EFFICIENT COMPUTATION OF SIMRANK FOR STATIC AND DYNAMIC DATASETS USING MAPREDUCE FRAMEWORK

Soujanya Duvvi*1, Venkata Ramana Kondapalli2

*1 Department of CS&SE, Andhra University, Visakhapatnam,A.P., India
soujanya.297@gmail.com

2 Assistant Professor, Dept. of CS&SE, Andhra University, Visakhapatnam,A.P., India
kvramana.auce@gmail.com

Abstract: The growth of data dynamically over the internet and on day to day basis and the need to store and access information efficiently brings up new challenges of finding related documents, similar nodes, domain & inter-domain similarities etc. Though SimRank is applicable to wide range of areas, we use this similarity ranking to find similarity between neighbours in a contextual way and evaluate in a numerical way. Here we use Jaccard Similarly for calculating similarity by using LSH and various other methods. We further optimize the Jaccard Algorithm by using Token Optimization join method. The obtained result is further evaluated with a combination of four other parameters and from the result obtained the similarity values of nodes that are greater than the optimal threshold value φ are retrieved from the huge graph.

Keywords: Jaccard Algorithm, Context Similarity, SimRank, Ranking Coefficient, MapReduce

INTRODUCTION

In today’s world everything and anything can be represented in the form of a set of interconnections between various kinds of objects which assure some kind of similarity between them. These interconnections (relations) and the objects together can be termed as a network(graph). A graph is an ordered pair \( G = (V;E) \) comprising a set of \( V \) vertices (nodes) together with a set of \( E \) edges (links), which are two element subsets of \( V \). These vertices can be categorized into two types:

1. Platonic Vertices -Comprising of Persons, Institutions, Artifacts and concepts
2. Affiliation Vertices -These bind the Platonic Vertices in a flexible way.

This type of modeling is associated in every field including semantic web, networks, bibliographic networks, social networks, citation networks, Link prediction etc. Due to the drastic growth of information in the recent times, information storage and retrieval is becoming a big hurdle for everyone especially the storage capabilities are becoming bottle necks for the administrators and network analysts. In the recent years enormous work is being done on how to explore the various networks depending on parameters like object similarity, link similarity, hyperlink analysis, contextual similarity and so on.

Considering one parameter and analyzing the graph, retrieving the data did not prove to be very efficient putting up constraints like space and time complexities, dynamic growth of graphs, duplication of data, redundant information collections, dangling or the sink nodes etc. To maximize the efficient accessing of data one should know how to explore the huge network structure so as to get the required information in an optimum way. For effective accessing of data in a graph the amount of similarity that is being existed between the nodes should be analyzed before hand. Here we propose a 2 way ranking system to maximize the efficiency of retrieval, of useful information in an optimum way, with less complexities, in a dynamic graph.

Here we consider contextual similarity as an initial step for finding the similarity. And later a 5 parameter evaluation will be done on the graph retrieved after the initial classification where the nodes are selected depending the optimum value of similarity between them. The value of similarity range from 0 to 1, 0 indicating least similarity or no similarity where as 1 indicating the maximum similarity. In domains such as social link analysis, citation networks etc deal with very large datasets, also called Big Data. In such scenarios, single machine cannot easily handle the entire data to produce effective results.

We adapt a new programming model and an associated implementation for processing large data sets with a parallel, distributed algorithm on a cluster called the MapReduce. MapReduce emerged as a new paradigm to query data in distributed systems which was initially proposed by Google and further an open source implementation is provided by the Apache group and named it Hadoop. Due to the large-scale nature of the data and the computationally intensive tasks, this problem is tackled using the Map Reduce programming paradigm.

A MapReduce program is composed of a Map() procedure that performs filtering and sorting of data and a Reduce() procedure that performs a summary operation like grouping of data broken into tokens by Map() procedure. Hadoop is
an open source framework that provides both reliability and
data motion for applications, which is also a scalable fault
tolerant distributed system for large scale data storage and
processing. Here we consider an example of a citation graph

![Citation Network](image)

where each node represent a domain of research papers
where DB is DataBase, IR is Information Retrieval, AI is
Artificial Intelligence, ML is Machine Learning. An edge
between 2 nodes indicates that a paper in one node cites a
paper in the other. Here apart from considering the
similarity between the nodes of similar type, inter-
disciplinary similarity is also considered. A paper from ML
can cite a paper in IR or DB which are being ignored in
other cases.

There are many other criteria that should be taken into
consideration for ranking the nodes which can distinguish
the nodes which are perceived as more and less meaningful,
more and less distant, more and less trusted, etc. The
ranking score assigned to a particular node is defined as a
function of these parameters. Furthermore different weights
can be given to different parameters according to users' preferences (e.g., trust could be given more weight than others). Some of the criteria include:
- Trust Values
- Rarity
- Popularity
- Association Length
- Context Definition
- Subsumption

In this paper we consider Jaccard similarity for calculating
the context similarity and then consider the first five
parameters in the above stated list.

**Related Work**

The related work can be categorized as follows.

Contextual Similarity measures compute similarities among
objects by comparing the contents of the objects involved,
such as texts and multimedia. Measures for computing the
similarities among objects are Cosine similarity, SVD,
LDA, LSI-based similarity measures etc. In a text-based
similarity measure, the similarity between two objects
becomes higher in general when the two objects have more
words in common leaving behind all the other parameters.

Link Based Similarity measures represent the relationships
among objects as links and compute the similarities using
the link information. The more neighbors two objects have
in common, the higher the similarity between the two. Link-
based measurements include Co-citation, Bibliographic
coupling, SimRank, and P-Rank.

Popularity Ranking assigns higher values to the nodes that are referenced maximum which can also referred
to be as hotspots in general terminology. This kind of
approach is almost similar to Google's PageRank but it also
considers the incoming and outgoing links where as in
PageRank it depends only on Out-going links.

Hyperlink Analysis is one of the most important
measurements for calculating the SimRank, which
recursively follows the policy that "2 nodes are similar if
they are referenced by similar nodes". The links are assigned
from source to destination where source node references the
destination node. It uses a recursive formula for calculating
the SimRank depending on the depth of the graph.

In this paper we propose a better type of ranking system. In
the first level Jaccard Similarity will be calculated over the
network and the various parameters like association length,
popularity, Context Similarity and Rarity of the nodes in the
graph are considered for ranking the nodes with higher
similarity.

**Organization of the rest of the paper**

The rest of the paper is organized as follows. Section 2 defines the problem to be addressed in this paper
and the solution developed for the problem. Section 3 describes the details of the similarity ranking and the
measures adopted for assigning a better ranking to the
nodes. Section 4 provides the Experimental Evaluation of
the proposed solution. And also effective optimization
techniques. Sections 5 discusses the Conclusion and Future
Work followed by the References in Section 6.

**METODOLOGY**

In this section, the definition of the problem will be
addressed and also the terminologies used in the paper are
listed out. An overview of the solution is also being
discussed in this section. Table 1 lists the frequently used
terminologies in this paper.

**Problem Definition**

A Citation graph G represented as G(V,E) where V is set
of vertices and E is set of edges joining them. Here our
main focus is on directed graphs.

**Definition 1:**

Given two vertices V_i and V_{i+1} in graph G, the measure
of similarity between can be defined as
\[ CS(V_i, V_{i+1}) = k_1 \cdot S + k_2 \cdot T + k_3 \cdot R + k_4 \cdot P + k_5 \cdot L \]
where S is the Jaccard Similarity measure represented as
\[ J(A, B) = \frac{|A \cap B|}{|A \cup B|} \]
where A and B are the set of elements
associated with the nodes V_i, V_{i+1}.

**Definition 2:**

Jaccard similarity is defined as the cardinality of their inter-
section divided by the cardinality of their union.

**Problem Definition :** Given two sets of elements S_i and S_{i+1}
and an optimum threshold value \( \theta \), our algorithm gives the
vertex pairs \((V_i, V_{i+1})\) whose similarity value is greater than
or equal to \( \theta \) and also whose contextual similarity value is
not less than the threshold \( \varphi \).

**Overview of Solution**
The Jaccard Similarity between two sets A and B is defined as the ratio between the size of their intersection divided by the size of the union. To calculate the Jaccard similarity between the input documents we first use a technique called hashing. Which is very useful to check if there are any duplicates of the documents. If the hash values are equal then the documents are similar, but if the hash values differ they are near duplicates or may be completely unequal.

Here we employ a byte-by-byte comparison of the two inputs. To determine whether they are near duplicates or different we use a special technique similar to hashing called "Locality Sensitive Hashing". The probability that two documents have the same hash outputs is equal the degree of resemblance. For doing this we take into consideration concept of shingling.

Then we consider another technique called minhashing. A hash function H() can be defined as "A function that takes shingles as input and results the hashcode". We find the minimum of the returned hashvalues using a separate function called Hmin(). The minimum values of the 2 sets will be equal when the min value of union is also within the intersection of the sets

\[ H_{\text{min}}(A) = H_{\text{min}}(B) \] when \( H_{\text{min}}(A \cup B) \in (A \cap B) \)

The probability of the minimum hashvalues in set A and in set B matching is equivalent to Jaccard Similarity of sets

\[ \text{Prob}[H_{\text{min}}(A) = H_{\text{min}}(B)] = f(A, B). \]

The following approach in being implemented as

- Represented each document as a set of k-shingles.
- Find the minhash of all shingles in document.
- Only inspect documents that have the same minhash for further consideration.

Here we reduce the problem of checking all pairs of documents for approximate similarity. Here by reducing the time and space for clustering. This process can be done in two-phases. This can be implemented using map and reduce methods.

This can be done in two passes. This can be implemented using map and reduce methods.

**Pass 1:**
Map: Finds z minhashes over k-shingles in each documents
Input: Text files in the format DOCID - CONTENT
Output: < "MINIMUM HASH VALUES", "DOCID - CONTENT" >
Reduce: Aggregates all documents with z identical hashes, also choosing a summary document for the cluster
Input: < "MINIMUM HASH VALUES", "DOCID - CONTENT" >
Output: < "MINIMUM HASH VALUES", "DOC1.DOC2....DOCN Summary Text" >

**Pass 2:**
Map: Echo the values with a common key
Input: < "MINIMUM HASH V ALUES", "DOC1.DOC2....DOCN Summary Text" >
Output: < "Key", "DOC1.DOC2....DOCN Summary Text" >
Reduce: Parse all lines into instances, which represent a cluster

An empty intersection between token sets indicates zero similarity between the corresponding objects. The similarity threshold in joins is always larger than zero, otherwise the join predicate would be true for all pairs resulting in a cross product of the input tables. Pairs of objects with an empty intersection on their token sets will never be in the join result. The sets are broken and expressed as an equality join on tokens. The algorithm mainly consists of three steps

- Hash - Produces a new table for each relation R and S which contains an intersection of tokens between the sets. A tuple consists of Hash codes, an object identifier and the respective token set.
- Join - Computes the equality join(\( R \bowtie S \))on tokens, and produce a tuple (hashcodes, DOC1......DOCN, SUMMARY TEXT) for every token that R and S have in common.
- Group - The partitioned results are joined together depending on the id’s. The number of tuples in the group is the intersection between the respective token sets R and S.

The hash step performs the hash operation on the shingles of every document. Then the set of hash values along with document-id and content of document are sent to the join step, where all the similar hash values are joined as key and all the joined document-id and summary text forming the value. These values are sent to the group step, where all the clusters above a specified threshold value are kept in the table and remaining are left out, giving the document-id and summary text as output.

Consider an example

<table>
<thead>
<tr>
<th>Table – 1 : Input Relation R</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Id</strong></td>
</tr>
<tr>
<td>32308</td>
</tr>
<tr>
<td>523015</td>
</tr>
<tr>
<td>301</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table – 2 : Input Relation S</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Id</strong></td>
</tr>
<tr>
<td>323015</td>
</tr>
</tbody>
</table>

Input: < "Key","DOC1.DOC2.......DOCN Summary Text" >
Output: Combine all clusters that contain a specific ratio.

**Similarity Ranking**
In this section the solution for the above mentioned problem is brought up through the extension for the Jaccard Similarity measure by using the optimization technique Token Based Filtering - Token Equality Joins. Here we consider joins between two relations R and S with attributes .id and tokens, where id is an identifier of some object and tokens is the set of tokens for the object identified by id . The join predicate is an overlap constraint on the tokens attribute, which represents the similarity between objects. This join is also known as set similarity join, which computes intersections between pairs of token sets and returns only the pairs that meet the overlap constraint.
### Table 3: Hash Relation R

<table>
<thead>
<tr>
<th>Hash values</th>
<th>Id</th>
<th>Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>11501001,20975228,3</td>
<td>32308</td>
<td>Find the system easy to use hundreds of MapReduce programs have been implemented</td>
</tr>
<tr>
<td>2307267,3855622</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40957371,31215483,2</td>
<td>523015</td>
<td>Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines</td>
</tr>
<tr>
<td>7834498,7244631</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23014247,65001616,7</td>
<td>301</td>
<td>special purpose computations that process large amounts of raw data</td>
</tr>
<tr>
<td>0050776,34172098</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 4: Hash Relation S

<table>
<thead>
<tr>
<th>Hash values</th>
<th>Id</th>
<th>Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>40957371,3</td>
<td>323015</td>
<td>Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines</td>
</tr>
<tr>
<td>31215483,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27834498,9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7244631</td>
<td>88901</td>
<td>crawled per host these to of most frequent queries in a appear in OSDI 2004</td>
</tr>
<tr>
<td>8544975,7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1816800,8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52519754,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10618067</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40957371,3</td>
<td>523015</td>
<td>Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines</td>
</tr>
<tr>
<td>31215483,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483,27834498,7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20975228,11501001,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11501001,20975228,3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2307267,3855622</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40957371,31215483,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7834498,7244631</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23014247,65001616,7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0050776,34172098</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 5: Join Relation (R ⊙ S)

<table>
<thead>
<tr>
<th>Hash values</th>
<th>Id</th>
<th>Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>40957371,3</td>
<td>323015</td>
<td>Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines</td>
</tr>
<tr>
<td>31215483,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27834498,9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7244631</td>
<td>88901</td>
<td>crawled per host these to of most frequent queries in a appear in OSDI 2004</td>
</tr>
<tr>
<td>8544975,7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1816800,8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52519754,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10618067</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40957371,3</td>
<td>523015</td>
<td>Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines</td>
</tr>
<tr>
<td>31215483,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>483,27834498,7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20975228,11501001,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11501001,20975228,3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2307267,3855622</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40957371,31215483,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7834498,7244631</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23014247,65001616,7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0050776,34172098</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 6: Group Relation

<table>
<thead>
<tr>
<th>Hash values</th>
<th>Id</th>
<th>Tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>40957371,3</td>
<td>523015</td>
<td>Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines</td>
</tr>
<tr>
<td>31215483,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27834498,9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7244631</td>
<td>88901</td>
<td>crawled per host these to of most frequent queries in a appear in OSDI 2004</td>
</tr>
<tr>
<td>8544975,7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1816800,8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52519754,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10618067</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32308</td>
<td>Find the system easy to use hundreds of MapReduce programs have been implemented</td>
<td></td>
</tr>
<tr>
<td>301</td>
<td>special purpose computations that process large amounts of raw data</td>
<td></td>
</tr>
</tbody>
</table>

The same procedure is applied for all the documents given in the input. For convenience, operation is shown only on 2 relations.

**Algorithm 1: Extended Jaccard Similarity**

**Input:** Two token tables X₁, X₂

**Output:** The pairs whose distance value is greater or equal to 0.5

1. \( X_3 \leftarrow \text{Hash}(\text{Tokens of } X_1) \)
2. \( X_4 \leftarrow \text{Hash}(\text{Tokens of } X_2) \)
3. \( A \leftarrow X_3 \bowtie X_4 \)
4. \( B \leftarrow \text{Group}(A) \)
5. \( \text{Overlap}(B) \leftarrow \text{SUM}(\min(X_1, X_2)) \)
6. \( D \leftarrow \left( 1 - \frac{\text{overlap}}{\text{size1}+\text{size2} - \text{overlap}} \right) \leq \varphi \)
7. Return D

The hash of the token bags are computed in the first 2 steps of the algorithm and are stored in the tables X₁ and X₄. Then the join operation is performed on the two hash relation tables and stored in Table-A in step 3. In step 4 the grouping of the joined hash relation takes place depending on the similar hash values and is stored in table B. The intersection values of the tokens are computed and stored in Table B. Suppose an element that appears \( n \) times in a relation C₁ and \( m \) times in a relation C₂ appears min of \( (n,m) \) times in the intersection \( C_1 \cap C_2 \). The intersection result is extended with the size of the Tokens and stored in Table D. Finally, all pairs of clusters with distance of \( \varphi = 0.5 \) are selected and the result relation D is returned.

**Algorithm 2: Primary Algorithm**

**Input:** Graph G, Two token tables X₁, X₂, Adjacency Matrix M

**Output:** Set of pruned pairs of similar nodes J

1. \( J,S \leftarrow \emptyset \)
2. \( T,R,P,L,W \leftarrow 0 \)
3. \( S \leftarrow \text{Extended Jaccard}(X_1, X_2) \)
4. \( (T,R,P,L) \leftarrow \text{Graph Properties}(M) \)
5. \( W \leftarrow k_1 \ast T + k_2 \ast R + k_3 \ast P + k_4 \ast L + k_5 \ast S \)
6. if \( W \geq 0.5 \)
7. \( J \leftarrow W \)
8. Return J

The step 1 initialize the sets J and S to \( \emptyset \). The steps 2 initializes parameters T,R, P, L,W to 0. Step 3 executes the function Extended Jaccard with parameters X₁,X₂ and gets the value of set S. Step 4 retrieves the values of parameters T,R,P,L by executing the function Graph Properties with parameter M. Step 5 Finds the summation of all the parameters by normalizing it. Step 6 checks if the values in...
W are greater than 0.5, the values that satisfy the condition are assigned to the set J in Step 7, Step 8 returns the set J.

Algorithm 3: GraphProperties
Input: Graph Adjacency Matrix M
Output: Trust Value T, Rarity Value R, Popularity P, Association length L
1. \( T \leftarrow \text{User Defined Value for the graph} \)
2. \( R \leftarrow \text{Least number of in – nodes} \)
3. \( P \leftarrow \text{Maximum number of out – links} \)
4. \( L \leftarrow \text{Number of outlinks from each of the nodes from source to destination} \)
5. Return T, R, P, L

The step 1 finds the number of the links between the source and the destination and then assigns it to the variable L. The step 2 gets the user defined value of trust for the graph and then assigns it to the variable T. The step 3 finds the least number of the in-links and then assigns it to the variable R. The step 4 finds the maximum number of the out-links and then assigns it to the variable P. The step 5 Returns all the values R, P, T, L.

The other parameters that are taken into consideration for ranking are as follows.

Trust
Various networks in a Semantic Association arise from different types of sources. Some of these sources may be more trusted than others. Hence, trust values need to be assigned by the user depending on its source. For our problem we use, trust values that were empirically assigned. The strength of an association is only as strong as its weakest link in its semantic association. It is considered as \( T = \min(T_i) \), where \( T_i \) is the trust value of the component.

Rarity
In a network there exist many kinds of relationships and entities. Some are rarely occurring entities and relationships and some may be frequently occurring entities and relationships. Thus depending upon the query, Rarity weight preference will be assigned to the entity. The Rarity rank of an association A, in terms of the rarity of the components within A can be determined based on the ontology.

\[
RR_t = \frac{|\text{Set of all entities}|}{|\text{Set of all entities}| - |\text{Entities of the same type}|}
\]

The overall Rarity weight, R of an association can be computed as:

\[
R = \frac{1}{\text{Length}(A)} \times \sum_{i=1}^{\text{Length}(A)} RR_t
\]

Popularity
In a network some entities have more incoming and outgoing relationships than others. Somewhat similar to PageRank algorithm used by Google, our approach takes into consideration the number incoming and outgoing relationships of entities. These entities can be thought of as hotspots. For example, authors with many publications would have high popularity. The Popularity of an association in terms of the popularity of its entities, can be defined as

\[
P_t = \max_{i \in S} \frac{|\text{Popularity of all entities}|}{|\text{Popularity of Entities of the same type}|}
\]

The overall Popularity weight P of an association can be computed as:

\[
R = \frac{1}{n} \times \sum_{i=1}^{n} P_i
\]

Association Length
In a network distance between entities may be longer some may even be shorter. Depending on the query or need, the longer or the shorter distance will be taken into account. The Association Length can be determined as:

\[
L(A) = \frac{1}{\text{Length}(A)}
\]

Overall Ranking Criterion
The overall association Rank using the before mentioned criteria, is defined as

\[
W = k_1 \ast S + k_2 \ast T + k_3 \ast R + k_4 \ast P + k_5 \ast L
\]

where \( k_i (1 \leq i \leq 5) \) add up to 1.0 and is intended to allow fine-tuning of the ranking criteria. This provides a flexible ranking approach to assess the overall relevance of associations.

Unlike taking either upper or lower bound for short listing the nodes, here we take into consideration the optimum threshold value. By considering this optimum value (i) The most consistent nodes can be retrieved (ii) Dangling nodes can be eliminated (iii) Nodes which are very less than the threshold value need not be considered as they will be of no use to graph (iv) The nodes which are very high than threshold or approx equal to 1 are already similar. It makes no sense of computing similarity to already known similar nodes.(v) By using this way space and time complexities can be reduced drastically.(vi) The computation overhead can also be reduced in this way.

RESULTS AND DISCUSSION

To validate the success of the above discussed system we evaluate it on Hadoop, an open source implementation of MapReduce. The implementation was written in Java using the Hadoop MapReduce Framework. We evaluated the implementation on a single node running Hadoop 0.21.0. The computing node is equipped with Intel i3 core Processor and 6GB RAM. The Hard Disk of 320 GB. Operating System used Linux; Languages – Java.

Citation Network dataset is a huge collection of various research papers comprising of all domains. In this network a node represents research papers of a specific domain. An edge between the nodes indicates that a paper in node 1 cites a paper in node 2. We restricted the network of papers to only four domains. There were 2500 research papers in each domain and a total of 10,000 records.
CONCLUSION

This thesis addresses the problem of how to exploit semantic relationships of named entities to improve relevance in search and ranking of documents. The method applied here is first analyzing the documents for their contextual similarity and picking the best nodes whose value is greater than the threshold value. Later the result is evaluated with other parameters and the obtained result whose result is greater than the obtained threshold value two are ranked and displayed to the user as output.

The problem mentioned above is implemented on a single node. In future we try to implement in a multinode environment and in clusters.

ACKNOWLEDGEMENTS

I profusely thanks all the staff, my parents, colleagues and friends who have extended their cooperation in various ways for completing my work. It is my pleasure to acknowledge the help of all those individuals.

REFERENCES

[9] Liangliang Cao, Hyun Duk Kim, Min-Hsuan Tsai, Brian Cho, Zhen Li and Indranil Gupta, Delta-SimRank Computing on MapReduce, BigMine’12, 2012.

D. Soujanya completed M.Tech. IT and her current interests are Data Mining, Programming in Java and Formal Languages and Automata Theory, Discrete Mathematics.

Dr. K. Venkata Ramana, Assistant Professor, Andhra University his current interests are Data Mining, Formal Languages and Automata Theory, Discrete Mathematics and Artificial Intelligence.