```
#-------------------------------------------------------------------------------
-----------------------
#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/>.
#-------------------------------------------------------------------------------
-------------------------
#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
#-------------------------------------------------------------------------------
-------------------------------------------------

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) 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++. [Long term: 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]

(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:

```
                x1--------x2--------x3------x4
                |       |       |     |
                t         h     -       t
```

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.

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. It is more apt to say there are no conclusions arrived at.

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.

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.

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 EXP-complete. Each voter in RHS of P(Good) has such a TQBF that is input to Majority circuit.

(*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)

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".

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) = Sigma_S(fouriercoeff(S)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

$$\cos(b\log q) + i\sin(b\log q) = -1/q^a$$

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 many 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

##
##

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 - Thomas 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..."

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]. 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

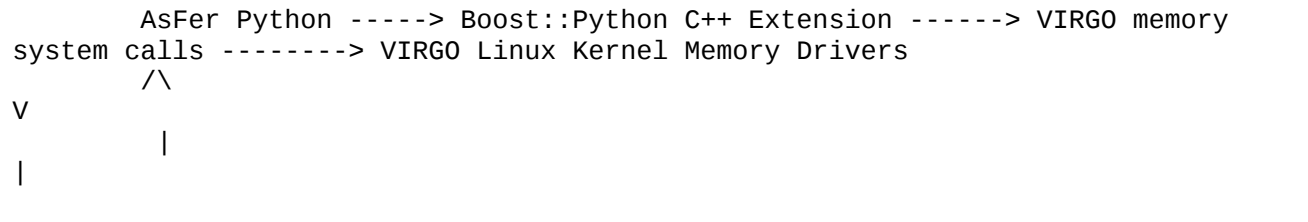
62. (THEORY) Thus correlation of the two sets *_Observed_* and *_Classic_* (each set has 2 subsets) for percentage similarity using any known similarity coefficient would unravel any cryptic pattern hidden astronomical datasets for that event and would be a circumstantial and scientific proof of rules 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 ONGOING) 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 QuasiPolynomial 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/ .



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 - ONGOING) 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

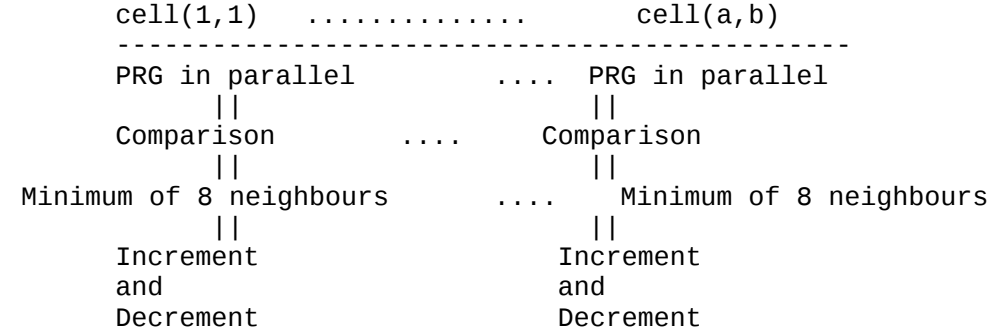
maximize X
subject to AX <= some maximum limit
and X >= 0

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 radii and objective function maximizes the radii variables.

(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/BigDataStorageAbstractionInAsFer.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

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> </>

---

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

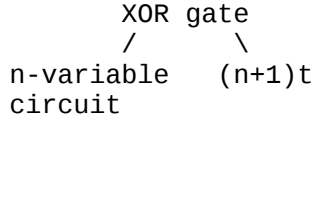
---

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.

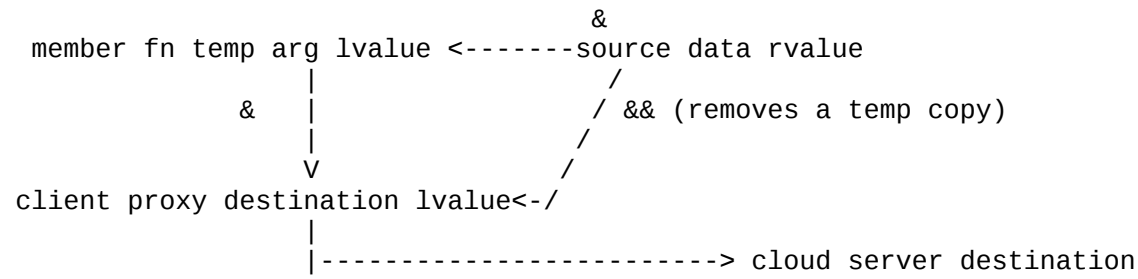
---
164. (FEATURE - DONE) Commits as on 16 September 2015
---
cpp-src/cloud_move - Implementation of Move Semantics for Cloud objects:

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) Related to 65 - Debug Analytics (as part of Software Analytics)

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.

References:
166.1 Learning Finite State Machines -
https://www.cs.upc.edu/~bballe/other/phdthesis.pdf

---
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. [QuasiPTIME algorithm for GI - Lazlo Babai - http://jeremykun.com/2015/11/12/a-quasipolynomial-time-algorithm-for-graph-isomorphism-the-details/]

---

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.

---

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.

---
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

---
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, 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.

---

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

---
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}

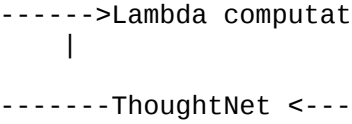
From Bose-Einstein condensate fitness model, ability of a vertex to attract links increases with decreasing energy of a vertex. This implies boundary(S) expands with presence of low energy vertices. Thus Bose-Einstein condensate model translates into an edge expander construction. Also high expansion 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 and 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 and 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

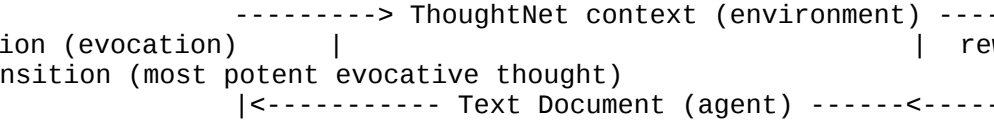
been included
	- Number of Backpropagation weight update iterations has been increased 10 fold to 3000000.
	- logs for some iterations have been included in testlogs/
- DeepLearning Convolution Network implementation:
	- Image patterns have been changed to 0, 8 and no-pattern.
	- Final neuron layer weights 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.CythonOptimized.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.CythonOptimized.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.CythonOptimized2.16March2016.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.CythonOptimized2.16March2016.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.CythonOpt
imized3.16March2016.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.CythonOpt
imized.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.CythonOpt
imized2.17March2016.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.CythonOpt
imized.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, 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, 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, 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, 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, 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, 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, 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, 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, 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]
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 ----|
                                                          |
                         V
                                                            V
                               |
                  Neo4j Graph DB<-------ThoughtNet_Neo4j.py <-----
ThoughtNet_HyperGraph_Generated.txt <-----------------------------<
                                                          |
                                                          V
                  Observations---------------->
DeepLearning_ReinforcementLearningMonteCarlo.py ---------> Evocations

--------------------------------------------------------------------------------
-----------------------
```

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.