PROTEIN STRUCTURE SIMILARITY AND VISUALIZATION

CMPS 561-FALL 2014

PROJECT REPORT

AVINASH REDDY AITHA (axa9979)

RAVI KIRAN RACHHA (rxr8118)

PROJECT GUIDE: MS. SUMI (sxs5729)

AVINASH REDDY AITHA (axa9979)
RAVI KIRAN RACHHA (rxr8118)

CONTENTS

AVINASH REDDY AITHA (axa9979)
RAVI KIRAN RACHHA (rxr8118)

1. INTRODUCTION

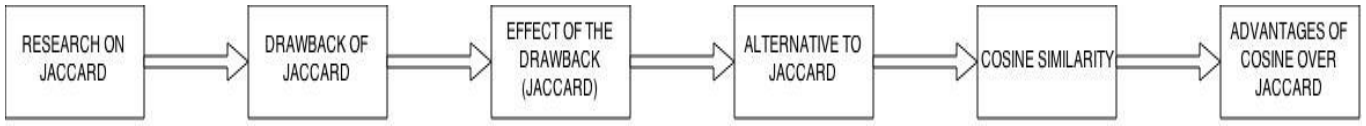
1.1 OBJECTIVE:

- Implementation of parallel algorithm to calculate pairwise similarity by comparing key and their count values between all files.
- Decrease the computation time in parallel implementation on compared with serial implementation.
- Construct lower triangle matrix with pairwise similarity values.
- To construct a similarity graph with proteins as nodes and similarity values as weight of the connection between the nodes.

Status of project: New project with Parallel implementation of Cosine and Jaccard similarity functions.

2. METHOD:

2.1 PROJECT STAGES:



- We worked on jaccard by reading various journals and web sites which are included in the references.
- Upon the various observations, we have concluded that the jaccard has few drawbacks which may lead to the error in the similarity calculation. They are mentioned below
    - It doesn’t consider term frequency (how many times a term occur in a document).
    - There is no difference between most frequent terms and less frequent terms.
    - It is easy to compute but not effective to determine the exact similarity between the files.
- Then we have started looking for various other similarity functions which will give the most accurate similarity value. We have find few of them which are mentioned below
    - Cosine Similarity
    - Jaccard Tanimoto
- We have opted cosine similarity as alternative one because it has less calculation part compared to jaccard tanimoto.

- Cosine overcomes the issue raised in Jaccard i.e term frequency is considered in similarity computation.
- Computation time is more in Cosine but much effective in evaluation when compared with Jaccard.

Jaccard:

Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

(If A and B are both empty, we define J (A, B) =1)

0<= J (A, B) < =1

Cosine:

Cosine similarity is used to measure the similarity between two or more documents using the formula mentioned below.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

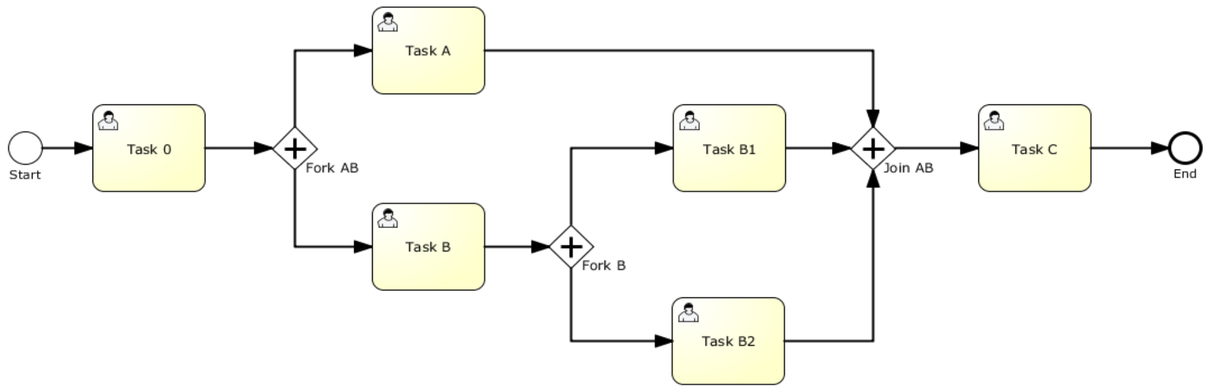
Where A, B are the two documents considered.

A.B is the dot product of two documents.

||A|| ||B|| are the magnitudes of documents A and B respectively.

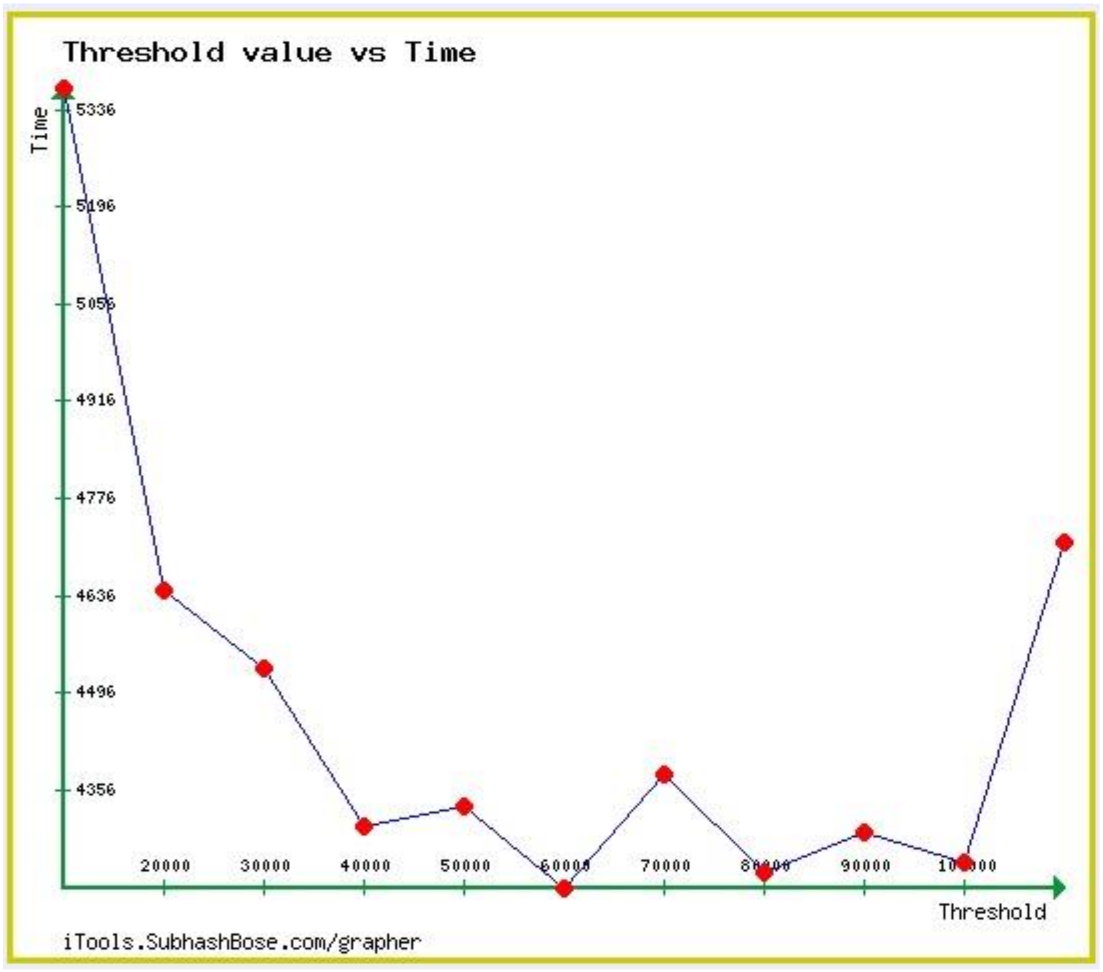
2.2 Approach:

- Similarity can be computed by either serial implementation or parallel implementation.
- Serial implementation resulting in the more computation time. In order to decrease the computation time, parallel approach is implemented.
- Parallel approach is implemented by the concept Multi-Threading in java.
- Multi-Threading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer.
- Since each input text file consists of many keys, we split the each file into blocks of keys by using the concept FORK and JOIN.
- In the implementation of parallel processing, we split the file into blocks of keys i.e pool of threads using the concept of FORK and compute the similarity for each thread.
- The blocks of keys which got split by the method FORK after executing each thread need to be combined back to form a single thread, this is done by using the method JOIN.



How threads get divided or distributed?

- The threads get divided by the threshold value which is included in the program.
- As the given key files consists of nearly 100000+ files it is tough to manage all the data at a time. Inorder to manage the keys we divide the total file into the block of keys with the help of threshold value. For example if the threshold value is 20000 then the file get divide into 5-6 block of keys, where each block consists of 20000 keys. Therefore each thread executes one block of keys at a time.
- The given graph gives the change in time with change in threshold value for same set of sample data.

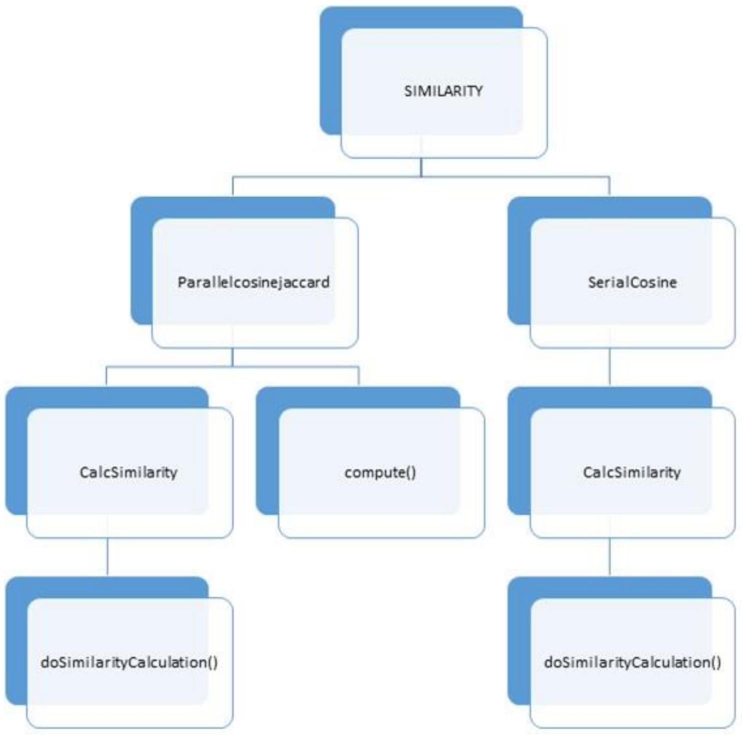


Threshold value vs Time graph

2.3 Tools & languages used:

- Netbeans IDE 8.0.1
- Gephi 0.8.2
- JDK 1.7

Flow chart for how functions are invoked:



- Similarity

  Execution of project starts from this Class. Objects from different classes are invoked to print the similarity between each and every file.

- Parallel Cosine

  Input for the project is taken from class Parallel Cosine and Output of Parallel Cosine is calculated and send to output files. This class comprises of two methods, compute ( ) and computeDirectly ( ).

  Method compute ( ) has two operations to be done in multi-threading. Fork and Join. Depending on the Threshold value, the input file is split into blocks of keys and computed for similarity.

  Method computeDirectly( ) continues with serial implementation for calculating the similarity.

  Logics for both Cosine parallel and Jaccard are implemented in this class and lower triangle matrix is printed with similarity values.

  Similarity values are passed to output files which are in .csv format to

  construct similarity graph using visualization tool.

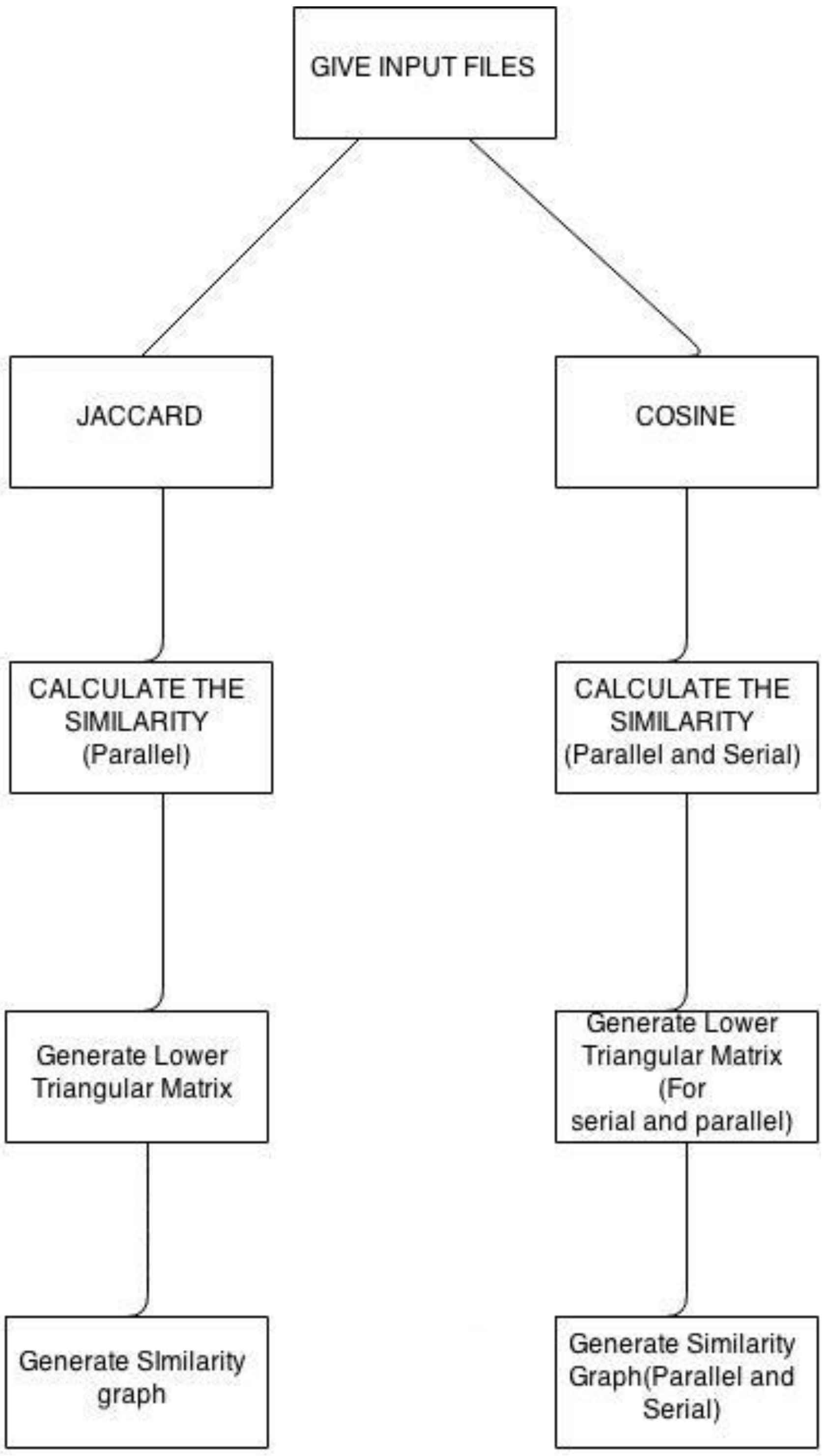
  - Serial Cosine

  Class Serial Cosine calculates the similarity and has serial implementation, sends the similarity values to output files to print only serial cosine values.

  - Calc Similarity

  In order to compare each and every file, iteration of files is run through this class.

AVINASH REDDY AITHA (axa9979)
RAVI KIRAN RACHHA (rxr8118)


GIVE INPUT FILES
JACCARD
COSINE
CALCULATE THE SIMILARITY (Parallel)
CALCULATE THE SIMILARITY (Parallel and Serial)
Generate Lower Triangular Matrix
Generate Lower Triangular Matrix (For serial and parallel)
Generate SImilarity graph
Generate Similarity Graph(Parallel and Serial)

Project Architecture

3. RESULTS:

The results displayed can be considered as the test cases for verifying the results. These results contain only few comparisons of the files considered. Remaining comparisons will be attached with the report separately for all inputs considered for the project.

a) Pairwise similarity calculated between the following Input files:

```
alpha_1_1_0_d1x3ka_.keys compared to alpha_1_1_0_d1x3ka_.keys
----------------------------------------------------------------------------------
Total comparison time for cosine in parallel is: 368 ms.
Similarity using Cosine parallel approach is: 1.0
Total comparison time for cosine in serial is: 78 ms
Similarity using Cosine serial approach is: 1.0
```

AVINASH REDDY AITHA (axa9979)
RAVI KIRAN RACHHA (rxr8118)

alpha_1_1_0_d1x3ka_.keys compared to alpha_1_1_0_d1x46a_.keys

----

Total comparison time using Parallel approach is: 143 ms

Similarity using Cosine parallel approach is: 0.6976726209108655

Total comparison time for cosine in serial is :163 ms

Similarity using Cosine serial approach is: 0.6976726209108655

alpha_1_1_0_d1x3ka_.keys compared to alpha_1_1_0_d2bk9a_.keys

----

Total comparison time using Parallel approach is: 117ms

Similarity using Cosine parallel approach is: 0.7078290154402871

Total comparison time for cosine in serial is: 247ms

Similarity using Cosine serial approach is: 0.7078290154402871

alpha_1_1_0_d1x3ka_.keys compared to alpha_1_1_0_d2c0ka_.keys

----

Total comparison time using Parallel approach is: 114ms

Similarity using Cosine parallel approach is: 0.655218790261514

Total comparison time for cosine in serial is: 329ms

Similarity using Cosine serial approach is: 0.655218790261514

```
alpha_1_1_0_d1x3ka_.keys compared to alpha_1_1_0_d2wtga_.keys
--------------------------------------------------------------------------------------
Total comparison time using Parallel approach is:  112ms
Similarity using Cosine parallel approach is: 0.7514734903175345
Total comparison time for cosine in serial is: 492ms
Similarity using Cosine serial approach is: 0.7514734903175345
```

b).Pairwise similarity calculated for other set of inputs:

```
d1a7.allKeys compared to d1a7.allKeys
-------------------------------------------------------
Total comparison time using Cosine Parallel approach is:  258ms
Similarity using Cosine parallel approach is: 1.0
Total comparison time for cosine in serial is: 71ms
Similarity using Cosine serial approach is: 1.0
```

d1a7.allKeys compared to d1au.allKeys

Total comparison time using Cosine Parallel approach is: 125 ms

Similarity using Cosine parallel approach is: 0.6844801772424071

Total comparison time for cosine in serial is: 143ms

Similarity using Cosine serial approach is: 0.6844801772424071

d1a7.allKeys compared to d1dl.allKeys

Total comparison time using Cosine Parallel approach is: 131 ms

Similarity using Cosine parallel approach is: 0.55766122666384

Total comparison time for cosine in serial is: 213ms

Similarity using Cosine serial approach is: 0.55766122666384

d1a7.allKeys compared to d1h8.allKeys

Total comparison time using Cosine Parallel approach is: 100ms

Similarity using Cosine parallel approach is: 0.6470141081924704

Total comparison time for cosine in serial is: 281ms

```
Similarity using Cosine serial approach is: 0.6470141081924704


d1a7.allKeys compared to d1lt.allKeys

-----------------------------------------------------

Total comparison time using Cosine Parallel approach is: 97ms

Similarity using Cosine parallel approach is: 0.6878779548106739

Total comparison time for cosine in serial is: 349ms

Similarity using Cosine serial approach is: 0.6878779548106739


d1a7.allKeys compared to d1q3.allKeys

-----------------------------------------------------

Total comparison time using Cosine Parallel approach is: 99ms

Similarity using Cosine parallel approach is: 0.6886773089365779

Total comparison time for cosine in serial is: 416ms

Similarity using Cosine serial approach is: 0.6886773089365779


d1au.allKeys compared to d1a7.allKeys

-----------------------------------------------------

Total comparison time using Cosine Parallel approach is: 101ms

Similarity using Cosine parallel approach is: 0.6844801772424071
```

```
Total comparison time for cosine in serial is: 69ms

Similarity using Cosine serial approach is: 0.6844801772424071


d1au.allKeys compared to d1au.allKeys

-----------------------------------------------------

Total comparison time using Cosine Parallel approach is: 53ms

Similarity using Cosine parallel approach is: 1.0

Total comparison time for cosine in serial is: 109ms

Similarity using Cosine serial approach is: 1.0


d1au.allKeys compared to d1dl.allKeys

-----------------------------------------------------

Total comparison time using Cosine Parallel approach is: 101ms

Similarity using Cosine parallel approach is: 0.794360033391251

Total comparison time for cosine in serial is: 178ms

Similarity using Cosine serial approach is: 0.794360033391251


d1au.allKeys compared to d1h8.allKeys

-----------------------------------------------------

Total comparison time using Cosine Parallel approach is: 245 ms
```

```
Similarity using Cosine parallel approach is: 0.8652887227762882

Total comparison time for cosine in serial is: ms

Similarity using Cosine serial approach is: 0.8652887227762882


d1au.allKeys compared to d1q3.allKeys

------------------------------------------------------

Total comparison time using Cosine Parallel approach is: 102 ms

Similarity using Cosine parallel approach is: 0.8209465006176231

Total comparison time for cosine in serial is: 380ms

Similarity using Cosine serial approach is: 0.8209465006176231


d1au.allKeys compared to d1lt.allKeys

------------------------------------------------------

Total comparison time using Cosine Parallel approach is: 105 ms

Similarity using Cosine parallel approach is: 0.7924887868315987

Total comparison time for cosine in serial is: 314ms

Similarity using Cosine serial approach is: 0.7924887868315987
```

```
d1au.allKeys compared to d1lt.allKeys

------------------------------------------------------

Total comparison time using Cosine Parallel approach is: 105 ms

Similarity using Cosine parallel approach is: 0.7924887868315987

Total comparison time for cosine in serial is: 314ms

Similarity using Cosine serial approach is: 0.7924887868315987
```

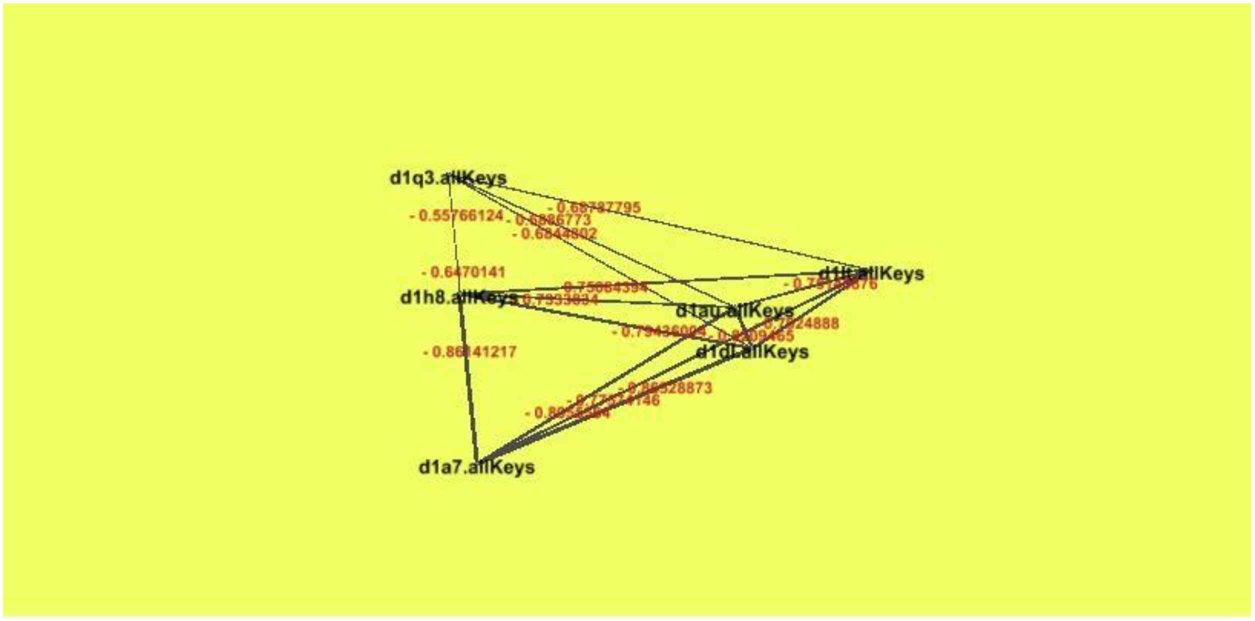
NOTE: Outputs of all files are not shown above, the complete output files of the given outputs will be attached with the report (Softcopy).

Lower Triangle matrix obtained for considered a set of 6 input files is:

```
Matrix for cosine similarity

============================

1.000000000000   0.000000000000   0.000000000000   0.000000000000   0.000000000000   0.000000000000


0.684480177242   1.000000000000   0.000000000000   0.000000000000   0.000000000000   0.000000000000


0.557661226664   0.794360033391   1.000000000000   0.000000000000   0.000000000000   0.000000000000


0.647014108192   0.865288722776   0.861412158981   1.000000000000   0.000000000000   0.000000000000


0.687877954811   0.792488786832   0.750843941955   0.77574144236   1.000000000000   0.000000000000


0.688677308937   0.820946500618   0.733383404961   0.805556412099   0.751458762221   1.000000000000
```

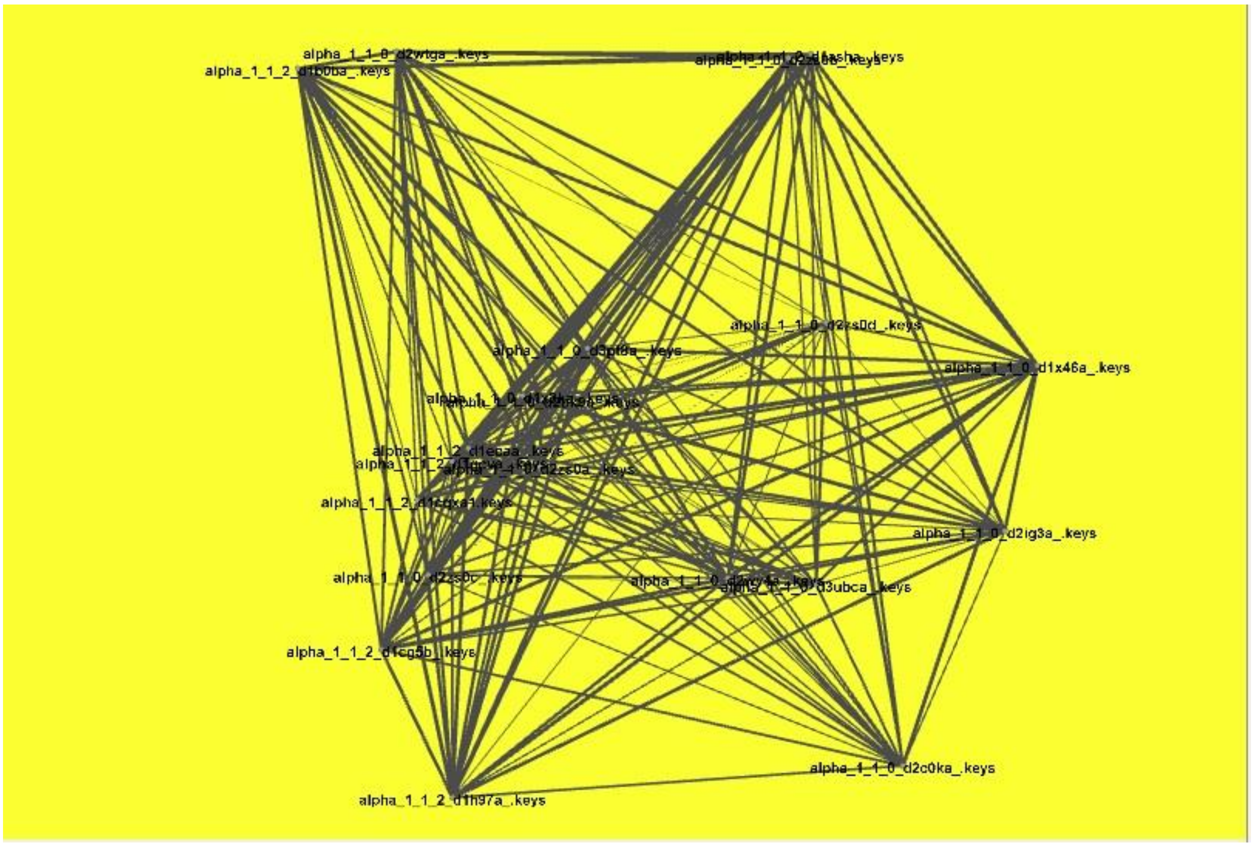
AVINASH REDDY AITHA (axa9979)
RAVI KIRAN RACHHA (rxr8118)

Similarity graph constructed for the similarity values of 6 files considered:



AVINASH REDDY AITHA (axa9979)
RAVI KIRAN RACHHA (rxr8118)

Similarity graph constructed for the similarity values of 20 files considered:



4. References:

Documents:

- Authors: Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze

  Title: An Introduction to Information Retrieval

  Edition: Online edition 2009, April 1

  Publisher: Cambridge University Press

- Author: Herbert Schildt Java:

  Title: Java: The Complete Reference

  Edition: Ninth

  Publisher: Oracle Press

- All class notes of Professor Dr. Vijay V Raghavan

URL:

Information regarding Cosine and Jaccard functions

http://csis.pace.edu/ctappert/dps/d861-12/session4-p2.pdf

http://en.wikipedia.org/wiki/Jaccard_index

http://en.wikipedia.org/wiki/Cosine_similarity

http://www.ibm.com/developerworks/library/j-jtp0730/

http://www.softwareengineeringsolutions.com/blogs/2010/07/21/implementing-thread-pools-using-java-executors/

http://en.wikipedia.org/wiki/Triangular_matrix

http://en.wikipedia.org/wiki/Gephi

6. APPENDIX

Here the basic source code of dividing the keys into set of blocks is done. The below given is a pseudo code

```
protected Integer compute() {

                final int length = end - start;
                if (length < SEQUENTIAL_THRESHOLD) {
        //here SEQUENTIAL_THRESHOLD=20000
                        return computeDirectly();
                }
                final int split = length / 2;
                final ParallelCosineJaccard left = new ParallelCosineJaccard(data1, start,
start + split,basefile,compareto);// left contains 20000 keys
                left.fork();
                final ParallelCosineJaccard right = new ParallelCosineJaccard(data1, start
+ split, end,basefile,compareto);// right contains rest of the keys
                left.join();
                right.compute();



                return null;

        }
```

Explanation (Dividing the total file into set of blocks):

The given compute() function divide the given file into a set of keys where the total file is divide into *left* and *right* part. Where left part consists of the 20000 (SEQUENTIAL_THRESHOLD).

And the right part contains the rest of the keys. And the right part of the keys are again sent as a parameters to the *compute()*. And the same process is continued and the each block consists of 20000 keys.