made const in queuesvc kernel thread

---
(FEATURE-DONE) Commits - CVXPY implementation for Eisenberg-Gale Convex Program - 18 August 2017

(*) First commits for Convex Optimized Market Equilibrium Prices
(*) Imports CVXPY Convex Program solver
(*) Objective function is a logistic variant of Eisenberg-Gale Convex Program i.e uses money * log(1+e^utility) instead of
 money * log(utility) because of curvature error (log is error flagged as concave and logistic is convex per:
 http://www.cvxpy.org/en/latest/tutorial/functions/index.html#vector-matrix-functions)
(*) Formulates constraints and objective functions based on http://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf - Page 106 and Equation 5.1
(*) But, For all installed solvers ECOS, ECOS_BB, SCS, LS solved convex program prints value as None despite all constraints and objective functions being convex. Also is_dcp() prints "not a disciplined convex program". Logs in testlogs/.
(*) Obviously it should have worked. Therefore this is only a partial implementation commit.
(*) This implementation uses numpy randomly initialized arrays for Money each buyer has and per-good utility(happiness) each buyer has.
(*) Replacing money with perceived merit values translates this Market Equilibrium - Intrinsic Value versus Market Price - to Merit
 Equilibrium - Intrinsic Merit versus Perceived Merit. This has been already described in NeuronRain AsFer Design Documents:
        - http://sourceforge.net/p/asfer/code/HEAD/tree/asfer-docs/AstroInferDesign.txt and
        - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

---
(FEATURE-DONE) Commits - Convex Optimization - DCCP - 21 August 2017

(*) import dccp has been added
(*) DCCP is the recent advancement and generalization of DCP for convex-concave programs
(*) method='dccp' has been added as parameter to solve()
(*) Objective function has been changed to log() from logistic() - curvature is concave which is in conflict with definition of
eisenberg-gale convex program in textbooks. Reason for this contradiction is unknown.
(*) But DCCP overcomes the DCP limitation and solve() prints converged solutions for objective functions
(*) logs have been committed to testlogs/
(*) CVXOPT solver has been installed but it does not solve the Eisenberg-Gale objective function. Only SCS solver works - by default applies
KKT conditions indirectly.

(FEATURE-DONE) Commits - Convex Optimization - DCCP - 22 August 2017

(*) Verbose set to True for printing Splitting Conic Solver progress information
(*) logs committed to testlogs/

(FEATURE-DONE) Commits - Convex Optimization update - 29 August 2017

(*) Removed hardcoded variable values in objective and constraints
(*) In the context of pricing, ECOS Error Metrics print the matrices of market clearing prices for goods
(Reference - pages 3072 and 3073 of
https://web.stanford.edu/~boyd/papers/pdf/ecos_ecc.pdf - KKT conditions in ECOS solver)

(FEATURE-DONE) Convex Optimization - Pricing Computation - 30 August 2017

(*) Prices of Goods/Services have been computed explicitly from Karush-Kuhn-Tucker Conditions (1,2,3 and especially 4)
(*) References:
      - Pages 106-108 of
http://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf)
      - KKT conditions and Conic Optimization-
https://arxiv.org/pdf/1312.3039.pdf
(*) logs committed to testlogs/

NEURONRAIN KingCobra - Module for Kernelspace Messaging and Computational Economics

```
You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

#-------------------------------------------------------------------------------
-----------------------
#K.Srinivasan
#NeuronRain Documentation and Licensing: http://neuronrain-
documentation.readthedocs.io/en/latest/
#Personal website(research): https://sites.google.com/site/kuja27/
#-------------------------------------------------------------------------------
-----------------------
*******************************************************************************
*********/

Theoretical Interludes, Design Notes and ToDo (long term with no deadline)
--------------------------------------------------------------------------

[This is a major research oriented subsystem of NeuronRain and inspired by COBRA
project (done by the author in his BE (1995-1999) along with few other esteemed
classmates.
http://sourceforge.net/projects/acadpdrafts/files/Excerpts_Of_PSG_BE_FinalProjec
t_COBRA_done_in_1999.pdf/download.)]

[KingCobra though a misnomer is expanded as ClOud With ARBiters MimicKING
containing the anagram]

1072. (THEORY) There is a cloud of nodes which execute a set of services from
randomly created clients.

1073. (THEORY) This cloud could be on NeuronRain (AsFer+USBmd+VIRGO+KingCobra)
platform or any other opensource cloud platforms like Hadoop Cluster.

1074. (THEORY) The Clients are publishers of Service requests which are of many
types -  miscellaneous types of Service that could be dynamically added through
other kernel modules and invoked through a switch-case or embedded in function
itself. Identified by unique id(s) for different types of services (for example
Problem reports, Suggestions etc.,)

1075. (THEORY) The Services on the Cloud are Subscribers to these requests of
specific type. Thus this is the conventional publisher-subscriber model.

1076. (THEORY) The requests flow through cloud using a workqueue (which could be
a lowlevel Linux workqueue or VIRGO queue or some other queuing middleware
software like ActiveMQ). The publishers enqueue and Subscribers dequeue the
requests.

1077. (THEORY) The difference is that the Cloud has nodes that "deceive" or
"corrupt".

1078. (THEORY) Service requests - are published by the clients in the need of a
service which could be defined by markup file. These requests are scheduled and
routed by the middleware to competent authority which services it (with or
without timeframe) and replies to the client.

1079. (THEORY) Problem reports - are published by clients which are
"dissatisfied" by the quality of service by the cloud. These are analyzed by
"arbiters" in the cloud which find the faulting node(s) and take action. This
allows manual intervention but minimizes it.

1080. (THEORY) Suggestions - are enhancement requests sent by clients and
require manual intervention.

1081. (THEORY) Cloud nodes have a Quality of Service metric calculated by a
model.
```

1082. (THEORY) The cloud has a reporting structure of nodes - either as a graph or tree. The graph is dynamically reorganized by weighting the Quality of Service of each node.

1083. (THEORY) The difficult part of the above is using Arbiters to find "faulty" nodes based on problem reports from clients.

1084. (THEORY) Brewer's CAP conjecture proved by [GilbertLynch] as a theorem (still debated) states that only 2 of the 3 (Consistency of data, Avaliability of data and Partition tolerance when some nodes or messages are lost) can be guaranteed and not all 3 are simultaneously achievable.

1085. (THEORY) CAP theorem does not seem to apply to the above faulty scenario with corrupt nodes under Consistency or Availablity or Partition Tolerance. This is because a corrupt node can have any 2 of the 3 - it can give consistent data, is available with success response or can make the cloud work with missing data in partition tolerance but yet can "corrupt" the cloud. Probably this needs to be defined as a new attribute called Integrity.

1086. (THEORY) As "corruption" is more conspicuous with monetary element, if above services are "charged" with a logical currency (e.g. bitcoin), then corruption in cloud is defineable approximately as (but not limited to)- "Undue favour or harm meted out to a client not commensurate with the charge for the service (or) unreasonable extra logical currency demanded to execute the service of same quality (or) deliberate obstruction of justice to a client with malevolent and unholy collusion with other cloud nodes with feigned CAP".

1087. (THEORY) Identifying criminal nodes as in (15) above seems to be beyond the ambit of CAP. Thus CAP with Integrity further places a theoretical limit on "pure" cloud. If Integrity is viewed as a Byzantine problem with faulty or corrupt processes in a distributed system, and if resilience factor is rf (expected number of faulty nodes), then most algorithms can ensure a "working" cloud only if resilience is ~30% or less (3*rf+1) of the total number of cloud nodes. Probably this could apply to Integrity also that places a limit of 30% on "corrupt nodes" for the Cloud to work with sanity. Translating this to a Governance problem, a corruption-free administration is achievable with a maximum limit of 30% "corrupt" elements.

1088. (THEORY and FEATURE) Analytics on the Problem reports sent to the cloud queue give a pattern of corrupt nodes. Intrinsic Merit ranking with Citation graph maxflow considering cloud as a flow network where a node positively or negatively cites or "opines" about a node, as mentioned in http://arxiv.org/abs/1006.4458 (author's Master's thesis) and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf(published by the author during PhD) give a p2p ranking of cloud nodes that can be used for analysis though may not be reliable fully. AsFer has bigdata analytics functionality that fits well to this point to analyse the problem reports with machine learning algorithms and set the key-value pairs that are read by VIRGO kernel_analytics module and exported kernelwide. The persisted REQUEST_REPLY.queue with the logged request-reply IPs and timestamps can be mined with AsFer bigdata capability (e.g. Spark)

1089. (THEORY) Policing the cloud nodes with arbiters - This seems to be limited by CAP theorem and Integrity as above. Also this is reducible to perfect inference problem in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt and drafts in https://sites.google.com/site/kuja27/

1090. (THEORY) Brooks-Iyengar algorithm for sensors in all cloud nodes is an improved Byazantine Fault Tolerant algorithm.

1091. (THEORY) BitCoin is a Byzantine Fault Tolerant protocol.

1092. (THEORY) Byzantine Fault Tolerance in Clouds is described in http://www.computer.org/csdl/proceedings/cloud/2011/4460/00/4460a444-abs.html, http://www.eurecom.fr/~vukolic/ByzantineEmpire.pdf which is more on Cloud of Clouds - Intercloud with cloud nodes that have malicious or corrupt software. Most of the key-value(get/set) implementations do not have byzantine nodes (for example CAP without Byzantine nodes in Amazon Dynamo: http://www.eurecom.fr/~michiard/teaching/slides/clouds/cap-dynamo.pdf)

1093. (THEORY) Related to point 18 - The problem of fact finding or fault finding using a cloud police has the same limitation as the "perfect inference" described in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt. "Money trail" involving the suspect node in point 28 is important to conclude something about the corruption. In real world tracking money trail is a daunting task. In cloud that abides by CAP, missing messages on trail can prevent reaching a conclusion thereby creating benefit-of-doubt. Also fixing exact value for a transaction that involves MAC currency message is undecidable - a normative economics problem can never be solved by exact theoretical computer science.

1094. (THEORY) Reference article on cloud BFT for Byzantine, Corrupt brokers - Byzantine Fault-Tolerant Publish/Subscribe: A Cloud Computing Infrastructure (www.ux.uis.no/~meling/papers/2012-bftps-srdsw.pdf)

1095. KingCobra messaging request-response design - options

1095a. Implementing a message subscription model in kernelspace where clients publish the message that is queued-in to subscribers' queue (Topic like implementation - use of ActiveMQ C implementation if available).

1095b. (DONE-minimal implementation) At present a minimum kernelspace messaging system that queues remote request and handles through workqueue handler is in place. This responds to the client once the Kingcobra servicerequest function finishes processing the request(reply_to_publisher() in KingCobra driver). Unlike the usual messaging server, in which client publishes messages of a particular type that are listened to by interested clients, one option is to continue the status-quo of KingCobra as a peer-to-peer messaging system. Thus every VIRGO node is both a kernelspace messaging client and server that can both publish and listen. Every message in the cloud can have a universally unique id assigned by a timestamp server - https://tools.ietf.org/html/rfc4122 (similar to bitcoin protocol) so that each message floating in the cloud is unique across the cloud (or) no two messages on the VIRGO cloud are same. The recipient node executing kingcobra_servicerequest_kernelspace() parses the unique-id (example naive unique-id is <ip-address:port>#localtimestampofmachine which is a simplified version of RFC4122) from the incoming remote request and responds to the remote client through kernel socket connection that gets queued-in the remote client and handled similar to incoming remote request. To differentiate request and response-for-request response messages are padded with a string "REPLY:<unique-id-of-message>" and requests are padded with "REQUEST:<unique-id-of-message>". This is more or less similar to TCP flow-control with SEQ numbers but state-less like UDP. Simple analogy is post-office protocol with reference numbers for each mail and its reply. Thus there are chronologically two queues: (1) queue at the remote VIRGO cloud service node for request (2) queue at the remote client for response to the request sent in (1). Thus any cloudnode can have two types of messages - REQUEST and REPLY. Following schematic diagram has been implemented so far.

1095c.(DONE) KingCobra - VIRGO queue - VIRGO cpupooling , mempooling and queue service drivers interaction schematic diagram:

1095d. (ONGOING) kingcobra_servicerequest_kernelspace() distinguishes the "REQUEST" and "REPLY" and optionally persists them to corresponding on-disk filesystem. Thus a disk persistence for the queued messages can either be implemented in 1) VIRGO queue driver 2) workqueue.c (kernel itself needs a rewrite (or) 3) KingCobra driver. Option (2) is difficult in the sense that it could impact the kernel as-a-whole whereas 1) and 3) are modularized. At present Option 3 persistence within KingCobra driver has been implemented.

1095e.Above option 24b implements a simple p2p queue messaging in kernel. To get a Topic-like behaviour in VIRGO queue might be difficult as queue_work() kernel function has to be repeatedly invoked for the same work_struct on multiple queues which are subscribers of that message in AMQ protocol. Moreover creating a queue at runtime on need basis looks difficult in kernel which is usually done through some CLI or GUI interface in ActiveMQ and other messaging servers.

1096. (ONGOING) For the timestamp service, EventNet described in http://sourceforge.net/p/asfer/code/HEAD/tree/AstroInferDesign.txt is a likely implementation choice. AsFer already has a primitive text files based EventNet graph implementation in place. Periodic topological sort (quite expensive) of EventNet gives logical ordering and thus a logical timestamp of the cloud events.

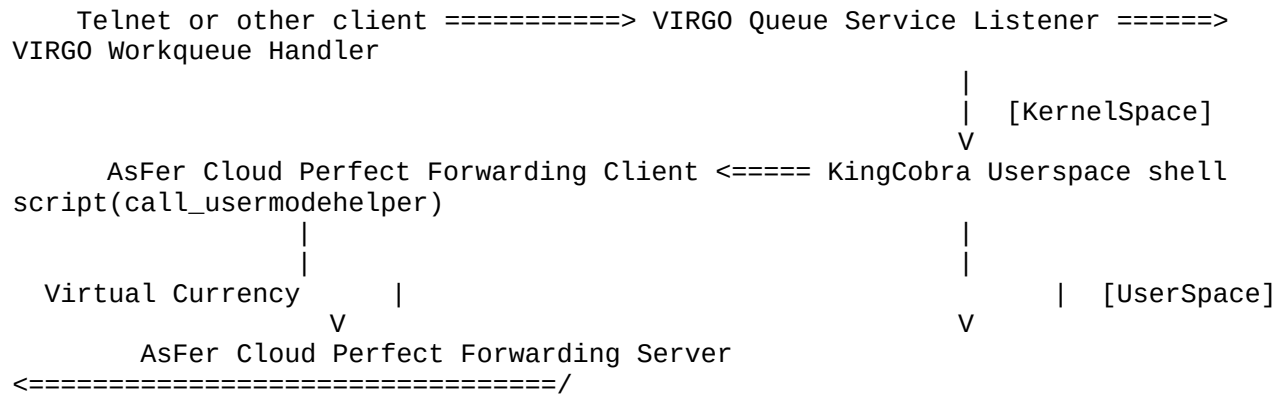
784. (THEORY - Neuro Cryptocurrency Implemented in AstroInfer - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt) MESSAGE-AS-CURRENCY PROTOCOL: If each message payload is also construed as a currency carrier, each message id can be mapped to a unique currency or a coin with fixed denomination. This is similar to each currency note having a serial number as unique id. Uniqueness is guaranteed since there can be only one message (or coin) with that id on the cloud. This simulates a scenaio - "Sender of the message pays the Receiver with a coin having the unique id and Receiver acknowledges receipt". This is an alternative to BitCoin protocol. Double spending is also prohibited since at any point in time the

message or "coin" with unique id can be sent by only one node in the cloud. Unique Cloudwide Timestamp server mimicks the functionality of "Mint". There is a difference here between conventional send-receive of messages - Once a message is sent to remote cloud node, no copy of it should exist anywhere in the cloud. That is, every MAC currency message is a cloudwide singleton. In pseudocode this is expressible as:

```
      m1=MAC_alloc(denomination)
      m2=m1 (---- this is disallowed)
```

Linux kernel allocation functions - kmalloc() - have a krefs functionality for reference counting within kernel. Refcount for MAC message can never exceed 1 across cloud for above singleton functionality - this has to be a clause everywhere for any unique MAC id. This requires a cloudwide krefs rather. Buyer decrements cloudwide kref and Seller increments it. In C++ this is done by std::move() and often required in "Perfect Forwarding" - http://thbecker.net/articles/rvalue_references/section_07.html - within single addressspace. By overloading operator=() with Type&& rvalue reference, the necessary networking code can be invoked that does the move which might include serialization. But unfortunately C++ and Linux kernel are not compatible. The Currency object has to be language neutral and thus Google Protocol Buffers which have C,C++,Java, Python .proto files compilers support might be useful but yet the move semantics in Kernel/C is non-trivial that requires cloudwide transactional kernel memory as mentioned in (31) below. A C++ standalone userspace client-server cloud object move implementation based on std::move() over network of Protocol Buffer Currency Objects has been added to AsFer repository at - http://sourceforge.net/p/asfer/code/HEAD/tree/cpp-src/cloud_move which can optionally be upcall-ed to userspace from VIRGO and KingCobra drivers. This C++ implementation is invoked in userspace with call_usermodehelper() from VIRGO Queue Messaging via the kernel workqueue handler.

784.1 Schematic Diagram for Cloud Perfect Forwarding with AsFer+VIRGOQueue+KingCobraUserspace:



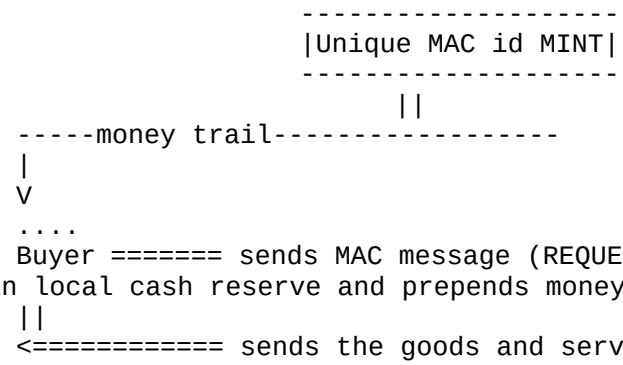
References:

784.2 An example distributed transactional memory implementation in cloud - http://infinispan.org/tutorials/simple/tx/ and http://www.cloudtm.eu/ - these are in userspace cloud (C++ and Java) and may not have cloud move functionality - move has to be simulated in a transaction: replicate data, delete in one endpoint and create in other endpoint

1097. (THEORY) SIMULATING A VIRTUAL ECONOMY with above MAC protocol (Message-as-currency): If each message sent is considered as "money" element and cloud nodes and clients are the consumers and producers of "electronic money", the timestamp "Mint" becomes a virtual Federal Reserve or Central Bank that controls the "electronic money" circulation in the cloud. Infact any REPLY messages could be mapped to a Service a client derives by s(p)ending the REQUEST "money message". Thus value(REQUEST) should equal value(REPLY) where value() is a function that

measures the value of a money denomination and the value of goods and/or services for that money. For example Rs.10000 or $10000 has no meaning if it doesn't translate into a value (analogy: erstwhile Gold Standard).When the value() function gets skewed phenomena like Inflation arise. Thus above model could also have a notion of value() and "electronic money inflation". Thus any "message money" with a unique id assigned by the cloud unique id(or logical timestamp) server can exist at most in only one node in the cloud. Money trail can be implemented by prefixing a header to the incoming message money in each cloud node that receives the money which traces the "path" taken. Cloud has to implement some Byzantine Fault Tolerant protocol. The value() function to some extent can measure the "deceit" as above. When a Buyer and Seller's value() functions are at loggerheads then that is starting point of "cloud corruption" at either side and might be an undecidable problem.

785. (THEORY - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt) TRADING WITH ABOVE KINGCOBRA MAC protocol - somewhat oversimplified:



In the above schematic, money with unique id in cloud reaches a buyer after many buyer-seller transitions called "money trail". The MAC currency is prefixed by each node to create a chain. Buyer then sends a request to the seller through MAC virtual currency and seller replies with goods and services. Seller prepends the money trail chain. When a transaction occurs the whole cloud need not be notified about it except only buyer and seller. MAC Mint could create a bulk of money denominations and circulate them in cloud economy.

References:

785.1 Price fixing for items in Buyer-Seller-Trader networks - Trading Networks - Market Equilibrium and Walrasian Model of Price fixing - http://www.cs.cornell.edu/~eva/traders.pdf
785.2 Algoithmic Game Theory - Market Equilibrium for Price - Equilibrium is a strategic standoff - both players can't better their own present by changing strategies e.g Buyer-Sellers are market players and equilibrium price is the one where both buyer and seller can't gain by varying it - http://www.cis.upenn.edu/~mkearns/nips02tutorial/nips.pdf. Buyer-Seller payoff matrix picturises the bargaining problem.
785.3 Price-setting in Trading networks - Chapter 11 - https://www.cs.cornell.edu/home/kleinber/networks-book/networks-book.pdf

1098. (THEORY) VALUE FOR ELECTRONIC MONEY: How is above MAC money earned - This again requires linking value to money (as money is not a value by itself and only a pointer to valuable item or resource). Thus any buyer can "earn" MAC money by something similar to a barter.

786. (THEORY and IMPLEMENTATION - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt) FIXING VALUE FOR MAC MONEY: To delineate corruption as discussed in 27 above with value() disparity between MAC money and REPLY goods

and services, an arbiter node in cloud has to "judge" the value of MAC sent by buyer and goods and services from seller and proclaim if corruption exists. Thus value() function itself has to be some kind of machine learning algorithm. This is related to or same as points 12 to 23 above. For example, while buying an item for few million bucks, value() has to take as input the description of the item and calculate the value "ideally" which is difficult. Because there are no perfect references to evaluate and only a weighted average of available market price range has to be taken as a reference which is error-prone. value() function can be recursively defined as("Reductionism" and a variant of Ricardo Labor theory of value - value of a commodity is proportional to cost of labor required to make it - e.g Software is approximately valued by lines of code,concept patent innovations and manhours):

```
      ------------------------------------------------------------------------
-------------------------
      |value(i) = summation(value(ingredients of i)) + cost(integrating the
ingredients to create item i) |
      ------------------------------------------------------------------------
-------------------------
```

Obviously the above recursion combinatorially explodes into exponential number of nodes in the recursion tree. Ideally recursion has to go deep upto quarks and leptons that makeup the standard model. If for practical purposes, recursion depth is restricted to t then size of value() tree is O(m^t) where m is average number of ingredients per component. Hence any algorithm computing the value() recursion has to be exponential in time. Computation of value() in the leaf nodes of the recursion is most crucial as they percolate bottom-up. If leaf nodes of all possible items are same (like quarks and leptons making up universe) then such atomic ingredient has to have "same" value for all items. Only the integration cost varies in the levels of the tree.For infinite case, value() function is conjectured to be undecidable - probably invoking some halting problem reduction .But above value() function could be Fixed Parameter Tractable in parameter recursion depth - t but yet could only be an approximation. A Turing machine computing value() function exactly might loop forever and thus Recursively Enumerable and not Recursive. A CVXPY implementation for Pricing Market Equilibrium has been implemented in KingCobra.

---

787. (THEORY - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt) Buyer-Seller and MAC electronic money transaction schematic:

```
      Buyer        A-------<id><refcnt:0>----------------------> Seller
<id><refcnt:1> (increments refcnt)
      (<id><refcnt:1>  |
       <id><refcnt:0>  |
      after decrement  |
      refcnt           |
      )---------------->
```

Above has to be transactional (i.e atomic across cloud nodes)

1099. (THEORY) MAC protocol reaper

Reaper thread in each cloud node harvests the zero refcounted allocations and invokes destructors on them. Same MAC id cannot have kref count of 1 or above in more than one cloud node due to the transaction mentioned previously.

1100. (THEORY) Cloud Policing With Arbiters - Revisited:

When a suspect node is analyzed when a complaint problem is filed on it, (1) it is of foremost importance on how flawless is the arbiter who investigates on that and is there a perfect way to choose a perfect arbiter. In the absence of the previous credibility of entire cloud judiciary is blown to smithereens and falls apart. (2) Assuming a perfect arbiter which is questionable, next thing is

to analyze the credibility of the node who sulked. This is nothing but the Citation problem in http://arxiv.org/abs/1106.4102 and http://www.nist.gov/tac/publications/2010/participant.papers/CMI_IIT.proceedings.pdf where a node can positively or negatively cite another node a "Societal Norm" which can be faulted and citations/opinions could be concocted with malafide intent(perjury). This is rather a generalization of PageRank algorithm with negative citations. Thus Perfect Cloud Arbitration could be an unsolvable problem. (3) Even if both arbiter and complainant are perfect which is again questionable, there are still loopholes - lack of evidences or implicated witnesses might portray a negative impression of a positive node. Thus there are 3 tiers of weakenings in cloud arbitration and there could be more. P(Good) series in https://sites.google.com/site/kuja27/ precisely addresses this problem.

---

788. (THEORY) MAC Money Flow as MaxFlow problem - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

---

Transactions happening in a cloud are edges between the nodes involved (buyer and seller). Thus it creates a huge directed graph. Flow of money in this graph can be modelled as Flow network. Minimum Cut of this graph shows crucial nodes in the graph which play vital role in cloud economy removal of which paralyzes the cloud (could be Central bank and other financial institutions). This graph has bidirectional edges where one direction is for money and the opposite direction is for Goods and Services. In the Flow network sum of flows is zero. But in the Money Flow Network each node is having a cash reserve ratio (CRR) due to commercial transactions which is confidential and privy to that node only and thus sum of flows can not be zero. Hub nodes in the Money Flow graph which can be obtained by getting k-core or D-core of the graph by some graph peeling algorithms are crucial nodes to the economy that contribute to Money circulation.

1101. (THEORY) Cycles and components in above MAC Money Flow Graph:

Above graph of money transactions could be cyclic which implies a supply chain. Strongly connected components of this graph are most related nodes that are in same industry.

1102. (THEORY) STOCK TRADING:

One of the component in above MAC Money Flow Graph of cloud could be a virtual Stock Exchange. Based on the financial and securities transactions of constituent organizations in the graph, index of the exchange varies.

---

789. (THEORY) Analysis of Poverty and Alleviation through above money flow graph - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

---

Weights of the edges of money flow graph are the denominations of the transaction. Thus high value edges and low value edges divide the Graph logically into Rich and Poor strata(Bourgeoisie and Proletariat subgraphs). Equitable graph is the one which does not have too much of value difference between Rich and Poor sets of edges - a utopian to achieve. Mathematically, it is an optimization LP problem that seeks to minimize sum(RichEdges)-sum(PoorEdges) or Sum(RichVertices) - sum(PoorVertices) - without harming either - to be precise. This requires money flow to be programmed to find a feasible

solution to this LP subject to constraints like work-pay parity etc.,(there could be more variables and constraints to this LP) . Due to CRR above Vertices also can be Rich and Poor in addition to Rich Edges and Poor Edges.

Previous Linear Program could be weighted to : Sum(w(i) * RichEdges) - Sum(w(k) * PoorEdges). Money Flow Graph (or Money Trail as it is commonly termed) is a potential expander of high regularity. Some constraints could be imposed on this Poverty Alleviation Linear Program e.g Number of Rich edges (x) must be atleast a non-negligible fraction of Number of poor edges (y) which guarantees equitable money trail:

```
      R(i) = Rich Edges
      P(i) = Poor Edges
minimize:
      Poverty = Sum_1_to_x(w(i) * R(i)) - Sum_1_to_y(w(k) * P(i))
subject to constraint:
      x >= epsilon * y, 0 < epsilon < 1
```

References:

789.1 J-PAL case study of social programs fund flow reforms - https://www.povertyactionlab.org/case-study/fund-flow-reforms-improved-social-program-delivery-india

---

790.(THEORY) Demand and Supply and Value() function - Quantitative Majority Circuit - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

---

Alternative to the recursive definition of value() function above can be done through Demand and Supply - more the demand and less the supply, price increases and vice-versa. This is quite subjective compared to absolute recursive definition above. To simulate demand and supply, the weights of the money edges (-> direction) in the bidirectional graph change and fluctuate dynamically over time for unchanging weights of the Goods and Services edges (<- direction) between any pair of Buyer-Seller vertices. This makes Money and G&S Flow graph a Dynamic Graph with edge weight update primitive. Parallels can be drawn between majority voting and Demand-Supply pricing - Candidates are replaced by commodities and Votes for candidates are replaced by Consumers demanding a product - but difference being the dynamism and inverse proportionality of demand-supply - Number of demanding consumers (voter leaves of majority circuit) increase with dwindling availability of commodity (Candidate voted for) which is a special variation of voting. Every commodity vertex and its buyer neighborhood in Money Trail graph could be translated to a majority voting circuit. In essence, leaves of majority circuit for demand-supply dynamically bloat or shrink depending on quantity of commodity which is a quantitative weighted version of majority circuit vis-a-vis conventional qualitative boolean and non-boolean (set partition/LSH) majority neglecting volume of a candidate. This quantitative dynamic majority circuit gadget can formalize any economic process which has fluctuating demand-supply underneath. In the context of social networks, demand-supply inverse relationship defines a least energy Intrinsic Fitness constraint - Vertex or commodity in a Market Bipartite Graph (edges between two sets - buyers and commodities) of least availability attracts most buyers and thus has most merit. Szemeredi's Regularity Lemma implies any graph can be approximately partitioned as union of bipartite graphs by which Money Trail graph can be decomposed as union of Bipartite graphs of random edge densities (buyers and sellers). Quantum of a commodity vertex in Buy-Sell Market Bipartite Graph could be represented by gradation of its color (darkest-most available to lightest-least available) which is dynamically recomputed. Section 727 of NeuronRain Unified Theory Drafts delves into Fame-Merit counterpart of this problem - Bipartite decomposition of WWW. BWBP model of majority has an

advantage of constant width 5 of variable length = 4^depth - Length of Quantitative majority BWBP varies depending on demand-supply.

References:

790.1 Barrington Theorem - Majority can be computed by Bounded Width Branching Program (BWBP) of width 5 and length 4^d - http://www.ccs.neu.edu/home/viola/classes/gems-08/lectures/le11.pdf
790.2 Szemeredi Regularity Lemma - Section 1.3 - http://www.math.ucsd.edu/~fan/teach/262/read/reg.pdf - "...the Regularity Lemma provides us with an approximation of an arbitrary dense graph with the union of a constant number of random-looking bipartite graphs..."

791.(THEORY) Hidden or Colored Money - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
---
In an ideal Cloud with only MAC currencies, colored money can co-exist if (not limited to) some money trails are missing, due to "cloud corruption", systemic failure, hardware and network issues etc.,. Probably this is the direct consequence of CAP theorem and can be conjectured to be undecidable. Hidden money is to some extent dependent on quantity of net flow (if non-zero) and how much of this net flow is contributed by Rich vertices and Edges. Money Circulation with Colored money can be formulated as Network Flow Problem with Time horizon and storage at nodes. Time horizon implies a certain flow has to happen before a stop time. Flow conservation is affected by this Dynamic Flow because of storage at nodes and money entering a node need not be equal to money leaving. Thus not all Hidden/Colored money is illegal in theoretical terms. As mentioned in 39.1, storage is simulated with a closed loop at nodes so that flow conservation is not seemingly violated. Profiteering is achieved by money flows over time in financial markets by assigining a multiplicative factor at each edge which accrues through a cycle and comes back to start node in cycle with a magnification in value. BlockChain techniques maintain ledgers which record all transactions globally thus decimating hidden unaccounted wealth if any. KingCobra experimental MAC currency relies on unique global identifier and global refcounts with atomic cloud transactions in linux kernel. Money Flow Graph mentioned in 34-39 has striking resemblance to Money Flow Markets already studied in Algorithmic Game Theory(AGT). 39.3 primarily devotes to Pricing and Equilibrium of Edges in Money Flow Graphs and not much on Colored Money Flow. An algorithm to find colored money flow could be a major advance in AGT. Prima facie there exists no zero-knowledge blackbox proof algorithm to find colored money because all storage data is a prerequisite which is impossible to know. Money trail for MAC currency described in 22,27,28 requires tracking of currencies. There are recent dollar and euro bills issued with Radio Frequency ID tags (RFID).

Total storage of money in Flow Market Graph = | Incoming money flow at Source - Incoming money flow at Sink | i.e Flow conservation is no longer obeyed.

There is a special vertex in Money Flow Market designated as Direct Taxation Hub which has incoming direct tax money flow edges from all other vertices in Flow market.Colored money is then approximately the taxed storage money estimated above minus the net flow of money received at Taxation Hub.

Colored Money = | Total storage money * Direct Taxation rate - Incoming Flow at Direct Taxation Hub Node|
Colored Money = | Incoming money flow at Source * Direct Taxation rate - Incoming money flow at Sink * Direct Taxation rate - Incoming Flow at Direct Taxation Hub Node |

Previous is an approximate naive zero-knowledge estimation of Colored money in Money Flow Market. This assumes that there is always a sink in Money Flow Market which is not necessarily valid unless notes are returned to mint. Satellite RFID Tracking technologies though invasive and intrusive like previous can present a rough figure of total money circulation in Source and Sink.

Above estimate is for direct taxation and resultant evasion. For indirect taxes (on Goods and Services etc.,), there is no direct impact on the money component and only G&S are affected. Combining Goods,Services and Income into one (direct+indirect) taxation could have a negative effect on Colored money because incentive to hide money no longer exists. For example, if only Goods and Services consumed by high-income brackets are preferentially taxed at high percentage replacing tax on income, income is also indirectly taxed in addition to G&S. Thus there is no necessity to tax income and any advantage of such direct taxation is indirectly usurped by preferential G&S taxation. Previous zero-knowledge estimate then becomes:

Colored Money = | Total Goods and Services * Taxation rate - Incoming Flow at Indirect Taxation Hub Node|
Colored Money = | GDP^2 * Tax-to-GDP ratio - Incoming Flow at Indirect Taxation Hub Node| where GDP is assumed to be equal to Total Goods and Services.

References:

791.1 Network Flows over time over Storage Area Networks (SAN) - https://hal.inria.fr/inria-00071643/document
791.2 Network Flow - [GoldbergTardosTarjan] - http://www.cs.cornell.edu/~eva/network.flow.algorithms.pdf
791.3 Algorithmic Game Theory - Flow Markets - [TimRoughGarden] - http://theory.stanford.edu/~tim/books.html
791.4 RFID tagged currencies - $100 bill - http://www.businessinsider.in/New-Smart-Paper-Could-Put-An-End-To-Dark-Money/articleshow/21134569.cms
791.5 Cons of RFID currencies - http://www.prisonplanet.com/022904rfidtagsexplode.html
791.6 Mechanism Design and Machine Learning - https://www.cs.cmu.edu/~mblum/search/AGTML35.pdf - Design of algorithms for maximizing gain in auctions involving sellers and buyers
791.7 Financial and Economic Networks - https://supernet.isenberg.umass.edu/bookser/innov-ch1.pdf

---
1103. Commits as on 1 March 2014

Example java Publisher and Listeners that use ActiveMQ as the messaging middleware have been committed to repository for an ActiveMQ queue instance created for KingCobra. For multiple clients this might have to be a Topic rather than Queue instance. Request types above and a workflow framework can be added on this. This will be a JMS compliant implementation which might slow down compared to a linux workqueue or queue implementation being done in VIRGO.

1104. Commits as on 17 March 2014

KingCobra userspace library and kernelspace driver module have been implemented that are invoked 1) either in usermode by call_usermodehelper()
2) or through intermodule invocation through exported symbols in KingCobra kernel module, by the workqueue handler in VIRGO workqueue implementation.

1105. Commits as on 22 March 2014

Minimalistic Kernelspace messaging server framework with kernel workqueue,handler and remote cloud client has been completed - For this VIRGO clone cpupooling driver has been added a clause based on a boolean flag, to

direct incoming request from remote client to VIRGO linux workqueue which is popped by workqueue handler that invokes a servicerequest function on the KingCobra kernel module. (Build notes: To remove any build or symbol errors, Module.symvers from VIRGO queue has to be copied to VIRGO cloudexec and built to get a unified VIRGO cloudexec Module.symvers that has exported symbol definitions for push_request()). End-to-end test with telnet path client sending a request to VIRGO cloudexec service, that gets queued in kernel workqueue, handled by workqueue handler that finally invokes KingCobra service request function has been done and the kern.log has been added to repository at drivers/virgo/queuing/test_logs/

1106. Commits as on 29 March 2014

Initial commits for KingCobra Request Response done by adding 2 new functions parse_ip_address() and reply_to_publisher() in kingcobra_servicerequest_kernelspace()

1107. Commits as on 30 March 2014

Both VIRGO cpupooling and mempooling drivers have been modified with use_as_kingcobra_service boolean flag for sending incoming remote cloud node requests to VIRGO queue which is serviced by workqueue handler and KingCobra service as per the above schematic diagram and replied to.

1108. Commits as on 6 April 2014

Fixes for REQUEST and REPLY headers for KingCobra has been made in virgo_cloudexec_mempool recvfrom() if clause and in request parser in KingCobra with strsep(). This has been implemented only in VIRGO mempool codepath and not in VIRGO clone.

1109. Commits as on 7 April 2014

New function parse_timestamp() has been added to retrieve the timestamp set by the VIRGO mempool driver before pushing the request to VIRGO queue driver

---
1110. Commits as on 29 April 2014

Intial commits for disk persistence of KingCobra request-reply queue messages have been done with addition of new boolean flag kingcobra_disk_persistence. VFS calls are used to open and write to the queue.

1111. Commits as on 26 August 2014

KingCobra driver has been ported to 3.15.5 kernel and bugs related to a kernel_recvmsg() crash, timestamp parsing etc., have been fixed. The random crashes were most likely due to incorrect parameters to filp_open() of disk persistence file and filesystem being mounted as read-only.

Version 14.9.9 release tagged on 9 September 2014

Version 15.1.8 release tagged on 8 January 2015

1112. Commits as on 17 August 2015

KingCobra + VIRGO Queuing port of Linux Kernel 4.1.5 :
 - changed the REQUEST_REPLY.queue disk persisted queue path to /var/log/kingcobra/REQUEST_REPLY.queue
 - kernel built sources, object files
 - kern.log with logs for telnet request sent to VIRGO queue driver, queued in kernel work queue and handler invocation for the KingCobra service request kernel function for the popped request; disk persisted /var/log/kingcobra/REQUEST_REPLY.queue

---
1113. Commits as on 14 October 2015

AsFer Cloud Perfect Forwarding binaries are invoked through call_usermodehelper() in VIRGO queue. KingCobra commands has been updated with a clause for cloud perfect forwarding.

1114. Commits as on 15 October 2015

- Updated KingCobra module binaries and build generated sources
- kingcobra_usermode_log.txt with "not found" error from output redirection (kingcobra_commands.c). This error is due to need for absolute path. But there are "alloc_fd: slot 1 not NULL!" after fd_install() is uncommented in virgo_queue.h call_usermodehelper() code. The kern.log with these errors has been added to testlogs
- kingcobra_commands.c has been changed to invoke absolute path executable. With uncommenting of fd_install and set_ds code in virgo_queue the return code of call_usermodehelper() is 0 indicating successful invocation

---
1115. Commits as on 10 January 2016

NeuronRain KingCobra research version 2016.1.10 released.

1116. NEURONRAIN VIRGO Commits for virgo_clone()/telnet -> VIRGO cpupooling -> VIRGO Queue -> KingCobra
- as on 12 February 2016

VIRGO commit:
https://github.com/shrinivaasanka/virgo-linux-github-code/commit/72d9cfc90855719542cdb62ce40b798cc7431b3d

Commit comments:

Commits for Telnet/System Call Interface to VIRGO CPUPooling -> VIRGO Queue -> KingCobra
---
*) This was commented earlier for the past few years due to a serious kernel panic in previous kernel versions - <= 3.15.5
*) In 4.1.5 a deadlock between VIRGO CPUPooling and VIRGO queue driver init was causing following error in "use_as_kingcobra_service" clause :
  - "gave up waiting for virgo_queue init, unknown symbol push_request()"
*) To address this a new boolean flag to selectively enable and disable VIRGO Queue kernel service mode "virgo_queue_reactor_service_mode" has been added.
*) With this flag VIRGO Queue is both a kernel service driver and a standalone exporter of function symbols - push_request/pop_request
*) Incoming request data from telnet/virgo_clone() system call into cpupooling kernel service reactor pattern (virgo cpupooling listener loop) is treated as

generic string and handed over to VIRGO queue and KingCobra which publishes it.
*) This resolves a long standing deadlock above between VIRGO cpupooling "use_as_kingcobra_service" clause and VIRGO queue init.
*) This makes virgo_clone() systemcall/telnet both synchronous and asynchronous - requests from telnet client/virgo_clone() system call can be either synchronous RPC functions executed on a remote cloud node in kernelspace (or) an asynchronous invocation through "use_as_kingcobra_service" clause path to VIRGO Queue driver which enqueues the data in kernel workqueue and subsequently popped by KingCobra.
*) Above saves an additional code implementation for virgo_queue syscall paths - virgo_clone() handles, based on config selected, incoming data passed to it either as a remote procedure call or as a data that is pushed to VIRGO Queue/KingCobra pub-sub kernelspace
*) Kernel Logs and REQUEST_REPLY.queue for above commits have been added to kingcobra c-src/testlogs/

1117. Commits - KingCobra 64 bit and VIRGO Queue + KingCobra telnet requests - 17 April 2017

---

*) Rebuilt KingCobra 64bit kernel module
*) telnet requests to VIRGO64 Queueing module listener driver are serviced by KingCobra servicerequest
*) Request_Reply queue persisted for this VIRGO Queue + KingCobra routing has been committed to c-src/testlogs.
*) kern.log for this routing has been committed in VIRGO64 queueing directory
) Similar to other drivers struct socket reinterpret cast to int has been removed and has been made const in queuesvc kernel thread

---

779. (FEATURE-DONE) Commits - CVXPY implementation for Eisenberg-Gale Convex Program - 18 August 2017 - - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

---

(*) First commits for Convex Optimized Market Equilibrium Prices
(*) Imports CVXPY Convex Program solver
(*) Objective function is a logistic variant of Eisenberg-Gale Convex Program i.e uses money * log(1+e^utility) instead of money * log(utility) because of curvature error (log is error flagged as concave and logistic is convex per: http://www.cvxpy.org/en/latest/tutorial/functions/index.html#vector-matrix-functions)
(*) Formulates constraints and objective functions based on http://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf - Page 106 and Equation 5.1
(*) But, For all installed solvers ECOS, ECOS_BB, SCS, LS solved convex program prints value as None despite all constraints and objective functions being convex. Also is_dcp() prints "not a disciplined convex program". Logs in testlogs/.
(*) Obviously it should have worked. Therefore this is only a partial implementation commit.
(*) This implementation uses numpy randomly initialized arrays for Money each buyer has and per-good utility(happiness) each buyer has.
(*) Replacing money with perceived merit values translates this Market Equilibrium - Intrinsic Value versus Market Price - to Merit Equilibrium - Intrinsic Merit versus Perceived Merit. This has been already described in NeuronRain AsFer Design Documents:
	-

http://sourceforge.net/p/asfer/code/HEAD/tree/asfer-docs/AstroInferDesign.txt
and
        - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt

---
780. (FEATURE-DONE - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt) Commits - Convex Optimization - DCCP - 21 August 2017

(*) import dccp has been added
(*) DCCP is the recent advancement and generalization of DCP for convex-concave programs
(*) method='dccp' has been added as parameter to solve()
(*) Objective function has been changed to log() from logistic() - curvature is concave which is in conflict with definition of
eisenberg-gale convex program in textbooks. Reason for this contradiction is unknown.
(*) But DCCP overcomes the DCP limitation and solve() prints converged solutions for objective functions
(*) logs have been committed to testlogs/
(*) CVXOPT solver has been installed but it does not solve the Eisenberg-Gale objective function. Only SCS solver works - by default applies
KKT conditions indirectly.

1118. (FEATURE-DONE) Commits - Convex Optimization - DCCP - 22 August 2017

(*) Verbose set to True for printing Splitting Conic Solver progress information
(*) logs committed to testlogs/

1119. (FEATURE-DONE) Commits - Convex Optimization update - 29 August 2017

(*) Removed hardcoded variable values in objective and constraints
(*) In the context of pricing, ECOS Error Metrics print the matrices of market clearing prices for goods
(Reference - pages 3072 and 3073 of
https://web.stanford.edu/~boyd/papers/pdf/ecos_ecc.pdf - KKT conditions in ECOS solver)

---
778. (FEATURE-DONE) Convex Optimization - Pricing Computation - 30 August 2017 - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
---
(*) Prices of Goods/Services have been computed explicitly from Karush-Kuhn-Tucker Conditions (1,2,3 and especially 4)
(*) References:
      - Pages 106-108 of
http://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf)
      - KKT conditions and Conic Optimization-
https://arxiv.org/pdf/1312.3039.pdf
(*) logs committed to testlogs/

1120. (FEATURE-DONE) KingCobra Kernelspace Messaging Driver for 4.13.3 64-bit kernel - 24 September 2017

(*) KingCobra driver in GitHub and SourceForge at present are 32-bit based on mainline 4.1.5 kernel
(*) Both USB-md and KingCobra kernel modules are subsidiaries of VIRGO kernel
(*) There is a necessity for 64-bit version of KingCobra for interoperability to VIRGO64 64-bit kernel on mainline version 4.13.3
(*) This requires separate repository for KingCobra because of significant kernel function changes between 4.1.5 and 4.13.3 and
idiosyncrasies of 64-bit
(*) KingCobra driver has been rebuilt on 4.13.3 64-bit kernel after some changes to function prototypes and new kingcobra64 repository is
initialized with these commits
(*) KingCobra kernel sockets have been TLS-ed by kernel_setsockopt(TX_TLS) newly introduced in 4.13 kernel.
(*) After this complete request-reply traffic from VIRGO64 system calls to VIRGO64 queueing and KingCobra is encrypted.

1121. (FEATURE-DONE) Commits - telnet - VIRGO64Queue - KingCobra64 - 25 September 2017

(*) Disk persisted KingCobra64 REQUEST-REPLY Queue written by VIRGO64 Queue to KingCobra64 telnet invocation after 4.13.3 64-bit KTLS upgrade
has been committed

1122. (FEATURE-DONE) VIRGO64 Queueing Kernel Module Listener - KingCobra64 - 4.13.3 - 6 October 2017

(*) telnet client connection to VIRGO64 Queue and a subsequent workqueue routing (pub/sub) to KingCobra64 has been tested on 4.13.3
(*) TX_TLS socket option has not been disabled and is a no-op because it has no effect on the socket.
(*) REQUEST_REPLY.queue for this routing from VIRGO64 queue and persisted by KingCobra64 has been committed to KingCobra64 repositories in GitHub and SourceForge

---
777. (FEATURE-DONE) KingCobra64 Neuro Electronic Currency transactional cloud move - Perfect Forward - 17 January 2018 - this section is an extended draft on respective topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
---
(#) Neuro Currency cloud perfect forward has been made transactional by wrapping it by Python Transaction Manager (widely used in Zope
Python Application Server)
(#) imports transaction python package and invokes begin() and commit() on subprocess call to neuro cloud move client and server

---
---

776. (FEATURE) Concurrent Managed Workqueue(CMWQ), VIRGO64 Queueing and KingCobra64 messaging - 12 June 2019 - this section is an extended draft on respective IoT messaging and kernel analytics topics in NeuronRain AstroInfer Design - https://github.com/shrinivaasanka/asfer-github-code/blob/master/asfer-docs/AstroInferDesign.txt
---
---
1. Existing workqueue underneath VIRGO64 queueing and requests routed by it to KingCobra64 messaging are old legacy workqueues which
have been revamped to Concurrent Managed Workqueue which supports concurrent messaging and lot of other options in queue creation.
2. create_workqueue() in VIRGO64 Queueing has been changed to alloc_workqueue() of Concurrent Managed Workqueue.
3. VIRGO64 Queueing request routing to KingCobra64 messaging has been tested with CMWQ and queueing log and kingcobra64 Request-Reply
Queue have been committed to respective testlogs of the drivers
4. reading from stream has been disabled in virgo_kernel_analytics.h
5. Reference - CMWQ documentation - https://www.kernel.org/doc/html/v4.11/core-api/workqueue.html
6. Byzantine Fault Tolerance in KingCobra64 persisted queue can be made available by performant CMWQ and routing to the Replicas of
REQUEST_REPLY.queue by any of the practical BFT protocols available.
7. Most important application of CMWQ based VIRGO64-KingCobra64 is in the context of kernelspace hardware messaging in IoT,Drones and other
analytics driven embedded systems.
8. An example usecase which is a mix of sync and async I/O in kernelspace:
        (*) Analytics Variables computed by userspace machine learning are read over socket stream by kernel_analytics driver and
exported kernelwide
        (*) Some interested Drone driver in kernel (example PXRC) reads the analytics variables synchronously and sends reply messages asynchronously to VIRGO64 Queuing driver over kernel sockets.
        (*) VIRGO Queuing routes the queued messages to KingCobra64 driver