Efficient Processing Node Proximity via Random Walk with Restart

Bingqing Lv[1], Weiren Yu[2], Liping Wang[1], and Julie A McCann[2]

[1] East China Normal University, Shanghai, China
[2] Imperial College London, UK
lvbingqings@gmail.com {weiren.yu,j.mccann}@imperial.ac.uk
lipingwang@sei.ecnu.edu.cn

Abstract. Graph is a useful tool to model complicated data structures. One important task in graph analysis is assessing node proximity based on graph topology. Recently, Random Walk with Restart (RWR) tends to pop up as a promising measure of node proximity, due to its proliferative applications in *e.g.* recommender systems, and image segmentation. However, the best-known algorithm for computing RWR resorts to a large LU matrix factorization on an *entire* graph, which is cost-inhibitive. In this paper, we propose hybrid techniques to efficiently compute RWR. First, a novel divide-and-conquer paradigm is designed, aiming to convert the large LU decomposition into small triangular matrix operations recursively on several partitioned subgraphs. Then, on every subgraph, a "sparse accelerator" is devised to further reduce the time of RWR without any sacrifice in accuracy. Our experimental results on real and synthetic datasets show that our approach outperforms the baseline algorithms by at least one constant factor without loss of exactness.

1 Introduction

Finding proximities between objects based on graph topology is an important task in web data management. It has a wide range of applications, *e.g.* nearest neighbor search, image segmentation, and collaborative filtering. With the growing quantities of complex structured data, many fundamental problems have been naturally arising in graph analysis: How closely connected are two nodes in a graph? How to efficiently assess the closeness between two nodes?

To tackle these questions, recent years have witnessed growing attention to node-to-node proximities (*e.g.* [5, 6, 4, 8, 10, 11, 9]). Among them, Random Walk with Restart (RWR) has become a very popular one, which is originally patented by Tong *et al.* [5]. RWR is a PageRank-like node proximity based on a random surfer model. In comparison with other relevance measures, RWR has the following two benefits [5]: (1) it can globally capture the entire topology of a graph; (2) its proximity values can be used for ranking objects with respects to a certain query, as opposed to PageRank that is query-independent.

Prior Approaches. However, existing methods to compute RWR are less desirable. To the best of our knowledge, there exist two noteworthy methods for

RWR computation: Tong *et al.* [5] developed a closed form of RWR, converting the computation of RWR into a matrix inversion problem, which requires $O(n^3)$ time for assessing all RWR proximities for n^2 pairs of nodes in a graph. Very recently, for top-K search, Fujiwara *et al.* [1] has proposed an excellent algorithm called k-dash, which can be regarded as the state-of-the-art one for computing RWR. Unfortunately, their strategy involves a large LU matrix decomposition over an *entire* graph, which is still time-consuming. Therefore, it is very imperative to devise novel techniques for accelerating RWR computation.

Our Contributions. In this paper, hybrid optimization techniques are proposed for optimizing RWR computation. Different to the framework of [1] that performs large LU decomposition on an *entire* graph, we utilize a novel divide-and-conquer method, with the aim to convert large LU decomposition into small triangular matrix operations on some partitioned subgraphs in a recursive manner. This enables a substantial improvement on the computational time of RWR. Besides, we take advantage of the sparsity of triangular matrix multiplications with a node prioritizing strategy, and apply a fast matrix multiplication algorithm, to further accelerate RWR computation. Finally, we conduct extensive experiments on real and synthetic datasets to verify the high efficiency of our proposed algorithms against other baselines.

Organization. The rest of this paper is structured as follows. Section 2 revisits the related work. Section 3 overviews the background of RWR. Section 4 proposes our divide-and-conquer method, k-LU-RWR, for RWR acceleration, followed by some improved strategies in Section 4.3. Experiment results are reported in Section 5. Section 6 concludes the paper.

2 Related Work

RWR has been widely accepted as a useful measure of node proximity based on graph topology since the pioneering work of Tong *et al.* [5]. In that work, a singular vector decomposition (SVD) based algorithm, B_LIN, was also proposed for computing RWR, by taking advantaging of block structure of a graph. However, this method still involves an matrix inversion on very dense matrices, which is rather expensive, requiring cubic time in the number of nodes.

Later, Fujiwara *et al.* [1] proposed a fast top-K search based on RWR proximities. Their algorithm involves two strategies: first, they deployed a large LU decomposition on an *entire* graph for computing RWR; second, they used BFS tree estimation and devised a pruning technique to skip unnecessary scanning of nodes for top-K results. However, after LU decomposition, matrix inversions of **L** and **U** on the *entire* graph are still costly. In contrast, our work deploys a divide-and-conquer method to invert **L** and **U** recursively on small subgraphs, therefore achieving high computational efficiency.

Most recently, Yu *et al.* [7] have developed an incremental algorithm that supports link incremental updates for RWR on dynamical graphs. In comparison, our work focuses on efficient computations of RWR on static graphs.

3 Preliminaries

Notations. Table 1 lists the notations used throughout this paper.

Symbols	Definitions
n	total number of nodes in a graph
c	restarting probability, $0 \leq c \leq 1$
$\mathbf{A} = [a_{i,j}]$	column-normalized adjacency matrix
$\boldsymbol{p}_i = [p_{i,j}]$	$n \times 1$ RWR vector for query i, with $p_{i,j}$ the proximity of node j *w.r.t.* i
$\mathbf{W}$	$\mathbf{W} := \mathbf{I} - (1-c)\mathbf{A}$
$\boldsymbol{v}_i$	$n \times 1$ vector, whose i^{th} element is 1, and 0 otherwise
$\mathbf{L}$	lower triangular matrix
$\mathbf{U}$	upper triangular matrix
$\mathbf{I}$	identity matrix

Table 1. Symbols and Definitions

RWR Overview. The formal definition of RWR is as follows [5]:

$$\boldsymbol{p_i} = c\mathbf{A}\boldsymbol{p_i} + (1-c)\boldsymbol{v_i}. \tag{1}$$

Intuitively, Equation (1) suggests that a random particle starts to walk from a given query node i, and the particle iteratively transmits to its neighbor with the transition possibility in proportion to the edge weight between them. At each step, it has a probability c to return to the original node i until it reaches a steady state. The element $p_{i,j}$ in vector $\boldsymbol{p_i}$ refers to the probability of the particle finally stays at node j.

Based on Equation (1), we have the following closed-form of $\boldsymbol{p_i}$:

$$\boldsymbol{p_i} = (1-c)(\mathbf{I} - c\mathbf{A})^{-1}\boldsymbol{v_i} = (1-c)\mathbf{W^{-1}}\boldsymbol{v_i}. \tag{2}$$

The straightforward way of solving $\boldsymbol{p_i}$ in Equation (1) is to adopt an iterative paradigm: $\boldsymbol{p_i}^{(k+1)} = c\mathbf{A}\boldsymbol{p_i}^{(k)} + (1-c)\boldsymbol{v_i}$, where $\boldsymbol{p_i}^{(k)}$ is the k-th iterative RWR vector *w.r.t.* query node i. There are two stopping criteria for this iterative method: one is, given a threshold ϵ, to check whether the norm of the difference of two consecutive iterative RWR vectors is below ϵ, *i.e.*, $\|\boldsymbol{p_i}^{(k+1)} - \boldsymbol{p_i}^{(k)}\| \leq \epsilon$; the other is, given the total number of iterations K, to check whether the number of iterations increasingly reaches K. However, both of these criteria may sacrifice a little accuracy, as compared with non-iterative methods. Therefore, in this paper our optimization techniques for RWR are based on non-iterative framework.

LU Factorization. Regarding the non-iterative methods for RWR, LU decomposition is the best-known method, which is based on a closed-form of $\boldsymbol{p_i}$, as shown in Equation (2). However, directly calculating $\mathbf{W^{-1}}$ requires high computation time as the inversion matrix could be dense even though $\mathbf{W}$ is sparse in most cases. To deal with this problem, we take advantage of the Crout's algorithm [2] to do LU decomposition. Consequently, we can compute the inversions of $\mathbf{L}$ and $\mathbf{U}$ instead, namely, $\mathbf{W^{-1}} = \mathbf{U^{-1}L^{-1}}$.

Algorithm 1 k-LU-RWR Algorithm

```
Input:
    $\mathbf{A}$ : the normalized adjacency matrix
    $n$ : total number of nodes
    $i$ : query node
    $c$ : restarting probability
    $k$ : partitioning times
Output:
    $\boldsymbol{p_i}$: ranking vector of node $i$
1: Compute $\mathbf{W} = \mathbf{I} - c\mathbf{A}$
2: Do LU decomposition for $\mathbf{W} = \mathbf{LU}$ according to Crout's algorithm
3: $\mathbf{L^{-1}}$=$recInvLU(\mathbf{L}, n, k)$
4: $\mathbf{U^{-1}}$=$recInvLU(\mathbf{U}, n, k)$
5: $\boldsymbol{p_i} = (1 - c)\mathbf{U}^{-1}\mathbf{L}^{-1}\boldsymbol{v_i}$
6: return $\boldsymbol{p_i}$
```

4 Our Solution

4.1 A Divide-and-Conquer Strategy for RWR

To meet the challenges raised by RWR, we propose an algorithm named k-LU-RWR shown in Algorithm 1. In consideration of Equation (2), we pre-compute and store $\mathbf{W^{-1}}$ offline which is from step 1 to step 4 in the algorithm. When a query node i comes, we simply calculate the proximities $\boldsymbol{p_i}$ online by only two multiplication operations of matrix-vector according to step 5.

Now let us concentrate on step 3 and step 4. As we have decided in Section 3, we take advantage of LU decomposition on $\mathbf{W}$. However, directly utilizing LU decomposition still requires to compute inversion matrices for both $\mathbf{L}$ and $\mathbf{U}$, so we apply following optimization strategies. We partition $\mathbf{L}$ and $\mathbf{U}$ into four parts, as presented in Figure 1. After that, we compute the matrix inversions of $\mathbf{L}$ and $\mathbf{U}$ according to Equation (3). Specifically, we do matrix inversions on $\mathbf{L_{1,1}}$, $\mathbf{L_{2,2}}$, $\mathbf{U_{1,1}}$ and $\mathbf{U_{2,2}}$, referred to as *triangle inversion*, and matrix multiplications on $\mathbf{L_{2,1}}$, $\mathbf{U_{1,2}}$ part, referred to as *rectangle multiplication.* When finished, we merge the block matrices into $\mathbf{L^{-1}}$ and $\mathbf{U^{-1}}$.

$$\mathbf{L^{-1}} = \begin{bmatrix} \mathbf{L_{1,1}^{-1}} & \mathbf{0} \\ -\mathbf{L_{2,2}^{-1}L_{2,1}L_{1,1}^{-1}} & \mathbf{L_{2,2}^{-1}} \end{bmatrix} \quad \mathbf{U^{-1}} = \begin{bmatrix} \mathbf{U_{1,1}^{-1}} & -\mathbf{U_{1,1}^{-1}U_{1,2}U_{2,2}^{-1}} \\ \mathbf{0} & \mathbf{U_{2,2}^{-1}} \end{bmatrix} \tag{3}$$

From Figure 1, it is clear that $\mathbf{L_{1,1}}$, $\mathbf{L_{2,2}}$ remain to be lower triangular matrices, and $\mathbf{U_{1,1}}$, $\mathbf{U_{1,2}}$ remain to be upper triangular matrices. This inspires us to do the partition and inversion procedures recursively, as represented in Figure 2. So we further devise a recursive algorithm recInvLU in Algorithm 2 to calculate triangle matrix inversions in step 3 and step 4 of Algorithm 1.

4.2 Time Complexity of k-LU-RWR

In the pre-computation stage, i.e. step 1 to step 4, the main time cost includes LU decomposition and matrix inversion. In LU decomposition part, we adopt

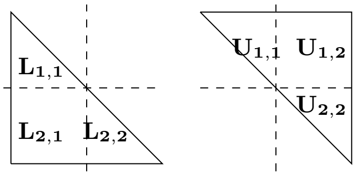

Fig. 1. **L** and **U** Partitions

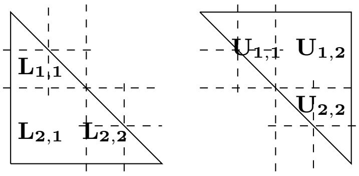

Fig. 2. Recursive **L** and **U** Partitions

Algorithm 2 recInvLU($\mathbf{M}$, n, k)

```
Input:
    M : the lower or upper triangular matrix after LU factorization
    n : size of matrix M
    k : partitioning times
Output:
    $\mathbf{M^{-1}}$: the inversion matrix of M
if $k = 0$ then
    $\mathbf{M^{-1}} = inverse(\mathbf{M})$
    return $\mathbf{M^{-1}}$
else
    Partition M into $\mathbf{M_{1,1}}$, $\mathbf{M_{1,2}}$, $\mathbf{M_{2,1}}$, and $\mathbf{M_{2,2}}$
    $\mathbf{M_{1,1}^{-1}}$=recInvLU($\mathbf{M_{1,1}}$, $\lfloor \frac{n}{2} \rfloor$, $k-1$)
    $\mathbf{M_{2,2}^{-1}}$=recInvLU($\mathbf{M_{2,2}}$, $\lceil \frac{n}{2} \rceil$, $k-1$)
    if M is lower triangular matrix then
        $\mathbf{M_{2,1}^{-1}} = -\mathbf{M_{2,2}^{-1}M_{2,1}M_{1,1}^{-1}}$
        Merge $\mathbf{M_{1,1}^{-1}}$, $\mathbf{M_{2,2}^{-1}}$ and $\mathbf{M_{2,1}^{-1}}$ into $\mathbf{M^{-1}}$
    else
        $\mathbf{M_{1,2}^{-1}} = -\mathbf{M_{1,1}^{-1}M_{1,2}M_{2,2}^{-1}}$
        Merge $\mathbf{M_{1,1}^{-1}}$, $\mathbf{M_{2,2}^{-1}}$ and $\mathbf{M_{1,2}^{-1}}$ into $\mathbf{M^{-1}}$
    end if
end if
```

Crout's algorithm, whose theoretical time complexity is $O(n^3)$. Similarly, in the inversion part, if we inverse **L** and **U** directly, the time complexity is $O(n^3)$ too. However, we applied partitioning strategy and the time complexity is given by Equation (4), where $T(MM(\frac{n}{2}))$ represents the cost of *rectangle multiplication* part in Algorithm 2.

$$\begin{aligned} T(recInvLU(n)) &= 2T(recInvLU(\frac{n}{2})) + 2T(MM(\frac{n}{2})) \\ &= n\log(n) + 2T(MM(\frac{n}{2})) \end{aligned} \tag{4}$$

In the query stage, we simply do two multiplications of matrix-vector. For this reason, the query response is nearly real-time.

4.3 Further Improvement of k-LU-RWR

From Equation (4), we know that the time of RWR mainly depends on the matrix multiplications. A straightforward implementation of the matrix multiplications needs cubic time in the number of nodes. We now introduce two enhanced versions for accelerating the matrix multiplications: (1) We can adopt sparse matrix storage data structure. Since the adjacency matrix **A** is often sparse, we can use the reordering strategy [1] to keep the sparsity of LU decomposition. This reordering strategy has a good performance in practice. However, when matrices are becoming dense, the worst case time is still $O(n^3)$. (2) We can also apply Strassen's algorithm [3] to reduce the time of the matrix multiplications to $O(n^{\log_2 7})$. Combining these two methods together, it requires $O(n^{\log_2 7})$ time in total for computing RWR, while eliminating unnecessary multiplications by filtering zero entries.

5 Performance Evaluation

5.1 Experiments Settings

We set the restart probability $c = 0.9$, as previously used in [5].

We conduct a set of experiments on the value of partition times k, to see how it effects on the experiment performance. To verify the effect of the Reorder strategy, we also do experiments on k-LU-RWR without reorder procedure called Un-LU-RWR, compared with k-LU-RWR, to contrast ratio of the number of nonzero entries in L^{-1} and U^{-1} to the edges in matrices, which indicates the time and storage costs we preserve. Moreover, the proposed algorithm is compared with NB_LIN [5] and k-dash [1] in terms of pre-computation time to show the efficiency of our algorithm. In k-dash, we compute proximities of top-n nodes for fair comparison. Besides, we do experiments to show the high response time on queries of k-LU-RWR.

We use real and synthetic datasets. All experiments were conducted on the machine with 2.5GHz CPU and 4.00GB main memory. Our algorithms are implemented in C++. The details of datasets are listed in Table 2.

Datasets	Number of nodes	Number of edges	Type
bcsstk25	$\approx 15K$	$\approx 252K$	real
bcsstk31	$\approx 36K$	$\approx 1,181K$	binary
bauru5727	$\approx 40K$	$\approx 145K$	binary
crystm03	$\approx 25K$	$\approx 584K$	real
pcrystk03	$\approx 25K$	$\approx 1,751K$	binary
synthetic10000	$10k$	$200k$	real
synthetic15000	$15k$	$250k$	real
synthetic20000	$20k$	$1000k$	real

Table 2. Datasets

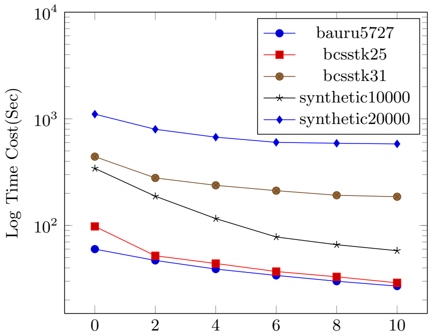

Fig. 3. Different k values of k-LU-RWR

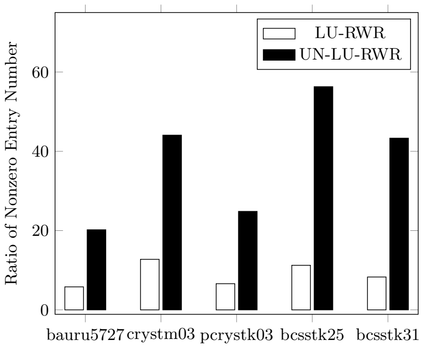

Fig. 4. Effect of Reorder Strategy

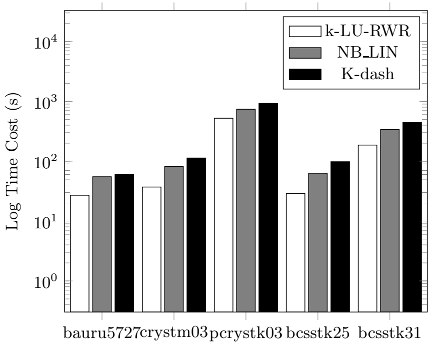

Fig. 5. Pre-computation Cost on RealData

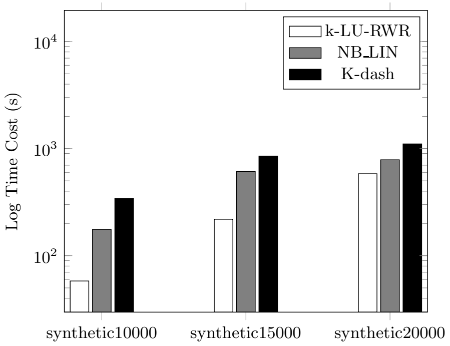

Fig. 6. Pre-computation Cost on SyntheticData

5.2 Experiment Results

We first conduct experiments on how the value of k influences on the efficiency of k-LU-RWR. The results are shown in Figure 3. When k varies from 0 to 10, we see the time costs decrease gradually on both real datasets and synthetic datasets. For the case $k = 0$, we directly calculate the inversions of $\mathbf{L}$ and $\mathbf{U}$ without partition technique. When k increases, the falling speeds differ on each dataset due to the different architectures, but the performance changes little when k grows to a certain degree. When $k = 10$, it preserves about 50% pre-computation cost with regard to $k = 0$.

To demonstrate the sparsity of matrices after applying Reorder strategy, we show the ratio of non-zero element numbers to matrix edges in Figure 4. From the figure, we can see almost 70% storage costs are saved by reordering the elements. It also indicates the storages and computation costs we saved by adopting Reorder strategy and sparse storage.

We compare pre-computation costs between k-LU-RWR, NB_LIN and k-dash. The results on real datasets are shown in Figure 5 and the results on synthetic datasets are shown in Figure 6. Figure 5 demonstrates that our algorithm preserves about 50% pre-computation w.r.t NB_LIN and about 70% pre-computation w.r.t k-dash. Figure 6 shows our algorithm saves over 50% pre-computation cost w.r.t NB_LIN and k-dash. There are mainly two reasons for the enhancements: (1) our algorithm adopts the idea of divide and conquer and takes advantage of a sparse manner so that it skips unnecessary calculations by ignoring zero entries; (2) by utilizing a fast matrix multiplication algorithm Strassen's Algorithm we saved one multiplication operation in each iteration by partitioning k times and reduce the time complexity.

We also do experiments to verify the efficiency of the query stage, in which we perform two matrix-vector multiplications. The query time on each datasets is a few hundred milliseconds, as expected.

6 Conclusion

This paper addressed the problem of efficiently computing RWR proximities based on graph topology. We first devised a divide-and-conquer paradigm to recursively do LU factorization over small subgraphs. Then, by taking advantage of sparsity of triangular matrix structure, we further accelerated RWR computation via fast matrix multiplication. Finally, we conducted extensive empirical results using real and synthetic dataset, showing the superiority of our proposed algorithm against the baselines in terms of computational time.

References

1. Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa. Fast and exact top-k search for Random Walk with Restart. *PVLDB*, 5:442–453, 2012.
2. W. H. Press. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press, 2007.
3. V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.
4. Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. PathSim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB*, 4:992–1003, 2011.
5. H. Tong, C. Faloutsos, and J.-Y. Pan. Fast Random Walk with Restart and its applications. In *ICDM*, pages 613–622, 2006.
6. W. Yu, J. Le, X. Lin, and W. Zhang. On the efficiency of estimating Penetrating Rank on large graphs. In *SSDBM*, pages 231–249, 2012.
7. W. Yu and X. Lin. IRWR: Incremental Random Walk with Restart. In *SIGIR*, pages 1017–1020, 2013.
8. W. Yu, X. Lin, and W. Zhang. Towards efficient SimRank computation on large networks. In *ICDE*, pages 601–612, 2013.
9. W. Yu, X. Lin, and W. Zhang. Fast incremental SimRank on link-evolving graphs. In *ICDE*, pages 304–315, 2014.
10. W. Yu, X. Lin, W. Zhang, L. Chang, and J. Pei. More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks. *PVLDB*, 7(1):13–24, 2013.
11. W. Yu, X. Lin, W. Zhang, Y. Zhang, and J. Le. SimFusion+: Extending simfusion towards efficient estimation on large and dynamic networks. In *SIGIR*, pages 365–374, 2012.