Towards Scalable X10 Based Link Prediction for Large Scale Social Networks

Hidefumi Ogata
Department of Computer Science, Tokyo Institute of Technology
ogata.h.ac@m.titech.ac.jp

Toyotaro Suzumura
IBM Research / JST CREST
suzumura@acm.org

ABSTRACT

The use of social application such as Twitter or FaceBook becomes popular in recent years. In particular, Twitter increases the number of the users rapidly from 2009 as the place that users can tweet anything in 140 characters.

In the area of social network analysis, the user network of Twitter is frequently analyzed. Haewoon, et.al.,[4] analyzed the Twitter user network from various point of view in 2009, and they show that the Twitter user network has some different feature from conventional social networks. Bongwon, et.al., also made a collection of 74 millions tweets in 2010, and investigated the influence that "retweet" gives for diffusion of the information. Such analysis not only reveal the unique characteristics of Twitter user network, but also make some networking service such as finding users who are similar to someone, or the recommendation of commodities by using tweet information.

There are some analysis such as clustering which needs entire data of the network. However, since social networks are increasing day by day, it becomes impossible to obtain the entire network by crawling. As a solution of this problem, there is the network analysis called link prediction. This enables to predict true network from a given part of the network. If we use link prediction, we can recover the entire network from the network data which we already obtained, and apply some analysis such as clustering to predicted network, then we may get the approximate result of the analysis for the entire network.

In our research, we implemented one of the link prediction algorithm named Link Propagation in X10, which is a parallel programming language. And evaluated its scalability and precision with Twitter user network data.

Keywords

social network analytics, link prediction, X10, high performance computing

1. INTRODUCTION

The number of users of social applications is increasing day by day. Now in 2013, the number of Twitter users exceeds 500 millions, which was 42 millions in 2009. Analyzing such a large scale network of social application is difficult for some certain reason. First, whole data of large network have to be stored in local storage before analyzing. If we run some analysis which needs entire data of the network, the data need to be loaded on memory. But it is generally prohibitive because the data is too big. Second, we cannot get the Twitter network data easily because of the limitation of use of Twitter API. Since we can only use Twitter API 150 times in 1 hour, it will take very long time to get all of Twitter network, which has 500 millions users.

To solve such a problem, we attempt to use a kind of network analysis called link prediction. It predicts unknown links from the known links on the network. If we have a partial data of a social network, we can approximately restore the entire network data by using link prediction. Once we get an approximate entire network, various network analysis can be applied to it and get some approximate result. Therefore, it is important to predict the unknown links with high precision.

Link Propagation algorithm [5] proposed by Raymond and Kashima is one of the link prediction algorithm, which performs with high precision and can be applied to large scale networks. In our study, we implemented the Link Propagation algorithm on parallel programming language X10 [3]. But in the paper of Link Propagation, some part of the algorithm are not explained in detail, so we complemented them appropriately in order to be scalable and precise algorithm.

We will explain link prediction and Link Propagation algorithm in section 2, and describe how to implement Link Propagation in X10 in section 3. The scalability evaluation and precision analysis are shown in 4, the related works are listed in 5, and conclusion in 6.

2. LINK PREDICTION PROBLEM

The link prediction problem is to predict links that will be added to the given network in the future. Link prediction has a significant role in the area of social network analysis or protein interaction analysis. For example, user recommendation service on SNS can be realized by link prediction. To predict interaction between unknown proteins from known proteins is also one of the link prediction problem. These are the example of using link prediction for predicting the future network from the current network.

On the other hand, link prediction may be used to predict lack of links from the partial network. We are focus on this surface and investigated how precise Link Propagation predicts the entire Twitter network from the part of it.

2.1 definition

Let $G = \langle V, E, W \rangle$ denotes a network (graph), V denotes a set of vertices, $E = \{\langle u, v \rangle | u, v \in V\}$ denotes a set of edges, and $W : E \longrightarrow \mathbb{R}$ denotes a set of edge weight. Then link prediction is regarded as the problem to predict the network $G' = \langle V, E', W' \rangle (s.t. E \subset E', W \subset W')$ from given network G. When $score : (G, u, v) \longmapsto r \in \mathbb{R}$ denotes the link prediction score and α denotes threshold, the link prediction function is given by eq.(1).

$$pred(u,v) = \begin{cases} 0 : score(G,u,v) < \alpha \\ 1 : score(G,u,v) \geq \alpha \end{cases} \tag{1}$$

2.2 Link Propagation

2.2.1 Abstract

In this section, we describe Link Propagation algorithm which we implemented. Link Propagation is one of the link prediction algorithm for large scale graphs proposed by Raymond and Kashima [5]. This algorithm uses the similarity between two pairs of vertices. If vertex A is similar to C, B is similar to D, and there is a link between C and D, then it predicts that the link between A and B may exist. This concept comes down to matrix calculation and Link Propagation do this in parallel.

Raymond and Kashima proposed exact Link Propagation and approximate Link Propagation. In exact Link Propagation, full size of similarity matrix is used. But in approximate Link Propagation, the similarity matrix is decomposed to the multiplication of small matrices by low rank approximation, that reduces the memory usage though the precision becomes worse than exact one. We employed the approximate Link Propagation for our research in order to analyze large scale networks such as Twitter network.

2.2.2 Algorithm

Link Propagation algorithm is as follows.

1. Calculate a similarity matrix S from the adjacency matrix of given network.

2. Decompose the similarity matrix S as $S = GG^T$ by low rank approximation.

3. Calculate a Core Matrix by the matrix operation of matrix G.

4. Get a prediction score of arbitrarily selected vertex pairs by the matrix operation of Core Matrix.

2.2.3 Similarity Matrix

A similarity matrix is calculated from an adjacency matrix, however Raymond and Kashima did not show the definition of similarity. Then we defined the similarity as,

$$S_1(u,v) = \begin{cases} |\Gamma(u) \cap \Gamma(v)| & : (u,v) \in E \\ 0 & : (u,v) \notin E \end{cases} \tag{2}$$

$$S_2(u,v) = \begin{cases} \dfrac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|} & : (u,v) \in E \\ 0 & : (u,v) \notin E \end{cases} \tag{3}$$

where, an adjacency matrix is a symmetrical matrix.

Now, we would like to mention about the sparsity of the similarity matrix. Generally, it is known that almost all of the large scale real networks and its adjacency matrices are sparse. Therefore, the similarity matrix given by above definition is also sparse matrix. In this paper, we assume that the similarity matrix is the sparse matrix.

2.2.4 Low Rank Approximation

In the step 2 of the algorithm, S is approximated to the small matrix G by those which decompose S to GG^T. For this purpose, we used Cholesky factorization.

Cholesky factorization decompose a positive-definite Hermitian matrix into a lower triangular matrix. In the range of real number, a positive-definite Hermitian matrix equals to a positive-definite symmetrical matrix. The algorithm of Cholesky factorization is shown in Algorithm 1. In this algorithm, since j-th column of lower triangular matrix L is calculated in j-th iteration of outer "for" loop, it can be interrupted the calculation at M-th iteration ($M < N$) and get an approximate lower triangular matrix (i.e., $A \approx LL^T$). It reduces the memory usage for storing data of matrices.

Algorithm 1 Cholesky factorization

```
Require: $A = \{a_{ij}\} \in \mathbb{R}^{N \times N}$
Ensure: $L = \{l_{ij}\} \in \mathbb{R}^{N \times N}, A = LL^T$
  for $j = 1$ to $N$ do
    $l_{jj} \leftarrow \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}$
    for $i = j + 1$ to $N$ do
      $l_{ij} \leftarrow \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}}{l_{jj}}$
    end for
  end for
```

Now we have to mention that Cholesky factorization cannot be directory applied to a similarity matrix of a network, because it may not be positive-definite though symmetrical. If we do Cholesky factorization with a non positive-definite similarity matrix, L must be a complex matrix, since the value in the root in $l_{jj} \leftarrow \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}$ becomes negative. Furthermore, L is probably not a sparse matrix though S is a sparse matrix, and that causes an increase of the memory usage. To solve these problem, we propose approximate Cholesky factorization for non positive-definite symmetrical sparse matrix.

Basic ideas of the approximate Cholesky factorization are to substitute a small positive value for the negative value in the root described above and to zero the value of l_{ij} with all $i, j (i \neq j)$ such that $a_{ij} = 0$. The approximate Cholesky factorization algorithm is shown in Algorithm 2.

Algorithm 2 Approximate Cholesky factorization

```
Require: $Symmetrical matrix A = \{a_{ij}\} \in \mathbb{R}^{N \times N}, M < N, \epsilon > 0$
Ensure: $Lowertriangular matrix L = \{l_{ij}\} \in \mathbb{R}^{N \times M}, A \approx LL^T$
  for $j = 1$ to $M$ do
    $l_{jj} \leftarrow a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2$
    if $l_{jj} \geq 0$ then
      $l_{jj} \leftarrow \sqrt{l_{jj}}$
    else
      $l_{jj} \leftarrow \epsilon$
    end if
    for $i = j + 1$ to $N$ do
      if $a_{ij} \neq 0$ then
        $l_{ij} \leftarrow \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}}{l_{jj}}$
      else
        $l_{ij} \leftarrow 0$
      end if
    end for
  end for
```

In the next step, Core Matrix is created by the matrix operation of L, which is calculated by approximate Cholesky factorization and is regarded as G.

2.2.5 *Core Matrix*

Core Matrix is the small matrix from which the link prediction scores are easily obtained. Once we calculate the Core Matrix, thus it is not needed to get it again. How to calculate the Core Matrix is as follows.

1. $d \equiv GG^T\mathbf{1}$
2. $\tilde{G} \equiv \text{diag}(d)^{-\frac{1}{2}}G$
3. Get eigenvalue decomposition of $\tilde{G}^T\tilde{G}$
   $\tilde{G}^T\tilde{G} = U\text{diag}(\lambda^{(1)}, \lambda^{(2)}, \ldots, \lambda^{(M)})U^T$
4. $V \equiv \tilde{G}U\text{diag}(\lambda^{(1)}, \lambda^{(2)}, \ldots, \lambda^{(M)})^{-\frac{1}{2}}$
5. $[\Lambda]_{i,j} \equiv \lambda^{(i)}\lambda^{(j)}$
6. $[D]_{i,j} \equiv \dfrac{\sigma(1+\sigma)[\Lambda]_{i,j}}{1+\sigma-\sigma[\Lambda]_{i,j}}$
7. Core Matrix: $C \equiv D * (V^TWV)$

σ appeared in Step 6 in the above is the parameter used for weighting of the similarity of the node pair. Since Raymond and Kashima fixed σ to 0.001 in their paper, we also followed that.

An operator $(*)$ denotes cell wise multiplication of two matrices, i.e. when $C = A * B$, $c_{ij} = a_{ij}b_{ij}$.

2.2.6 *Obtaining Link Prediction Scores*

Eq. (5) is used to obtain the link prediction scores from the Core Matrix, where $v_{(i)}$ is the vector which corresponds to i-th row of V, i.e. $V^T = (v_{(1)}, v_{(2)}, \ldots, v(N))$.

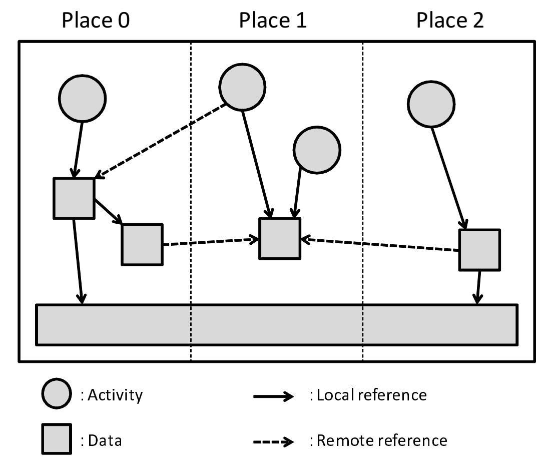

Figure 1: Illustration of PGAS Model

$$P = \frac{1}{1+\sigma}W + \frac{1}{(1+\sigma)^2}VCV^T \quad (4)$$

$$pred(i,j) = [P]_{i,j} = \frac{1}{1+\sigma}[W]_{i,j} + \frac{1}{(1+\sigma)^2}v_{(i)}^T C v_{(j)} \quad (5)$$

3. IMPLEMENTATION OF LINK PROPAGATION

3.1 Parallel Programming Language X10

Recently, Using multicore processor is popular, however it is not easy to write a program that brings out the best performance of the parallel environment consists of such multicore processors. If you use openMP, it becomes easier to parallelize the program by adding special directives in the C or C++ codes, but it may not improve the performance in some cases. On the other hand, MPI make it possible with detailed tuning to outperform, but it tend to be more complicated work than openMP.

X10 is the parallel programming language based on PGAS model that aimed for achieving both productivity and good performance. PGAS (Partitioned Global Address Space) is a parallel programming model where all processors exist on a single address space (Figure 1). Since the programmers do not have to pay attention to the address spaces independent from other process, PGAS language have higher productivity than MPI.

Other languages based on PGAS model are listed below.

- UPC (Unified Parallel C)
- Co-Array Fortran
- Titanium
- Chapel

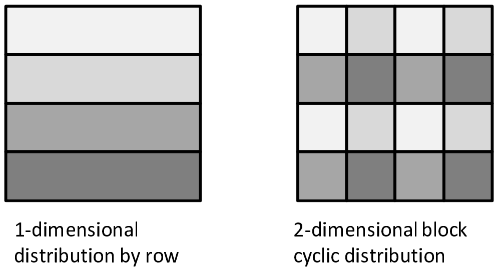
1-dimensional distribution by row
2-dimensional block cyclic distribution

Figure 2: The example of distributing a matrix into 4 compute nodes. The sub matrices of same color are stored in same node. The right figure shows the 2-dimensional distribution used in ScaLAPACK.

3.2 Inplementation

3.2.1 Distributed Sparse Matrix

The important thing when implement Link Propagation in X10 is the way to store the data of large matrices on memory. We would like to handle billion scale network, but it is impossible to store the data of all vertex pairs. Therefore, it is to be desired that the large matrices are stored as sparse matrix format. It reduces huge memory usage, because an adjacency matrix of large scale network has many non zero values and they are ignored in the sparse matrix format. Nevertheless, the memory size of such matrices are still too big to have on one compute node, so matrix data should be distributed in multiple nodes.

There are some types of matrix distribution method. The simplest is 1-dimensional distribution, where the rows (or columns) of a matrix are partitioned (left figure of Figure 2). 2-dimensional distribution is a generalization of 1-dimensional distribution, where the rows and columns are partitioned (right figure of Figure 2). ScaLAPACK library [1] uses one of 2-dimensional distribution named "Block Cyclic Data Distribution", that makes matrix operation faster than 1-dimensional distribution. Additionally, Yoo et.al. proposed the fast matrix-vector multiplication for special type of 2-dimensional distribution. An advantage of 2-dimensional distribution is to decrease the communication cost by data localization. Thus the larger the matrix is, the bigger the benefit of reducing communication cost is. However, it is not effective for a small matrix and implementation tends to be more complicated than 1-dimensional distribution.

In our implementation of Link Propagation, we employed 1-dimensional distribution, because almost all step in Link Propagation algorithm are matrix operation for the small matrix G decomposed from the similarity matrix and it is enough efficient for a small matrix. Furthermore, in 1-dimensional distribution, $\tilde{G}^T\tilde{G}$ is calculated faster than in 2-dimensional distribution. As shown in Figure 3, A^TA is calculated as the sum of results of matrix multiplication in all place. It is more complex in 2-dimensional distribution because some fine grained communication is needed.

The X10 code which computes $\tilde{G}^T\tilde{G}$ is shown in Figure 4. The matrix object named DenseMatrix is created in line 1, each place computes the matrix multiplication on parallel, and team.allreduce gets the sum of the results.

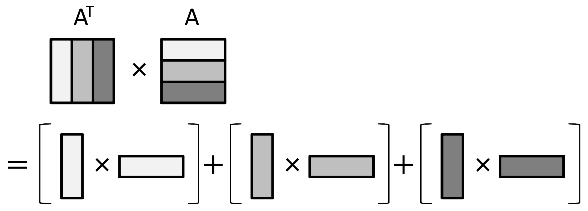

Figure 3: The calculation of A^TA.

```
val result = PlaceLocalHandle.make[DenseMatrix](
             Dist.makeUnique(),
             () => new DenseMatrix(nCol, nCol));
finish for(p in Place.places()) async at(p) {
  val localResult = result();
  for(var i:Int = 0; i < m.nRow; i++) {
    for(var jj:Int = m.offset(i);
        jj < m.offset(i + 1); jj++) {
      val j = m.columns(jj);
      for(var kk:Int = m.offset(i);
          kk < m.offset(i + 1); kk++) {
        val k = m.columns(kk);
        localResult(j, k) = localResult(j, k)
           + m.vertices(jj) * m.vertices(kk);
      }
    }
  }
  team.allreduce(here.id, result().data, 0,
       result().data, 0, result().data.size,
       Team.ADD);
}
```

Figure 4: The calculation of $\tilde{G}^T\tilde{G}$ written in X10

3.2.2 Implementation of Approximate Cholesky Decomposition

The algorithm of approximate Cholesky decomposition was already described in Algorithm 2, but we modified it in order to compute more efficiently. When $i = j$, $a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2$ and $a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk}$ are the same, so the algorithm can be modified to Algorithm 3. In this paper, we call it Modified Approximate Cholesky Decomposition (MACD).

It is easy to parallelize MACD. The computation step of parallelized MACD is shown in Figure 5. The only communication occurred in this algorithm is to broadcast the elements whose value is determined in j-th row of L, and other parts of the algorithm are completely calculated in parallel.

3.2.3 Eigenvalue Decomposition by GotoBLAS2

Link Propagation needs eigenvalue decomposition, but the matrix to be decomposed is so small matrix that it can be stored in one compute node, thus we did eigenvalue decomposition by using GotoBLAS2, which is the linear calculation library [2].

Algorithm 3 Modified Approximate Cholesky Decomposition

```
Require: $A = \{a_{ij}\} \in \mathbb{R}^{N \times N}, M < N, \epsilon > 0$
Ensure: $L = \{l_{ij}\} \in \mathbb{R}^{N \times M}, A \approx LL^T$
  for $j = 1$ to $M$ do
    for $i = j$ to $N$ do
      if $a_{ij} \neq 0$ then
        $l_{ij} \leftarrow a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}$
      else
        $l_{ij} \leftarrow 0$
      end if
    end for
    if $l_{jj} \geq 0$ then
      $l_{jj} \leftarrow \sqrt{l_{jj}}$
    else
      $l_{jj} \leftarrow \epsilon$
    end if
    for $i = j + 1$ to $N$ do
      $l_{ij} \leftarrow \frac{l_{ij}}{l_{jj}}$
    end for
  end for
```

CPU	Intel Xeon 2.93GHz (6 cores) x 2
Memory	54GB
MPI	MVAPICH2-1.9a2
X10	version 2.3.1
GotoBLAS2	version 1.13

Table 1: The Environment of TSUBAME 2.0

4. EVALUATION

4.1 Environment and Dataset

We run Link Propagation on TSUBAME 2.0 supercomputer at Tokyo Institute of Technology in Japan. Table 1 shows the environment of TSUBAME 2.0.

The dataset which used for performance evaluations is the subset of Twitter user network we obtained in 2012. Each of vertex represents one user and the edge $A \rightarrow B$ denotes that A follows B. The number of vertices and edges are shown in Table 2. This network data is distributed into multiple files. The number of files is 1179, and the size of each file is about 500MB.

4.2 Process of Experiment

The input of Link Propagation is the network in which some edges are removed. Then we evaluate the precision of Link Propagation; how precise the removed edges are recovered. The precision of link prediction is generally evaluated by AUC (Area Under the Curve), so we also use it.

The process of the experiment is as follows.

	# of vertices
Dataset A	594,455
Dataset B	2,650,983

Table 2: The Number of Vertices of Each Dataset

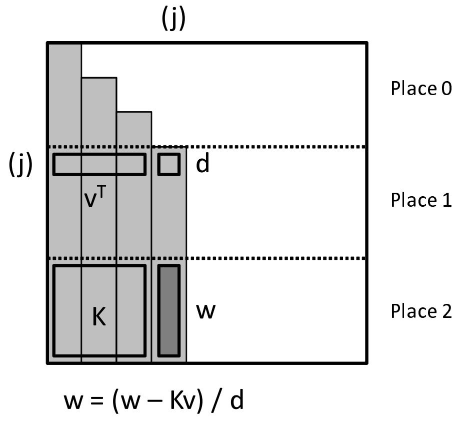

Figure 5: The computation step of parallelized MACD. For example, place 2 communicates with only the place which has v and d to calculate w.

1. Load the Twitter network data and remove some edges.
2. Make an adjacency matrix from the network data.
3. Apply Link Propagation to the adjacency matrix and get Core Matrix.
4. Get the link prediction scores of removed edges and some existing edges.
5. Calculate AUC from the prediction results.

4.3 Scalability

Scalability evaluation of Link Propagation is shown in Figure 6 and Figure 7. It shows the scalability up to 16 nodes with both dataset A and B. Since almost all process of Link Propagation are matrix operation, it seems that it gains the great benefit of parallelization.

4.4 Precision

Precision evaluation is shown in Figure 8. It shows constant precision even if approximation level in Cholesky decomposition is changed. This result is different from we expected, but the cause of it is not found yet.

5. RELATED WORKS

5.1 Related with Link Prediction

Song et.al. [6] proposed the link prediction algorithm based on proximity sketch and proximity embedding. This algorithm can dynamically predict the links of real time networks. But its scalability is worse than Link Propagation, since it does not aim for distributed environment.

Papadimitriou et.al. proposed FriendLink, which is a link prediction score like Katz$_\beta$ or RWR. They showed that calculating FriendLink is two times faster than Katz$_\beta$, but the precision was as low as RWR.

5.2 Related with Low Rank Approximation

Our Link Propagation uses Cholesky decomposition, but there are some more low rank approximation methods.

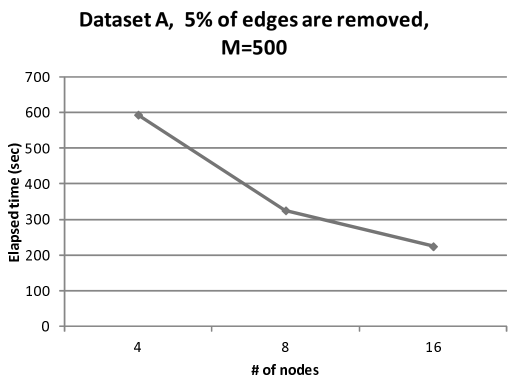

Figure 6: Scalability evaluation with dataset A.

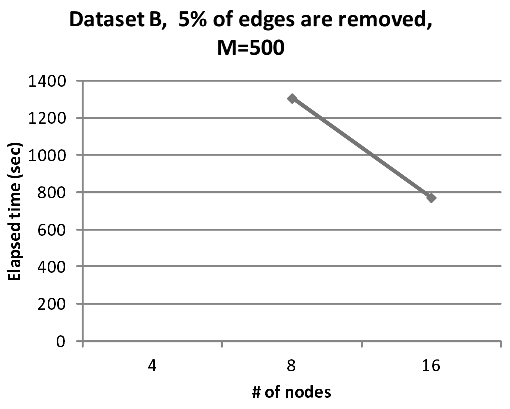

Figure 7: Scalability evaluation with dataset B.

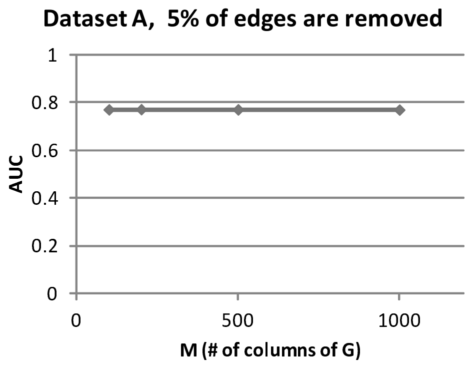

Figure 8: Precision evaluation with dataset A.

Drineas et.al. proposed CUR algorithm, where the columns and the rows of matrix A are sampled and construct sub matrices C, R, then A is decomposed as $A \approx CUR$. This algorithm is very fast but it cannot decompose like $A = LL^T$, so it is unsuited for Link Propagation.

6. CONCLUSION

We implemented the Link Propagation algorithm for large scale social networks in X10. At that time, we proposed original type of Cholesky decomposition (MACD) and showed that it was scalable and keeped the sparsity of the matrix. Actually, the scalability of Link Propagation became fine, but the precision was lower than expected. In the next step, we should investigate what is the cause of low precision.

7. REFERENCES

[1] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. Whaley. Scalapack: a portable linear algebra library for distributed memory computers – design issues and performance. *Computer Physics Communications*, 97(1-2):1–15, 1996.

[2] K. Goto and K. Milfeld. Texas advanced computing center - gotoblas2. `http://www.tacc.utexas.edu/tacc-projects/gotoblas2`.

[3] IBM. X10: Performance and productivity at scale. `http://x10-lang.org`.

[4] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? *WWW '10 Proceeding of the 19th international conference on World wide web*, pages 591–600, 2010.

[5] R. Raymond and H. Kashima. Fast and scalable algorithms for semi-supervised link prediction on static and dynamic graphs. *Machine Learning and Knowledge Discovery in Databases*, 6323:131–147, 2010.

[6] H. H. Song, T. W. Cho, V. Dave, Y. Zhang, and L. Qiu. Scalable proximity estimation and link prediction in online social networks. *IMC '09*, pages 322–335, 2009.