Delta-SimRank Computing on MapReduce

Liangliang Cao
IBM Watson Research Center
liangliang.cao@us.ibm.com

Hyun Duk Kim,
University of Illinois at Urbana-Champaign
hkim277@illinois.edu

Min-Hsuan Tsai
University of Illinois at Urbana-Champaign
mtsai2@illinois.edu

Brian Cho
University of Illinois at Urbana-Champaign
bcho2@illinois.edu

Zhen Li
University of Illinois at Urbana-Champaign
zhenli3@illinois.edu

Indranil Gupta
University of Illinois at Urbana-Champaign
indy@illinois.edu

ABSTRACT

Based on the intuition that "two objects are similar if they are related to similar objects", SimRank (proposed by Jeh and Widom in 2002) has become a famous measure to compare the similarity between two nodes using network structure. Although SimRank is applicable to a wide range of areas such as social networks, citation networks, link prediction, etc., it suffers from heavy computational complexity and space requirements. Most existing efforts to accelerate SimRank computation work only for static graphs and on single machines. This paper considers the problem of computing SimRank efficiently in a distributed system while handling dynamic networks which grow with time. We first consider an abstract model called Harmonic Field on Node-pair Graph. We use this model to derive SimRank and the proposed Delta-SimRank, which is demonstrated to fit the nature of distributed computing and can be efficiently implemented using Google's MapReduce paradigm. Delta-SimRank can effectively reduce the computational cost and can also benefit the applications with non-static network structures. Our experimental results on four real world networks show that Delta-SimRank is much more efficient than the distributed SimRank algorithm, and leads to up to 30 times speed-up in the best case[1].

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithm

Keywords

SimRank, Delta-SimRank, Distributed Computing

[1]Source codes are provided at `pikachu.ifp.uiuc.edu/~mtsai2/delta_simrank/`

BigMine'12, August 12, 2012 Beijing, China

1. INTRODUCTION

As social community websites (including Facebook, Twitter, Quora, Groupon, etc.) have become increasingly popular, it has been more and more important to measure similarities between two objects with network structures. The problem of measuring "similarity" of objects arises in many applications: to predict which user might be a potential friend, to recommend music or videos to customers, and to promote sales of products to specific user groups. To accomplish these goals, it is no longer enough to just match the subject with the limited user profiles; it is more preferable to explore the rich resource of the network structure. Intuitively, a video might be interesting to a user if he likes another movie that is similar. Two people might want to know each other if their friends are similar.

To explore the network structure to measure object similarity, Jeh and Widom [12] proposed SimRank to measure the similarity between two nodes in the network. SimRank is based on the idea that "two objects are similar if they are related to similar objects." Compared with other domain-specific measures, SimRank usually yields the best performance in network context. SimRank has been successfully used for many applications in social networks, including citation networks [17], and student-course networks [8]. It is also applied to social annotation [3], information retrieval [4], and link prediction [18].

Despite its effectiveness, SimRank is very expensive to compute in two aspects. First, the time complexity of computing SimRank is huge. It might take about 46 hours to compute the SimRank measures in a synthetic network with 10K nodes on a single machine [19]. Second, when the network is large, it will require a huge amount of memory to compute SimRank, which is often beyond the ability of a single computer. For example, for a graph with 1M nodes, several TB memory is needed to cache all the SimRank scores. Although there have been quite a few works to improve the SimRank algorithm, these works were all intended for single computers [2, 19, 17, 8, 26]. and thus are confined to single machine's computational power and memory limits.

In the era of Internet, searching engines and social community service are built on distributed architectures. Google's MapReduce [6] is a popular paradigm in these settings for large-scale computations. In this study, we aim to design an efficient algorithm to compute SimRank on MapReduce. We first present a distributed implementation of SimRank on MapReduce. We show that the use of a distributed system makes it possible to compute SimRank in large networks. Our first algorithm greatly reduces the high computational cost and memory requirement, at the cost of lots of data transfer in the distributed system. To further improve the

performance, we design a new algorithm for distributed computing. To understand our new model, we will introduce an abstract model, named Harmonic Field On the Node-pair Graph (HFONG), which provides a general model to analyze the evolvement of node-pair similarities. We introduce a special case of HFONG, named Delta-SimRank, and prove that the problem of computing SimRank in a network can be transformed to Delta-SimRank. Compared with original SimRank, Delta-SimRank fits better in the scenario of distribute computing as it leads to less communication traffic and faster speed. In addition, Delta-SimRank can be used for not only static graphs but also dynamic graphs whose nodes and edges keep increasing. Our experimental evaluation on four real datasets validates the success of Delta-SimRank on distributed systems.

Following are the contributions of this study: (1) We first implement SimRank on MapReduce and find that the bottleneck lies in huge network loads. (2) We introduce harmonic field analysis on node-pair graphs, which gives rise to the new Delta-SimRank algorithm. (3) We prove that Delta-SimRank enjoys quite a few nice properties, which lead to an efficient algorithm on MapReduce to compute SimRank scores not only on static graphs but also on graphs with increasing number of nodes and edges.

2. PROBLEM STATEMENT

SimRank [12] is a link-based similarity measure. Unlike other similarity measures such as Euclidian distance which are computed using object attributes, SimRank suggests a complementary similarity metric which is applicable in any domain with relationships between objects.

We consider a graph $G(V, E)$ that consists of a set of nodes V and a set of links E. A link from node a_1 to a_2 is denoted as $(a_1, a_2) \in E$. For a node a, we use $I(a) = \{b \in V | (b, a) \in E\}$ to denote all the nodes that have a link to a, and call these the in-neighbors of a.

In SimRank, the similarity between two nodes (or objects) a and b is defined as the average similarity between nodes linked with a and those with b. Mathematically, assigning $s(a, b)$ as the similarity value between node a and b, we are looking for a stable solution on the graph which satisfies

$$s(a,b) = \begin{cases} 1 & \text{if } a = b \\ \frac{C}{|I(a)||I(b)|} \sum_{c \in I(a), d \in I(b)} s(c,d) & \text{if } a \neq b, \end{cases} \quad (1)$$

where C is a decay factor satisfying $0 < C < 1$.

In practice, SimRank is computed in an iterative manner. Denoting $s^t(a, b)$ as SimRank scores at iteration t, Jeh and Widom [12] initialize the SimRank scores as

$$s^0(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b, \end{cases} \quad (2)$$

and then updated using (1). The update process will take T iterations, and the final s^T will converge to the similarity value in (1).

Note that computing SimRank directly using (1) is very expensive. At first glance, SimRank computation looks like the famous PageRank algorithm [22] since they are both iterative algorithms running on graphs. However, computing PageRank is a much easier task since we can finish one iteration of PageRank by visiting all the edges in the graph once. Given a graph with N nodes and D edges, the time complexity of computing PageRank is $O(D)$ and a space complexity is $O(N)$. In contrast, computing SimRank by iteratively evaluating (1) leads to a time complexity of $O(N^4)$ in the worst case and the space complexity of $O(N^2)$. As a result, how to compute SimRank in large scale graphs remains an unsolved problem.

Algorithm 1 : A naive implementation of SimRank Algorithm on MapReduce

Input: Graph G, initialized s^0
for t = 0: T-1
  Map Function$((a, b), s^t(a, b))$
    find a, b's neighbor $I(a)$ and $I(b)$ respectively
    for each $c \in I(a)$, $d \in I(b)$
      output $(c, d), s^t(a, b)$
  Reduce Function (Key = (c,d), Values = vs[])
    if $c = d$
      $s^{t+1}(c, d) = 1$
    else
      $s^{t+1}(c, d) = \frac{C}{len(vs)} \text{sum}(vs)$
    output (c, d),$s^{t+1}(c, d)$
Output: updated s^T

3. A NAIVE IMPLEMENTATION OF SIMRANK ON MAPREDUCE

Our goal is to speed up SimRank computation on large networks that cannot be efficiently processed in a single machine, by dividing both the computation time and memory requirements across the multiple machines. To achieve our goal, we develop algorithms under the MapReduce framework. MapReduce is attractive because it runs computations in parallel on a large cluster of machines, while handling details such as data distribution, fault-tolerance, load balancing, etc.

In order to use the framework, programs are written as map and reduce functions. Map functions are run in parallel on each part of the input data. The resulting output is key-value pairs. Each pair is sent, in most cases over the network, to a reduce process according to its key. Each reduce process then groups values of the same key and runs the reduce function over these values to produce the final output.

Algorithm 1 shows our first algorithm of computing SimRank on MapReduce. The key of the map function is a pair of nodes denoted as (c, d), which is called a node-pair for brevity. In map function, each SimRank score $s(a, b)$ is distributed to all the neighboring node-pairs corresponding to the key of node-pair (c, d). In the reduce function, the new SimRank score $s(c, d)$ is updated by summing up all the values passed to node-pair (c, d).The MapReduce-based SimRank algorithm has two advantages. On one hand, the computation is distributed to multiple machines. Moreover, the required size of memory is greatly reduced.

However, we should realize that such a naive implementation has its limitations. Each mapper needs to send $s^t(a, b)$ multiple times to the reducer. For a graph with N nodes, suppose p is the average number of neighbors to which a node is connected; we can estimate the amount of data transferred from mappers to reducers as $O(p^2N^2)$. Mapper and reducer processes are very likely to exist on different machines, so the huge amount of data transfer between mappers and reducers will slow down the entire distributed system and even result in many IO errors.

To alleviate the burden of computing SimRank, this paper develops Delta-SimRank, a new algorithm with less data transfer and faster speed in distributed systems. Next we will start by discussing an abstract model named Harmonic Field on the the Node-pair Graph(HFONG), from which we will gain good insights into Delta-SimRank and make our algorithm easy to understand.

4. HARMONIC FIELD ON NODE-PAIR GRAPH

We first review the node-pair graph representation [12], which provides an alternative view for graph similarity. Given N subjects in the original graph $G = \{V, E\}$, we construct a node-pair graph $G^2 = \{V^2, E^2\}$ with N^2 nodes. In the node-pair graph, each node denotes one pair of subjects of the original graph. For example, one node ab in G^2 corresponds to a pair of nodes a and b in G. To embed the neighborhood information, we construct the node-pair graph in the following way: In G^2, there is an edge $(ab, cd) \in E^2$ if $(a, c) \in E$ and $(b, d) \in E$.

Suppose each node in $ab \in G^2$ corresponds to a non-negative value $f(ab)$. We call such values *node-pair scores*. By using the notation of the node-pair graph, we can rewrite the SimRank updating step as

$$f^{t+1}(ab) = \begin{cases} 1 & \text{if } a = b \\ \frac{C}{|I(a)||I(b)|} \sum_{ij \in I(ab)} f^t(ij) & \text{if } a \neq b. \end{cases} \quad (3)$$

Starting from (3), we study a more general model. Suppose the nodes in a G^2 are separated into two disjoint sets U and L satisfying $V^2 = U \bigcup L$. Letting x or y denote a node in G^2, the general model can be written as

$$f^{t+1}(x) = \begin{cases} f^0(x) & \text{if } x \in L \\ \sum_{y \in I(x)} w_{xy} f^t(y) & \text{if } x \in U, \end{cases} \quad (4)$$

where $I(x)$ denotes the neighboring nodes of x, w_{xy} is a weight between x and y, and $f^0(x)$ satisfies $0 \leq f^0(x) \leq 1$. Here L stands for the set where node-pair scores are fixed, and U is complementary to L. In U, a node-pair score is updated as the the weighted average of f in its neighborhood. Such a function f is called a harmonic function [7]. We name the model in (4) as the *Harmonic Field On Node-pair Graphs* (HFONG). HFONG is related to Zhu et al's semi-supervised learning model [28]: if we view G^2 as a graph in which L is labeled with f^0, and U is not labeled, then the process of updating f is similar to the process of finding the optimal labels of U. However, Zhu's work aims to estimate the discrete labels, while our goal is to estimate the harmonic function between 0 and 1.

An important property of the HFONG in (4) is that, due to the maximum principle of harmonic functions [28][7], it will converge to a unique solution. It is easy to find the analytical solution of the converged value. Suppose we organize the scores on V^2 into a long vector $\mathbf{f}$, and the corresponding vectors on U and L are $\mathbf{f}_u$ and $\mathbf{f}_l$, respectively, with $\mathbf{f} = \begin{bmatrix} \mathbf{f}_l \\ \mathbf{f}_u \end{bmatrix}$. Note that $\mathbf{f}_l = f_l^0$ is a vector which remains unchanged during the iterations. Letting W be the weight matrix with each element w_{xy} , we can split W into four blocks over the set of U and L by $W = \begin{bmatrix} W_{ll} & W_{lu} \\ W_{ul} & W_{uu} \end{bmatrix}$. Then (4) can be written in a vector form

$$\mathbf{f}_l \leftarrow \mathbf{f}_l^0 \quad (5)$$

$$\mathbf{f}_u \leftarrow W_{uu}\mathbf{f}_u + W_{ul}\mathbf{f}_l \quad (6)$$

When the HFONG converges, we have $\mathbf{f}_u = W_{uu}\mathbf{f}_u + W_{ul}\mathbf{f}_l$, which leads to

$$\mathbf{f}_u = (I - W_{uu})^{-1} W_{ul}\mathbf{f}_l. \quad (7)$$

Note that the convergence values will be the same even subject to different initialization.

We cannot use (7) directly to compute SimRank because solving linear equations directly in (7) is too expensive for large networks. However, we make use of Eq. (7) to analyze the following two examples of HFONG.

Example 1: If we let $\mathbf{f}_l^0 = \mathbf{1} = [1, 1, ...]^T$ in HFONG, we can analyze the SimRank function as

$$f^{t+1}(x) = \begin{cases} 1 & \text{if } x \in L \\ \sum_{y \in I(x)} w_{xy} f^t(y) & \text{if } x \in U. \end{cases} \quad (8)$$

From the discussion of HFONG we get the converged score

$$\mathbf{f}_u = (I - W_{uu})^{-1} W_{ul}\mathbf{1}. \quad (9)$$

Example 2: For another example, we assume $\mathbf{f}_l^0 = \mathbf{0} = [0, 0, ...]^T$. The SimRank scores are thus updated according to

$$f^{t+1}(x) = \begin{cases} 0 & \text{if } x \in L \\ \sum_{y \in I(x)} w_{xy} f^t(y) & \text{if } x \in U. \end{cases} \quad (10)$$

From the discussion of HFONG we get the converged score

$$\mathbf{f}_u = (I - W_{uu})^{-1} W_{ul}\mathbf{0} = \mathbf{0}. \quad (11)$$

It is easy to see the difference between Example 1 and 2: the node-pair scores in Example 1 generally converge to a non-zero vector, while the scores in Example 2 will converge to zero. Next we will show Example 2 corresponds to an important model named Delta-SimRank and its properties are preferred in distributed computing.

5. DELTA-SIMRANK

Now we consider the representation in the original graph corresponding to Example 2 in the last section. Suppose $x \in V^2$ corresponds to node a, b in the original graph, and y corresponds to node c, d. Let $L = \{aa | a \in V\}$, $U = \{ab | a, b \in V, a \neq b\}$, and

$$w_{xy} = w_{ab,cd} = \frac{C}{|I(a)||I(b)|},$$

then we can write Example 2 in a new form

$$\Delta^{t+1}(a, b) = \begin{cases} 0 & \text{if } a = b \\ \frac{C}{|I(a)||I(b)|} \sum_{c \in I(a), d \in I(b)} \Delta^t(c, d) & \text{if } a \neq b. \end{cases} \quad (12)$$

We call the model as Delta-SimRank and $\Delta^t(a, b)$ as the Delta score at iteration t. Both Delta-SimRank and SimRank are examples of HFONG. However, Delta-SimRank has some unique properties which make it attractive for distributed computing. We will discuss these properties in the following.

PROPERTY 1. *The computation of SimRank can be solved by the use of Delta-SimRank.*

PROOF. If we initialize

$$\Delta^1(a, b) = s^1(a, b) - s^0(a, b), \quad (13)$$

then it is easy to see that (we first consider $a \neq b$):

$$\Delta^{t+1}(a, b) = \frac{C}{|I(a)||I(b)|} \sum_{c \in I(a), d \in I(b)} s^t(c, d) - s^{t-1}(c, d) \quad (14)$$

$$= \frac{C}{|I(a)||I(b)|} \sum_{c,d} s^t(c, d) - \frac{C}{|I(a)||I(b)|} \sum_{c,d} s^{t-1}(c, d)$$

$$= s^{t+1}(a, b) - s^t(a, b)$$

Note that this condition holds even for $a = b$.

$$\Delta^{t+1}(a, a) = s^{t+1}(a, a) - s^t(a, a) = 1 - 1 = 0 \quad (15)$$

Then we can transform the problem of computing SimRank to the problem of updating Delta scores:

$$\Delta^{t+1}(a,b) = \frac{C}{|I(a)||I(b)|} \sum_{c\in I(a), d\in I(b)} \Delta^t(c,d), \quad \text{if } a \neq b$$

$$s^{t+1}(a,b) = s^t(a,b) + \Delta^{t+1}(a,b)$$

□

Note that (12) in fact models the change of SimRank, which is the reason for the name Delta-SimRank.

PROPERTY 2. *If the initialized $\Delta^1(a,b) \geq 0$ for all possible node-pairs, then these Delta scores keep non-negative for all iterations.*

PROOF. We first consider Delta score in the second iteration.

$$\begin{aligned}\Delta^2(a,b) &= \frac{C}{|I(a)||I(b)|} \sum_{c\in I(a), d\in I(b)} \Delta^1(c,d) \\ &\geq \frac{C}{|I(a)||I(b)|} \sum_{c\in I(a), d\in I(b)} 0 \\ &= 0.\end{aligned}$$

Similarly, we can prove $\Delta^t(a,b) \geq 0$ holds for $t = 3, 4, \cdots$. □

PROPERTY 3. *After some iterations of updating, Delta-SimRank scores converge to zero.*

PROOF. Considering Eq.(11) in the last section, we can see that Delta-SimRank scores will converge to **0**. □

Table 1: The evolvement of similarity score between Univ and Prof B

iterations:	3	4	···	8	9	10
SimRank:	0.128	0.128	···	0.128	0.132	0.132
Delta-SimRank:	0.128	0	···	0	0.004	0

To get an intuitive understanding of Delta-SimRank, we use a toy example shown in Figure 1, and observe the similarity between nodes. This example was also used in Jeh and Widom's paper [12], and we employ this example to study the differences between SimRank and Delta-SimRank.

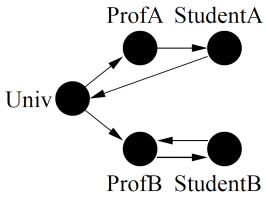
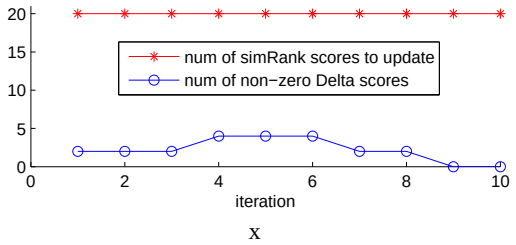


Figure 1: A toy network used in origin SimRank paper.

Table 1 illustrates the evolvement of the SimRank score and Delta score between University and Professor B in the toy network. For this example we set decay factor $C = 0.8$ as in [12]. Note that using a small decay factor will lead to faster convergence. From Table 1 we can see that in a lot of iterations the Delta score is zero while most of the SimRank score is a non-zero value. SimRank value keeps increasing in some iterations while staying unchanged in some other iterations.



x

Figure 2: Updating Δ on the toy network in Figure 1.

Algorithm 2 : Computing Delta-SimRank on MapReduce

Input: Graph G, initialized Δ^t

Map function((a,b), $\Delta^t(a,b)$)
  if $a = b$ or $\Delta^t(a,b) \leq \epsilon$
    return
  find a, b's neighbor $I(a)$ and $I(b)$ respectively
  for each $c \in I(a)$, $d \in I(b)$
    output (c,d), $\frac{C}{|I(c)||I(d)|}\Delta^t(a,b)$
Reduce function (Key = (c,d), Values = vs[])
  if $c = d$
    output $\Delta^{t+1}(c,d) = 0$
  else
    output $\Delta^{t+1} = \text{sum}(vs)$

Output: updated Δ^{t+1}

Figure 2 shows the evolvement of non-zero Delta-scores in the toy example. We can see that for this toy example, the similarity scores between different nodes converge at different speeds. A small number of Delta-SimRank values are zeros. If we compute SimRank scores directly, we need to re-compute the pair-wise similarity for all nodes. In Algorithm 1, the SimRank scores are generally non-zero and all the scores should be sent from mappers to reducers. However, in Delta-SimRank, we send only non-zero Delta scores to reducers. In other words, we need only transfer the non-zero data across the MapReduce system and communication traffic is lower. Based on the above discussion, we can see that *compared with SimRank, Delta-SimRank is more efficient to compute on MapReduce.* In the next section we will discuss how to design the distributed algorithm for Delta-SimRank.

6. DELTA-SIMRANK ON MAPREDUCE

In this section, we first discuss how to implement Delta-SimRank in eq.(12) efficiently on MapReduce, and then propose a faster solution of SimRank.

Algorithm 3 : An efficient approach to compute SimRank

Input: : Graph G, init SimRank s^0

Update SimRank using Algorithm 1 and obtain s^1.
Init Delta-SimRank by $\Delta^1 = s^1 - s^0$
for t = 1: T-1
  update Δ^{t+1}-SimRank as in Algorithm 2.
  $s^{t+1} = s^t + \Delta^{t+1}$

Output: updated SimRank score s^T

Algorithm 2 describes our implementation of Delta-SimRank on MapReduce (single iteration). Delta-SimRank shares a lot of similarities with SimRank in Algorithm 1, since both of them are examples of HFONG. However, there are two significant differences be-

tween Algorithm 2 and Algorithm 1. First, Delta-SimRank checks whether $\Delta^t(a,b) \leq \epsilon$ before sending the data to reducers. This will significantly reduce the amount of the data transmission. In addition, Delta-SimRank pre-computes the coefficient $\frac{C}{|I(c)||I(d)|}$ in the map function. Since only non-zero Δ are sent to the reducer, the size of neighborhood $|I(c)||I(d)|$ is no longer equal to the length of vs and need to be pre-computed. In the implementation, we can store $|I(c)|$ into a separate file and need not compute them in the map function.

The communication traffic of Delta-SimRank is lower than that of SimRank. Suppose there are only M non-zero Delta scores, then the data transferred from mappers to reducers is $O(p^2M)$. In contrast, we have discussed in Section 3 that the load of SimRank is $O(p^2N^2)$. When the communication traffic is low, distributed system will suffer less from transmission errors which further improves the system efficiency.

Based on Algorithm 2, we can design a new efficient algorithm for SimRank. Intuitively, the new method does not recompute the SimRank scores of the nodes that have already converged, but focus on the nodes whose Δ^t are non-zero. Our algorithm is summarized in Algorithm3.

Next we discuss some implementation issues for Delta-SimRank.

Rounding Errors: Although we have proved that SimRank can be exactly transformed into a Delta-SimRank problem (ref. Property 3), in practice there might exist rounding errors since in Algorithm 3, we use $\Delta \leq \epsilon$ as the condition to check whether Δ is zero. To analyze the effects of rounding errors, we consider the following property:

PROPERTY 4. *If* $\max_{a,b} \Delta^t(a,b) \leq \epsilon$, *we can estimate the upperbound of* $\Delta^t(a,b)$ *after* t_0 *iteration by* $\max_{a,b} \Delta^{t+t_0}(a,b) \leq C^{t_0}\epsilon$.

We omit the details of the proof due to space limited. Based on this property, we can see that the rounding error will decrease with more iterations with $C \leq 1$. In practice, we use $\epsilon = 10^{-4}$ in our experiment, and the rounding error will be negligible in most of the applications.

Number of Iterations: How to select the number of iterations is an important problem. Jeh and Widom [12] empirically suggested choose $T = 5$ with a decay factor $C = 0.8$. However, Lizorkin et al. [19] showed that $T = 5$ is not enough to obtained a converged solution. They suggested to use a larger T or a smaller decay factor. From all our experiments, we observed that when $C = 0.4$, Delta-SimRank scores become all zeros within 8 iterations, which means SimRank converges. On some datasets it converges even faster (within 5 iterations).

Input Distribution: In our algorithm, each mapper need not access the whole adjacent matrix. On the contrary, the necessary information is limited to the neighbors of a given node-pair. As a result, the input distribution is easy for sparse graph. One might argue that the size of neighborhood $|I(c)|$ or $|I(d)|$ requires going over all the nodes in the graph, however, such neighborhood size could be computed before hand and then saved with $O(N)$ space. As a result, our model is fit for the distributed computing.

7. EXPANDING GRAPHS

In real life and Internet communities, networks are rarely static. People might get to know new friends, or have new collaborators; social media communities such as Facebook or Twitter accommodate new users every day. To model those networks, we define an *expanding graph* as the graph in which the number of nodes or edges keeps increasing. To compute the SimRank scores in an expanding graph, a straightforward way is to update the adjacent matrix and compute the scores from a fresh start. However, this naive way neglects the scores computed in previous stages, and is not efficient enough for real life applications. In this section, we will use Delta-SimRank to develop an efficient way of computing SimRank scores in expanding graphs.

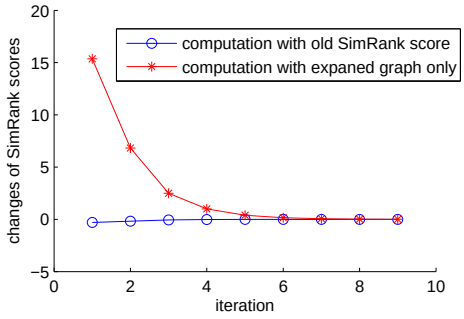


Figure 3: Delta-SimRank on expanded graph can converge to the solution faster

Suppose the old graph is $G = \{V^{old}, E^{old}\}$, and the new graph is $G = \{V, E\}$. To guarantee that the graph is expanding, the constraint satisfies $E^{old} \subseteq E$, $V^{old} \subseteq V$. In general cases, the number of newly added nodes or edges is relatively small compared with the size of G^{old}. To compute the SimRank scores on G, a naive way is to re-compute SimRank again. However, with Delta-SimRank we can employ the information from the old graph to speed up the computation. Suppose s^{old} is the SimRank score on G^{old}, we can get the corresponding initialization on G by

$$s^{init}(a,b) = \begin{cases} s^{old}(a,b) & \text{if } a,b \in V^{old}, (a,b) \in E^{old} \\ 1 & \text{if } a = b, a \in V, a \notin V^{old} \\ 0 & \text{otherwise.} \end{cases}$$

Then we can use s^{init} as the initialization on graph G. Based on the discussion of HFONG, it is not difficult to see this initialization will also converge to the ground-truth SimRank scores. However, s^{init} is close to the SimRank scores so that we might get a better estimation for Delta-SimRank.

After one iteration of SimRank from s^{init}, we obtain s^*. Then we can initialize Delta score by $\Delta^1(a,b) = s^* - s^{init}$, and proceed Delta-SimRank algorithm. Note that the initialized value is a good initialization so that Δ will converge to zeros faster. To show the effectiveness of our algorithm, we employ a toy example with a subgraph with 99 nodes from Facebook network, expand the graph by adding a new node. Figure 3 compares the speed of convergence using traditional method and using our new algorithm. Obviously our new algorithm is fit for expanded graph and enjoys a faster convergence speed with lower communication traffic.

8. EXPERIMENTS

To validate the success of our Delta-SimRank algorithm, we evaluate its time and accuracy on Hadoop, an open source implementation of MapReduce. We compare our algorithms with the *distributed SimRank* in Algorithm 1, and also show the improvement of using a distributed implementation over a single core algorithm. We use the following datasets for the experiments:

- **Facebook social network dataset** is a subgraph from the Facebook New Orleans regional network [24], which contains 10,000 selected users and 269,037 friendship links between those users. This social network is undirected and the average number of neighbors per node is 26.9.

- **Wiki-vote dataset** is the history of the administrator election when ordinary users are promoted to administratorship in the Wikipedia community [15]. There were 7,115 users participating in the elections, resulting in 103,689 total votes. Thus this wiki-vote social network contains 7,115 nodes with 103,689 directed edges. The average number of neighbors per node for this network is 14.6.

- **Collaboration network datasets** include ca-GrQC (5424 nodes, 28980 edges) and ca-HepTh (9877 nodes, 51971 edges) [16], which are the networks that illustrate the collaborations between physics researchers submitted on arXiv. The nodes in each graph represent authors and an edge shows that two connected researchers have collaborations. The average numbers of neighbors per node for the two networks are 5.3 and 5.3.

- **DBLP co-author datasets** We use a real data set from the DBLP Computer Science Bibliography [23] to build a co-authorship network. In this network, a node represents an author, and an edge between two nodes denotes that the authors have collaborated papers. We restricted network of papers published in four areas (data mining, database, machine learning, and information retrieval). To remove the isolated nodes with less collaboration, we only select the top 1000 authors. There are 8548 edges in the network and the average number of neighbors per node is 8.5.

The implementation was written in Python using the Dumbo library.[2] We evaluated the implementation on the UIUC IFP distributed computing system, which contains eight computation nodes running Hadoop 0.21.0. Each computing node is equipped with two Intel Quad Core Xeon 2.4 GHz CPUs and 16 GB memory. For each job, 64 map tasks and 8 reduce tasks were assigned in the cluster. Note that the absolute times in our experiments do not necessarily reflect a speed limit to our algorithms, because MapReduce can scale up performance by adding more machines. Still, despite using our small testbed, our system still handled large networks with impressive performance.

8.1 Delta-SimRank vs. SimRank on Distributed Systems

Table 2 illustrates the time and accuracy of our Delta-SimRank algorithm (Section 5) and the distributed SimRank algorithm (Section 3). We observe that Delta-SimRank is significantly faster than the SimRank algorithm. In the Facebook dataset, Delta-SimRank works 17.6 times faster than the SimRank algorithm, while in Wiki-Vote dataset the Delta-SimRank algorithm is 24.6 times faster. In ca-GrQc and ca-HepTh, the networks are sparser and each node is connected to fewer neighbors. From the experiments we can see Delta-SimRank is fit for processing such sparse networks, and the ratio of speed is 16.0 and 33.7 times faster on the two datasets. On the DBLP dataset with smallest network, the overhead of distributing jobs takes effect. However, the improvement in speed is still 1.8 times.

As discussed in Section 6, the effect of rounding error in Delta-SimRank is very small. Table 2 validates this conclusion. The largest mean square errors are 8.06×10^{-6}, which is negligible for most of the applications.

The reason for Delta-SimRank's success lies in reducing the communication traffic load. Table 3 compares the amount of the intermediate data in Delta-SimRank and SimRank. Such intermediate data is generated by map functions and received by reduce functions, and becomes the bottleneck for computing SimRank on large networks. Figure 4 further illustrates the amount of intermediate data transferred in each iteration. For the proposed Delta-SimRank algorithm the intermediate data dramatically decreases over iterations as only the difference of SimRank scores will be transferred. On the other hand, the SimRank algorithm requires transferring all the SimRank scores even when scores have almost converged. The speedup ratio of Delta-SimRank is highest on ca-HepTh (33.7 times faster), corresponding to the most significant reduce of intermediate data size (45.7 times smaller). Overall our Delta-SimRank method is consistently more efficient than traditional SimRank method.

To examine the scalability of Delta-SimRank, we first evaluate the proposed method on different sizes of subgraph from the Facebook dataset. The number of nodes and edges for the four subgraphs are (10,26), (100,494), (1000,13797) and (10000,269037). As illustrated in Figure 5, we can see that the computational complexity of the SimRank method is about quadratic in number of nodes while that of the proposed Delta-SimRank is much better.

[2]https://github.com/klbostee/dumbo

Table 2: Comparison of SimRank and Delta-SimRank over 5 datasets in terms of running time (sec), and mean squared error of similarity score between two algorithms.

Dataset	Time		Speedup Ratio	MSE
	SimRank	Delta-SimRank		
Facebook	190870	10835	17.6	9.36 e-07
Wiki-Vote	32065	1305	24.6	8.11 e-12
ca-GrQc	9712	608	16.0	6.19 e-06
ca-HepTh	33120	984	33.7	6.48 e-06
DBLP	1264	689	1.8	1.23 e-09

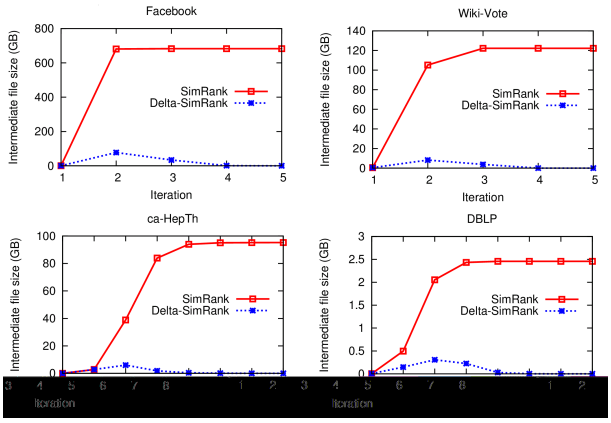


Figure 4: The amount of intermediate data transferred in the distributed system for each iteration.

Table 3: Comparison of SimRank and Delta-SimRank over 5 datasets on intermediate data size (GB).

Dataset	SimRank	Delta-SimRank	data size ratio
Facebook	2731	113	24.2
Wiki-Vote	380	12.6	31.2
ca-GrQc	142	6.4	22.2
ca-HepTh	505	11	45.9
DBLP	14.8	0.7	24.1

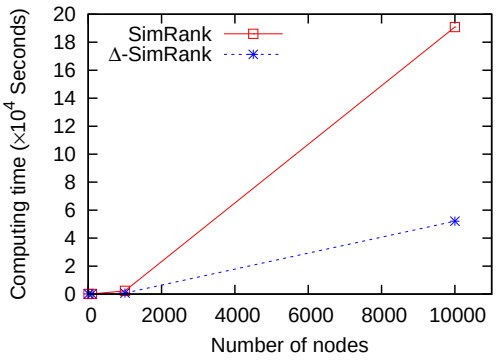


Figure 5: Comparing the scalability of SimRank and proposed Delta-SimRank algorithm

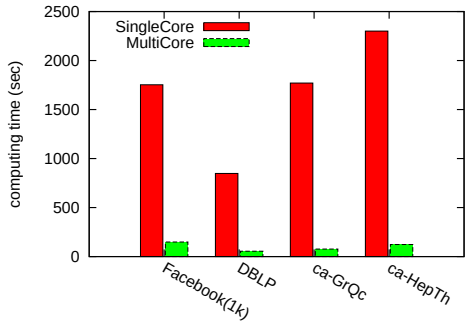


Figure 6: Comparison of running time of SimRank on single core machine and Delta-SimRank algorithm in distributed systems.

8.2 Distributed Computing vs. Single Core Computing

To show the advantages of distributed computing, we compare the performance of single core SimRank and Delta-SimRank in Figure 6. The single core implementation follows the same workflow as Algorithm 1, but working on a single machine instead of distributed systems. Note that single core computation can only handle small datasets due to the limitation of computational power and storage. We only show comparison on DBLP, ca-GrQc and ca-HepTh datasets together with a subset of Facebook dataset (Facebook 1K). From the figure we can see that with the help of scalable MapReduce paradigm, distributed Delta-SimRank is significantly better than the single core implementation. Moreover, we can further improve the efficiency by employing more computers and enlarging the distributed systems. Although there are many other algorithms which improve the efficiency of SimRank in single machines, their performances are limited to the computation power and memory size in a single machine. From the results we believe distribute direction is a promising direction.

8.3 Expanding Graph

We further evalute the proposed Delta-SimRank algorithm for expanding graph on four datasets. We randomly remove one node and the edges related to that node from the graph and obtain the SimRank scores, which are then used to compute the initial delta scores for Delta-SimRank on the expanding graph (i.e., the original graph).

Table 4: Performance of Delta-SimRank for expanding graph.

Dataset	Time (s)	Iteration	MSE	Speedup Ratio
Facebook (1k)	254	2	6.12 e-06	2.92
ca-GrQc	210	3	7.63 e-06	2.90
ca-HepTh	452	3	7.48 e-07	2.18
DBLP	146	2	6.34 e-07	2.96

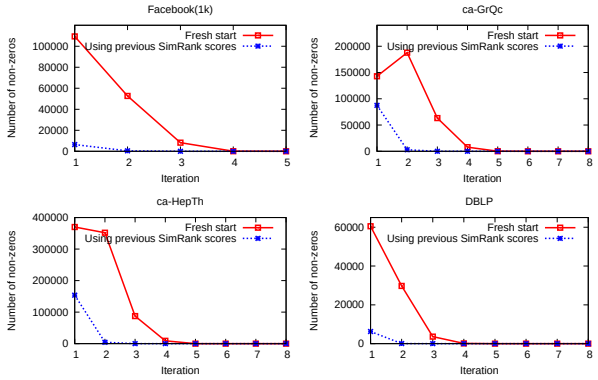


Figure 7: Comparison of the amount of non-zeros for expanding graph.

Table 4 shows the comparison of the results to the SimRank scores obtained directly from original graph. It can be seen that with the Delta-SimRank algorithm, the expanding graphs converge within very few iterations. Also the intermediate data (non-zero delta's) is small compared to the fresh start from the original graph (Figure 7).

9. RELATED WORKS

Motivated by the success of Google's distributed computing systems and the popularity of Hadoop, many researchers have been working on designing efficient algorithms on MapReduce systems. Chu *et al.* [5] showed that many popular machine learning algorithms, including weighted linear regression, naive Bayes, PCA, K-means, EM, Neural Network, logistic regression, and SVM can be implemented in the framework of MapReduce. However, [9] shows that for some algorithms, the distributed algorithm might suffer from the overhead in distributed computing. Many algorithms have been implemented in Apache Mahout library. [1]. Ye *et al.* [25] implemented decision trees on Hadoop. Gonzalez *et al.* [10] discussed how to implement parallel belief propagation using the Splash approach. The MapReduce framework has gained much popularity in machine learning tasks.

MapReduce also finds a lot of applications in the area of graph mining. It is easy to implement PageRank algorithm with MapReduce framework [6]. This idea is further extended by Kang *et al.* [14] who implemented graph mining library, PEGASUS, on the Hadoop platform. They first introduce a GIM-V framework for matrix-vector multiplication and show how it can be applied to graph mining algorithms such as diameter estimation, the PageRank estimation, random walk with restart calculation, and finding connected-components. More recently, Malewicz *et al.* proposed the Pregel system [20] which can reduce the system IO in MapReduce. Compared with these problems, computing SimRank yields higher computational complexity and leads to communication traffic. It is not trivial to design an efficient SimRank algorithm on MapReduce. To the best of our knowledge, our work is the first to

study this problem.

The idea of our Delta-SimRank is consistent with the previous studies in computing PageRanks [13], [21], in the fundamental idea that the scores in the graph should be updated adaptively. However, our new model has the following unique properties: (1) Our Delta-SimRank is implemented using the framework of MapReduce, with the benefit of reducing communication traffics in distributed systems. (2) Some previous works believe the ranking scores will never change after it becomes stable, which is not true for SimRank computation. For the example in Table 1, the SimRank score can still increase after a few iterations of staying un-changed. (3) This paper focuses on the problem of SimRank computation, which is much more expensive to compute than the traditional PageRank algorithms.

Quite a few studies are devoted to speed up the SimRank computation, although these works are designed for a single machine. Antonellis *et al.* [2] extended SimRank equations to take into consideration an evidence factor for incident nodes and link weights. Their algorithm shows better performance than SimRank; however, the computation issue is not solved. Lizorkin *et al.* proposed several optimization techniques for speeding up SimRank iterative computation [19]. Their method improved computational complexity from $O(n^4)$ to $O(n^3)$ in the worst case. Li *et al.* [17] proposed to estimate SimRank scores of both static graph and dynamic graph by solving the eigenvectors of a large linear matrix. However, their approach is limited to relatively small scale, and not applicable for large scale network due to huge memory consumption. Fogaras and Racz [8] suggested speeding up SimRank computation through probabilistic computation via the Monte Carlo method. In another work, Yin *et al.* [26] employed a hierarchical structure named SimTree for SimRank computation. By merging computations through the same branches in SimTree, this method saves a lot of computations. However, this method cannot solve the computation problem in large networks. Yu et al [27] developed an efficient SimRank algorithm which reduce the time complexity from $O(n^3)$ to $O(min(n \cdot m, n^{log_2 7}))$. He and his colleagues [11] employed graphics processing units (GPUs) to accelerate the computation. Unlike [11] which utilizes GPU's high memory bandwidth between graphic processing units, this paper aims to reduce the communication traffic between mapper and reducers. There is little similarity between the two algorithms due to significant differences in MapReduce and GPUs.

10. CONCLUSION

In this paper we developed Delta-SimRank, a new efficient algorithm for solving SimRank on a distributed cluster using MapReduce. Our discussion starts from the analysis of the HFONG model, and then derives Delta-SimRank algorithm which can be used to compute SimRank scores with less communication traffic and faster speed. Our experiments on four real datasets verify the success of Delta-SimRank. In the best case, we get up to 30 times speed-up compared with the distributed SimRank algorithm.

11. ADDITIONAL AUTHORS

Additional authors: ChengXiang Zhai and Thomas S. Huang, email: `czhai@cs.illinois.edu`, `huang@ifp.uiuc.edu`, University of Illinois at Urbana-Champaign.

12. REFERENCES

[1] Mahout library. http://lucene.apache.org/mahout.

[2] I. Antonellis, H. Garcia-Molina, and C.-C. Chang. Simrank++: Query rewriting through link analysis of the click graph. *Proc. VLDB Endow.*, 2008.

[3] S. Bao, G. Xue, X. Wu, Y. Yu, B. Fei, and Z. Su. Optimizing web search using social annotations. In *International conference on World Wide Web*, pages 501–510. ACM New York, NY, USA, 2007.

[4] Y. Champclaux, T. Dkaki, and J. Mothe. Enhancing high precision using structural similarities. In *IADIS*, volume 8, pages 494–498, 2008.

[5] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, and A. Y. Ng. Map-reduce for machine learning on multicore. *NIPS*, 2006.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, 2004.

[7] P. Doyle and J. Snell. Random walks and electric networks. 1984.

[8] D. Fogaras and B. Rácz. Scaling link-based similarity search. In *WWW*, 2005.

[9] D. Gillick, A. Faria, and J. DeNero. MapReduce: Distributed Computing for Machine Learning, 2006.

[10] J. Gonzalez, Y. Low, and C. Guestrin. Residual splash for optimally parallelizing belief propagation. AISTATS, 2009.

[11] G. He, H. Feng, C. Li, and H. Chen. Parallel simrank computation on large graphs with iterative aggregation. In *KDD*, pages 543–552, 2010.

[12] G. Jeh and J. Widom. Simrank: A measure of structural-context similarity. In *In KDD*, 2002.

[13] S. Kamvar, T. Haveliwala, and G. Golub. Adaptive methods for the computation of PageRank. *Linear Algebra and its Applications*, 386:51–65, 2004.

[14] U. Kang, C. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system - implementation and observations. *ICDM*, 2009.

[15] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. *WWW*, 2010.

[16] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. KDD*, 1(1), 2007.

[17] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu. Fast computation of simrank for static and dynamic information networks. *EDBT*, 2010.

[18] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

[19] D. Lizorkin, P. Velikhov, M. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for simrank computation. *Proc. VLDB Endow.*, 1(1), 2008.

[20] G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146. ACM, 2010.

[21] F. McSherry. A uniform approach to accelerated pagerank computation. In *WWW*, pages 575–582, 2005.

[22] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1998.

[23] prepared by the Knowledge Discovery Laboratory in University of Massachusetts. Dblp computer science bibliography. http://kdl.cs.umass.edu/data/dblp/dblp-info.html.

[24] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *WOSN*, 2009.

[25] J. Ye, J.-H. Chow, J. Chen, and Z. Zheng. Stochastic gradient boosted distributed decision trees. In *CIKM '09*, 2009.

[26] X. Yin, J. Han, and P. S. Yu. Linkclus: efficient clustering via heterogeneous semantic links. In *VLDB '06*, 2006.

[27] W. Yu, X. Lin, and J. Le. A space and time efficient algorithm for simrank computation. In *APWeb*, pages 164–170, 2010.

[28] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, 2003.