Fast Random Walk with Restart and Its Applications

Hanghang Tong
Carnegie Mellon University
htong@cs.cmu.edu

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

Jia-Yu Pan
Carnegie Mellon University
jypan@cs.cmu.edu

Abstract

How closely related are two nodes in a graph? How to compute this score quickly, on huge, disk-resident, real graphs? Random walk with restart (RWR) provides a good relevance score between two nodes in a weighted graph, and it has been successfully used in numerous settings, like automatic captioning of images, generalizations to the "connection subgraphs", personalized PageRank, and many more. However, the straightforward implementations of RWR do not scale for large graphs, requiring either quadratic space and cubic pre-computation time, or slow response time on queries.

We propose fast solutions to this problem. The heart of our approach is to exploit two important properties shared by many real graphs: (a) linear correlations and (b) block-wise, community-like structure. We exploit the linearity by using low-rank matrix approximation, and the community structure by graph partitioning, followed by the Sherman-Morrison lemma for matrix inversion. Experimental results on the Corel image and the DBLP dabasets demonstrate that our proposed methods achieve significant savings over the straightforward implementations: they can save several orders *of magnitude in pre-computation and storage cost, and they achieve up to 150x speed up with 90%+ quality preservation.*

1 Introduction

Defining the relevance score between two nodes is one of the fundamental building blocks in graph mining. One very successful technique is based on random walk with restart (RWR). RWR has been receiving increasing interest from both the application and the theoretical point of view (see Section (5) for detailed review). An important research challenge is its speed. especially for large graphs.

Mathematically, RWR requires a matrix inversion. There are two straightforward solutions, none of which is scalable for large graphs: The first one is to pre-compute and store the inversion of a matrix ("*PreCompute*" method); the second one is to compute the matrix inversion on the fly, say, through power iteration ("*OnTheFly*" method). The first method is fast on query time, but prohibitive on space (quadratic on the number of nodes on the graph), while the second is slow on query time.

Here we propose a novel solution to this challenge. Our approach, B_LIN, takes the advantage of two properties shared by many real graphs: (a) the block-wise, community-like structure, and (b) the linear correlations across rows and columns of the adjacency matrix. The proposed method carefully balances the off-line pre-processing cost (both the CPU cost and the storage cost), with the response quality (with respect to both the accuracy and the response time). Compared to *PreCompute*, it only requires pre-computing and storing the low-rank approximation of a large but sparse matrix, and the inversion of some small size matrices. Compared with *OnTheFly*, it only need a few matrix-vector multiplication operations in on-line response process.

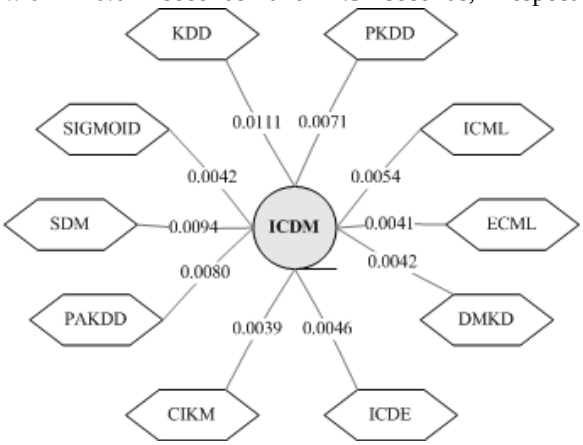
The main contributions of the paper are as follows:

- A novel, fast, and practical solution (B_LIN and its derivative, NB_LIN);
- Theoretical justification and analysis, giving an error bound for NB_LIN;
- Extensive experiments on several typical applications, with real data.

The proposed method is operational, with careful design and numerous optimizations. Our experimental results show that, in general, it preserves 90%+ quality, while (a) saves several orders of magnitude of pre-computation and storage cost over *PreCompute*, and (b) it achieves up to 150x speedup on query time over *OnTheFly*. For the DBLP author-conference dataset, with light pre-computational and storage cost, it achieves up to 1,800x speedup with no quality loss. Figure (1-a) shows some results for the auto-captioning application as in [22]. Figure (1-b) shows some results for the neighborhood formation application as in [25].

'Jet' 'Plane' 'Runway' 'Texture' 'Candy' 'Background'

(a) Automatic image captioning. The proposed method and *OnTheFly* output the same result within 0.04 seconds and 4.5 seconds, respectively.



(b) Neighborhood formulation. Find the 10 most related conferences for ICDM. The proposed method and *OnTheFly* output the same result within 0.013 seconds and 23.97 seconds, respectively.

Figure 1. Application examples by RWR

The rest of the paper is organized as follows: the proposed method is presented in Section 2; the justification and the analysis are provided in Section 3. The experimental results are presented in Section 4. The related work is given in Section 5. Finally, we conclude the paper in Section 6.

2 Fast RWR

2.1 Preliminary

Table 1 gives a list of symbols used in this paper.

Random walk with restart is defined as equation (1) [22]: consider a random particle that starts from node i. The particle iteratively transmits to its neighborhood with the probability that is proportional to their edge weights. Also at each step, it has some probability c to return to the node i. The relevance score of node j wrt node i is defined as the steady-state probability $r_{i,j}$ that the particle will finally stay at node j [22].

$$\vec{r}_i = c\tilde{\mathbf{W}}\vec{r}_i + (1-c)\vec{e}_i \tag{1}$$

Equation (1) defines a linear system problem, where $\vec{r}_i$ is determined by:

$$\begin{aligned} \vec{r}_i &= (1-c)(I - c\tilde{\mathbf{W}})^{-1}\vec{e}_i \\ &= (1-c)\mathbf{Q}^{-1}\vec{e}_i \end{aligned} \tag{2}$$

The relevance score defined by RWR has many good properties: compared with those pair-wise metrics, it can capture the global structure of the graph [14]; compared with those traditional graph distances (such as shortest path, maximum flow etc), it can capture the multi-facet relationship between two nodes [26].

One of the most widely used ways to solve random walk with restart is the iterative method, iterating the equation (1) until convergence, that is, until the L_2 norm of successive estimates of $\vec{r}_i$ is below our threshold ξ_1, or a maximum iteration step m is reached. In the paper, we refer it as *OnTheFly* method. *OnTheFly* does not require pre-computation and additional storage cost. Its on-line response time is linear to the iteration number and the number of edges[1], which might be undesirable when (near) real-time response is a crucial factor while the dataset is large. A nice observation of [25] is that the distribution of $\vec{r}_i$ is highly skewed. Based on this observation, combined with the factor that many real graphs has block-wise/community structure, the authors in [25] proposed performing RWR only on the partition that contains the starting point i (method *Blk*). However, for all data points outside the partition, $r_{i,j}$ is simply set 0. In other words, *Blk* outputs a local estimation of $\vec{r}_i$.

On the other hand, it can be seen from equation (2) that the system matrix $\mathbf{Q}$ defines all the steady-state probabilities of random walk with restart. Thus, if we can pre-compute and store $\mathbf{Q}^{-1}$, we can get $\vec{r}_i$ real-time (We refer to this method as *PreCompute*). However, pre-computing and storing $\mathbf{Q}^{-1}$ is impractical when the dataset is large, since it requires quadratic space and cubic pre-computation [2].

On the other hand, linear correlations exist in many real graphs, which means that we can approximate $\tilde{\mathbf{W}}$ by low-rank approximation. This property allows us to approximate $\mathbf{Q}^{-1}$ very efficiently. Moreover, this enables a global estimation of $\vec{r}_i$, unlike the local estimation obtained by *Blk*. However, due to the low rank approximation, such kind of estimation is conducted at a coarse resolution.

[1] Here, we store $\tilde{\mathbf{W}}$ in a sparse format.

[2] Even if we use *OnTheFly* to compute each column of $\mathbf{Q}^{-1}$, the pre-computation cost is still linear to the number of node n.

Table 1. Symbols

Symbol	Definition
$\mathbf{W} = [w_{i,j}]$	the weighted graph, $1 \leq i, j \leq n$
$\tilde{\mathbf{W}}$	the normalized weighted matrix associated with $\mathbf{W}$
$\tilde{\mathbf{W}}_1$	the within-partition matrix associated with $\tilde{\mathbf{W}}$
$\tilde{\mathbf{W}}_2$	the cross-partition matrix associated with $\tilde{\mathbf{W}}$
$\mathbf{Q}$	the system matrix associated with $\mathbf{W}$: $\mathbf{Q} = \mathbf{I} - c\tilde{\mathbf{W}}$
$\mathbf{D}$	$n \times n$ matrix, $D_{i,i} = \sum_j w_{i,j}$ and $D_{i,j} = 0$ for $i \neq j$
$\mathbf{U}$	$n \times t$ node-concept matrix
$\mathbf{S}$	$t \times t$ concept-concept matrix
$\mathbf{V}$	$t \times n$ concept-node matrix
$\mathbf{0}$	a block matrix, whose elements are all zeros
$\vec{e}_i$	$n \times 1$ starting vector, the i^{th} element 1 and 0 for others
$\vec{r}_i = [r_{i,j}]$	$n \times 1$ ranking vector, $r_{i,j}$ is the relevance score of node j wrt node i
c	the restart probability, $0 \leq c \leq 1$
n	the total number of the nodes in the graph
k	the number of partitions
t	the rank of low-rank approximation
m	the maximum iteration number
ξ_1	the threshold to stop the iteration process
ξ_2	the threshold to sparse the matrix

2.2 Algorithm

In summary, the skewed distribution of $\vec{r}_i$ and the block-wise structure of the graph lead to a local/fine resolution estimation; the linear correlations of the graph lead to a global/coarse resolution estimation. In this paper, we combine these two properties in a unified manner. The proposed algorithm, B_LIN is shown in table (2). A pictorical description of B_LIN is given in figure (2).

$$\tilde{\mathbf{W}}_1 = \begin{pmatrix} \tilde{\mathbf{W}}_{1,1} & \mathbf{0} & ... & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{W}}_{1,2} & ... & \mathbf{0} \\ ... & ... & ... & ... \\ \mathbf{0} & ... & \mathbf{0} & \tilde{\mathbf{W}}_{1,k} \end{pmatrix} \quad (3)$$

$$\mathbf{Q}_1^{-1} = \begin{pmatrix} \mathbf{Q}_{1,1}^{-1} & \mathbf{0} & ... & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_{1,2}^{-1} & ... & \mathbf{0} \\ ... & ... & ... & ... \\ \mathbf{0} & ... & \mathbf{0} & \mathbf{Q}_{1,k}^{-1} \end{pmatrix} \quad (4)$$

2.3 Normalization on W

B_LIN takes the normalized matrix $\tilde{\mathbf{W}}$ as the input. There are several ways to normalize the weighted matrix $\mathbf{W}$. The most natural way might be by row normalization [22]. Complementarily, the authors in [27] propose using the normalized graph Lapalician ($\tilde{\mathbf{W}} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$). In [26], the authors also propose penalizing the famous nodes before row normalization for social network.

It should be pointed out that all the above normalization methods can be fitted into the proposed B_LIN. However, in this paper, we will focus on the normalized graph Laplacian[3] for the following reasons:

- For real applications, these normalization methods often lead to very similar results. (For cross-media correlation discovery, our experiments demonstrate that normalized graph Laplacian actually outperforms the row normalization method, which is originally proposed by the authors in [22]
- Unlike the other two methods, normalized graph Laplacian outputs the symmetric relevance score (that is $r_{i,j} = r_{j,i}$), which is a desirable property for some applications.
- The normalized graph Laplacian is symmetric, and it leads to a symmetric $\mathbf{Q}_1$, which will save 50% storage cost.
- It might be difficult to develop an error bound for B_LIN in the general case. However, as we will show

[3] It should be pointed out that strictly speaking, $\vec{r}_i$ is no longer a probability distribution. However, for all the applications we cover in this paper, it does not matter since what we need is a relevance score. On the other hand, we can always normalized $\vec{r}_i$ to get a probability distribution.

Table 2. B_LIN

Input: The normalized weighted matrix $\tilde{\mathbf{W}}$ and the starting vector $\vec{e}_i$
Output: The ranking vector $\vec{r}_i$
Pre-Computational Stage(Off-Line):
p1. Partition the graph into k partitions by METIS [19];
p2. Decompose $\tilde{\mathbf{W}}$ into two matrices: $\tilde{\mathbf{W}} = \tilde{\mathbf{W}}_1 + \tilde{\mathbf{W}}_2$ according to the partition result, where $\tilde{\mathbf{W}}_1$ contains all within-partition links and $\tilde{\mathbf{W}}_2$ contains all cross-partition links;
p3. Let $\tilde{\mathbf{W}}_{1,i}$ be the i^{th} partition, denote $\tilde{\mathbf{W}}_1$ as equation(3);
p4. Compute and store $\mathbf{Q}_{1,i}^{-1} = (\mathbf{I} - c\tilde{\mathbf{W}}_{1,i})^{-1}$ for each partition i;
p5. Do low-rank approximation for $\tilde{\mathbf{W}}_2 = \mathbf{USV}$;
p6. Define $\mathbf{Q}_1^{-1}$ as equation (4). Compute and store $\tilde{\mathbf{\Lambda}} = (\mathbf{S}^{-1} - c\mathbf{V}\mathbf{Q}_1^{-1}\mathbf{U})^{-1}$.
Query Stage (On-Line):
q1. Output $\vec{r_i} = (1-c)(\mathbf{Q}_1^{-1}\vec{e_i} + c\mathbf{Q}_1^{-1}\mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{V}\mathbf{Q}_1^{-1}\vec{e_i})$.

in Section 3.3, it is possible to develop an error bound for the simplified version (NB_LIN) of B_LIN, which also benefits from the symmetric property of the normalized graph Laplacian.

2.4 Partition number k: case study

The partition number k balances the complexity of $\tilde{\mathbf{W}}_1$ and $\tilde{\mathbf{W}}_2$. We will evaluate different values for k in the experiment section. Here, we investigate two extreme cases of k.

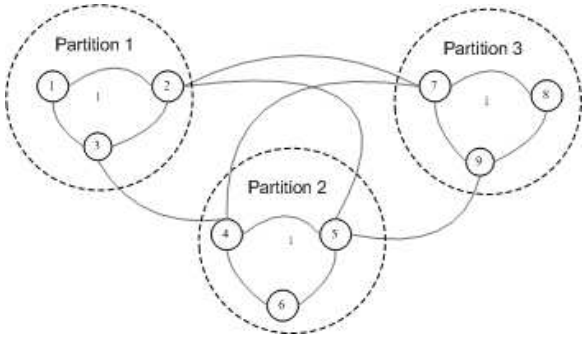
First, if $k = 1$, we have $\tilde{\mathbf{W}}_1 = \tilde{\mathbf{W}}$ and $\tilde{\mathbf{W}}_2 = \mathbf{0}$. Then, B_LIN is just equivalent to the *PreCompute* method.

On the other hand, if $k = n$, we have $\tilde{\mathbf{W}}_1 = \mathbf{0}$ and $\tilde{\mathbf{W}}_2 = \tilde{\mathbf{W}}$. In this case, $\mathbf{Q}_1 = \mathbf{I}$ and we have the following simplified version of B_LIN as in table(3). We refer it as NB_LIN.

Table 3. NB_LIN

Input: The normalized weighted matrix $\tilde{\mathbf{W}}$ and the starting vector $\vec{e}_i$
Output: The ranking vector $\vec{r}_i$
Pre-Computational Stage(Off-Line):
p1. Do low-rank approximation for $\tilde{\mathbf{W}} = \mathbf{USV}$;
p2. Compute and store $\tilde{\mathbf{\Lambda}} = (\mathbf{S}^{-1} - c\mathbf{VU})^{-1}$.
Query Stage (On-Line):
q1. Output $\vec{r_i} = (1-c)(\vec{e_i} + c\mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{V}\vec{e_i})$.

An application of random walk with restart is neighborhood formulation in the bipartite graph [25]. Suppose there



(a) Original weighted graph, consisting of 3 partitions, which are indicated by the dash circles

$$\begin{pmatrix} \\ \bar{W} \\ \\ \end{pmatrix} = \begin{pmatrix} \bar{W}_{1,1} & 0 & 0 \\ 0 & \bar{W}_{1,2} & 0 \\ 0 & 0 & \bar{W}_{1,3} \end{pmatrix} + \begin{pmatrix} \\ U \\ \\ \end{pmatrix} (S) \begin{pmatrix} \\ V \\ \\ \end{pmatrix}$$

(b) Decompose original weighted graph into within-partition matrix ($\tilde{\mathbf{W}}_1$), which is block-diagonal, and cross-partition matrix, which is approximated by low-rank approximation ($\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}$)

$$\begin{pmatrix} \\ I - c\bar{W} \\ \\ \end{pmatrix}^{-1} = \begin{Bmatrix} \begin{pmatrix} Q_{1,1}^{-1} & 0 & 0 \\ 0 & Q_{1,2}^{-1} & 0 \\ 0 & 0 & Q_{1,3}^{-1} \end{pmatrix} \\ + \\ c \begin{pmatrix} Q_{1,1}^{-1} & 0 & 0 \\ 0 & Q_{1,2}^{-1} & 0 \\ 0 & 0 & Q_{1,3}^{-1} \end{pmatrix} \begin{pmatrix} \\ U \\ \\ \end{pmatrix} (\bar{\Lambda}) \begin{pmatrix} \\ V \\ \\ \end{pmatrix} \begin{pmatrix} Q_{1,1}^{-1} & 0 & 0 \\ 0 & Q_{1,2}^{-1} & 0 \\ 0 & 0 & Q_{1,3}^{-1} \end{pmatrix} \end{Bmatrix}$$

(c) Approximate the inverse of $(\mathbf{I} - c\tilde{\mathbf{W}})$ by the inversion of a few small size matrices($\mathbf{Q}_{1,1}$, $\mathbf{Q}_{1,2}$, $\mathbf{Q}_{1,3}$ and $\tilde{\mathbf{\Lambda}}$), which can be pre-computed and stored more efficiently.

Figure 2. A pictorical description of B_LIN

are n_1 and n_2 nodes for each type of objects in the bipartite graph; $\mathbf{M}$ is the $n_1 \times n_2$ bipartite matrix. The normalized weighted matrix, the starting vector and the ranking vector have the following format:

$$\tilde{\mathbf{W}} = \begin{pmatrix} \mathbf{0} & \mathbf{M} \\ \mathbf{M^T} & \mathbf{0} \end{pmatrix} \quad \vec{r}_i = \begin{pmatrix} \vec{r}_{i,1} \\ \vec{r}_{i,2} \end{pmatrix} \quad \vec{e}_i = \begin{pmatrix} \vec{e}_{i,1} \\ \vec{e}_{i,2} \end{pmatrix} \qquad (5)$$

As a direct application of NB_LIN, we have the following fast algorithm (BB_LIN) for one class of bipartite graph when $n_1 \gg n_2$ as in table (4)

2.5 Low-rank approximation on $\tilde{\mathbf{W}}_2$

One natural choice to do low-rank approximation on $\tilde{\mathbf{W}}_2$ is by eigen-value decomposition[4]:

[4]if the other two normalization methods are used, we can do singular vector decomposition instead.

Table 4. BB_LIN

Input: The normalized weighted matrix $\tilde{\mathbf{W}}$ and the starting vector $\vec{e}_i$ as equation(5)
Output: The ranking vector $\vec{r}_i$ as equation(5)
Pre-Computational Stage(Off-Line):
p1. Compute and store $\tilde{\mathbf{\Lambda}} = (\mathbf{I} - c^2\mathbf{M}^T\mathbf{M})^{-1}$;
Query Stage (On-Line):
q1. $\vec{r}_{i,1} = (1-c)(\vec{e}_{i,1} + c^2\mathbf{M}\tilde{\mathbf{\Lambda}}\mathbf{M}^T\vec{e}_{i,1} + c\mathbf{M}\tilde{\mathbf{\Lambda}}\vec{e}_{i,2})$
q2. $\vec{r}_{i,2} = (1-c)(c\tilde{\mathbf{\Lambda}}\mathbf{M}^T\vec{e}_{i,1} + \tilde{\mathbf{\Lambda}}\vec{e}_{i,2})$
q3. Output $\vec{r_i} = (\vec{r}_{i,1}, \vec{r}_{i,2})^T$.

$$\tilde{\mathbf{W}}_2 = \mathbf{U}\mathbf{S}\mathbf{U}^T \tag{6}$$

where each column of $\mathbf{U}$ is the eigen-vector of $\tilde{\mathbf{W}}_2$ and $\mathbf{S}$ is a diagonal matrix, whose diagonal elements are eigen-values of $\tilde{\mathbf{W}}_2$.

The advantage of eigen-value decomposition is that it is 'optimal' in terms of reconstruction error. Also, since $\mathbf{V} = \mathbf{U}^T$ in this situation, we can save 50% storage cost. However, one potential problem is that it might lose the sparsity of original matrix $\tilde{\mathbf{W}}_2$. Also, when $\tilde{\mathbf{W}}_2$ is large, doing eigen-value decomposition itself might be time-consuming.

To address this issue, in this paper, we also propose the following heuristic to do low-rank approximation as in table (5). Its basic idea is that, firstly, construct $\mathbf{U}$ by partitioning $\tilde{\mathbf{W}}_2$; and then use the projection of $\tilde{\mathbf{W}}_2$ on the sub-space spanned by the columns of $\mathbf{U}$ as the low-rank approximation.

Table 5. Low Rank Approximation by Partition

Input: The cross-partition matrix $\tilde{\mathbf{W}}_2$ and t
Output: Low rank approximation of $\tilde{\mathbf{W}}_2$: $\mathbf{U}, \mathbf{S}, \mathbf{V}$
1. Partition $\tilde{\mathbf{W}}_2$ into t partitions;
2. Construct an $n \times t$ matrix $\mathbf{U}$. The i^{th} column of $\mathbf{U}$ is the sum of all the columns of $\tilde{\mathbf{W}}_2$ that belong to the i^{th} partition;
3. Compute $\mathbf{S} = (\mathbf{U}^T\mathbf{U})^{-1}$;
4. Compute $\mathbf{V} = \mathbf{U}^T\tilde{\mathbf{W}}_2$.

3 Justification and Analysis

3.1 Correctness

Here, we present a brief proof of the proposed algorithms

3.1.1 B_LIN

Lemma 1 *If* $\tilde{\mathbf{W}} = \tilde{\mathbf{W}}_1 + \mathbf{U}\mathbf{S}\mathbf{V}$ *holds, B_LIN outputs exactly the same result as* PreCompute.

Proof: Since $\tilde{\mathbf{W}}_1$ is a block-diagonal matrix. Based on equation (3) and (4), we have

$$(\mathbf{I} - c\tilde{\mathbf{W}}_1)^{-1} = \mathbf{Q}_1^{-1} \tag{7}$$

Then, based on the Sherman-Morrison lemma [23], we have:

$$\begin{aligned} \tilde{\mathbf{\Lambda}} &= (\mathbf{S}^{-1} - c\mathbf{V}\mathbf{Q}_1^{-1}\mathbf{U})^{-1} \\ (\mathbf{I} - c\tilde{\mathbf{W}})^{-1} &= (\mathbf{I} - c\tilde{\mathbf{W}}_1 - c\mathbf{U}\mathbf{S}\mathbf{V})^{-1} \\ &= \mathbf{Q}_1^{-1} + c\mathbf{Q}_1^{-1}\mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{V}\mathbf{Q}_1^{-1} \\ \vec{r_i} &= (1-c)(\mathbf{Q}_1^{-1}\vec{e_i} + c\mathbf{Q}_1^{-1}\mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{V}\mathbf{Q}_1^{-1}\vec{e_i}) \end{aligned}$$

which completes the proof of Lemma 1. It can be seen that the only approximation of B_LIN comes from the low-rank approximation for $\tilde{\mathbf{W}}_2$.

We can also interpret B_LIN from the perspective of latent semantic/concept space. By low-rank approximation on $\tilde{\mathbf{W}}_2$, we actually introduce a $t \times t$ latent concept space by $\mathbf{S}$. Furthermore, if we treat the original $\tilde{\mathbf{W}}$ as an $n \times n$ node space, $\mathbf{U}$ and $\mathbf{V}$ actually define the relationship between these two spaces ($\mathbf{U}$ for node-concept relationship and $\mathbf{V}$ for concept-node relationship). Thus, it can be seen that, instead of doing random walk with restart on the original whole node space, B_LIN decomposes it into the following simple steps:

(1) Doing RWR within the partition that contains the starting point (multiply $\vec{e_i}$ by $\mathbf{Q}_1^{-1}$);

(2) Jumping from node-space to latent concept space (multiply the result of (1) by $\mathbf{V}$);

(3) Doing RWR within the latent concept space (multiply the result of (2) by $\tilde{\mathbf{\Lambda}}$);

(4) Jumping back to the node space(multiply the result of (3) by $\mathbf{U}$);

(5) Doing RWR within each partition until convergence (multiply the result of (4) by $\mathbf{Q}_1^{-1}$).

3.1.2 NB_LIN

Lemma 2 *If* $\tilde{\mathbf{W}} = \mathbf{U}\mathbf{S}\mathbf{V}$ *holds, NB_LIN outputs exactly the same result as* PreCompute.

Proof: Taking $\tilde{\mathbf{W}}_1 = \mathbf{0}$ and $\mathbf{Q}_1 = \mathbf{I}$, by applying Lemma 1, we directly complete the proof of Lemma 2.

3.1.3 BB_LIN

Lemma 3 *BB_LIN outputs exactly the same result as* Pre-Compute.

Proof: Substituting equation (5) into equation (2), we have

$$\begin{aligned}\vec{r}_{i,1} &= (1-c)(\mathbf{I}-c^2\mathbf{M}\mathbf{M}^T)^{-1}(c\mathbf{M}\vec{e}_{i,2}+\vec{e}_{i,1})\\ \vec{r}_{i,2} &= (1-c)(\mathbf{I}-c^2\mathbf{M}^T\mathbf{M})^{-1}(c\mathbf{M}^T\vec{e}_{i,1}+\vec{e}_{i,2})\end{aligned}$$

Solving $\vec{r}_{i,2}$ directly completes the proof of 'q2' in table (4).

Define a new RWR, which takes 1) $(c\mathbf{M}\vec{e}_{i,2}+\vec{e}_{i,1})$ as the new starting vector; 2) $(c\mathbf{M}\mathbf{M}^T)$ as the new normalized weighted matrix; and 3) $(\mathbf{M}(c\mathbf{I})\mathbf{M}^T)$ as the low-rank approximation. Applying Lemma 2 to this RWR, we complete the proof for 'q1' in table (4), which in turn completes the proof of Lemma 3.

3.2 Computational and storage cost

In this section, we make a brief analysis for the proposed algorithms in terms of computational and storage cost. For the limited space, we only provide the result for B_LIN.

3.2.1 On-line computational cost

It is not hard to see that, at the on-line query stage of B_LIN (table 2, step q1), we only need a few matrix-vector multiplication operations as shown in equation (8). Therefore, B_LIN is capable of meeting the (near) real-time response requirement.

$$\begin{aligned}\vec{r}_0 &\leftarrow \mathbf{Q}_1^{-1}\vec{e}_i\\ \vec{r}_i &\leftarrow \mathbf{V}\vec{r}_0\\ \vec{r}_i &\leftarrow \tilde{\mathbf{\Lambda}}\vec{r}_i\\ \vec{r}_i &\leftarrow \mathbf{U}\vec{r}_i\\ \vec{r}_i &\leftarrow \mathbf{Q}_1^{-1}\vec{r}_i\\ \vec{r}_i &\leftarrow (1-c)(\vec{r}_0+c\vec{r}_i)\end{aligned} \quad (8)$$

3.2.2 Pre-computational cost

The main off-line computational cost of the proposed algorithm consists of the following parts:

(1) partitioning the whole graph;

(2) inversion of each $\mathbf{I}-c\tilde{\mathbf{W}}_{1,i}, (i=1,...,k)$;

(3) low-rank approximation on $\tilde{\mathbf{W}}_2$;

(4) inversion of $(\mathbf{S}^{-1}-\mathbf{V}\mathbf{Q}_1^{-1}\mathbf{U})$.

Thus, instead of solving the inversion of the original $n\times n$ matrix, B_LIN1) inverses $k+1$ small matrices ($\mathbf{Q}_{1,i}^{-1}$, i=1,...,k, and $\tilde{\mathbf{\Lambda}}$); 2) computes a low-rank approximation of a sparse $n\times n$ matrix ($\tilde{\mathbf{W}}_2$), and 3) partitions the whole graph.

3.2.3 Pre-storage cost

In terms of storage cost, we have to store $k+1$ small matrices ($\mathbf{Q}_{1,i}^{-1}, (i=1,...,k)$, and $\tilde{\mathbf{\Lambda}}$), one $n\times t$ matrix ($\mathbf{U}$) and one $t\times n$ matrix ($\mathbf{V}$). Moreover, we can further save the storage cost as shown in the following:

- An observation from all our experiments is that many elements in $\mathbf{Q}_{1,i}^{-1}$, $\mathbf{U}$ and $\mathbf{V}$ are near zeros. Thus, an optional step is to set these elements to be zero (by the threshold ξ_2) and to store these matrices as sparse format. For all experiments in this paper, we find that this step will significantly reduce the storage cost while almost not affecting the approximation accuracy.
- The normalized graph Laplacian is symmetric, which leads to 1) a symmetric $\mathbf{Q}_{1,i}^{-1}$, and 2) $\mathbf{U}=\mathbf{V}^T$, if eigen-value decomposition is used when computing the low-rank approximation[5]. By taking advantage of this symmetry property, we can further save 50% storage cost.

3.3 Error Bound

Developing an error bound for the general case of the proposed methods is difficult. However, for NB_LIN (table 3), we have the following lemma:

Lemma 4 *Let $\vec{r}$ and $\hat{\vec{r}}$ be the ranking vectors* [6] *by* PreCompute *and by NB_LIN, respectively. If NB_LIN takes eigen-value decomposition as low-rank approximation, $\|\vec{r}-\hat{\vec{r}}\|_2 \le (1-c)\sum_{i=t+1}^{n}\frac{1}{(1-c\lambda_i)}$, where λ_i is the i^{th} largest eigen-value of $\tilde{\mathbf{W}}$.*

Proof: Taking the full eigen-value decomposition for $\tilde{\mathbf{W}}$:

$$\tilde{\mathbf{W}} = \sum_{i=1}^{n}\lambda_i\cdot u_i\cdot u_i^T = \mathbf{U}\mathbf{S}\mathbf{U}^T \quad (9)$$

where λ_i and u_i are the i^{th} largest eigen-value and the corresponding eigen-vector of $\tilde{\mathbf{W}}$, respectively. $\mathbf{U}=[u_1,...u_n]$, and $\mathbf{S}=diag(\lambda_1,...,\lambda_n)$

Note $u_i\cdot u_i^T=\mathbf{I}$. We have:

[5] On the other hand, if we use partition-based low-rank approximation as in table (5), $\mathbf{U}$ and $\mathbf{V}$ are usually sparse and thus can be efficiently stored

[6] Here, we ignore the low script i of $\vec{r}$ and $\hat{\vec{r}}$ for simplicity

$$\tilde{\mathbf{\Lambda}} = (\mathbf{S}^{-1} - c\mathbf{U}^T\mathbf{U})^{-1} = \sum_{i=1}^{n} \frac{\lambda_i}{(1-c\lambda_i)} \cdot u_i \cdot u_i^T \quad (10)$$

By Lemma 2, we have:

$$\vec{r} = (1-c)\sum_{i=1}^{n} \frac{1}{(1-c\lambda_i)} \cdot u_i \cdot u_i^T \cdot \vec{e_i}$$
$$\hat{\vec{r}} = (1-c)\sum_{i=1}^{t} \frac{1}{(1-c\lambda_i)} \cdot u_i \cdot u_i^T \cdot \vec{e_i} \quad (11)$$

Thus, we have

$$\begin{aligned} \|\vec{r} - \hat{\vec{r}}\|_2 &= \|(1-c)\sum_{i=t+1}^{n} \frac{1}{(1-c\lambda_i)} \cdot u_i \cdot u_i^T \cdot \vec{e_i}\|_2 \\ &\leq (1-c)\|\sum_{i=t+1}^{n} \frac{1}{(1-c\lambda_i)} \cdot u_i \cdot u_i^T\|_2 \cdot \|\vec{e_i}\|_2 \\ &= (1-c)\sum_{i=t+1}^{n} \frac{1}{(1-c\lambda_i)} \quad (12) \end{aligned}$$

which completes the proof of Lemma 4.

4 Experimental Results

4.1 Experimental Setup

4.1.1 Datasets

- CoIR

This dataset contains 5,000 images. The images are categorized into 50 groups, such as beach, bird, mountain, jewelry, sunset, etc. Each of the categories contains 100 images of essentially the same content, which serve as the ground truth. This is a widely used dataset for image retrieval. Two kinds of low-level features are used, including color moment and pyramid wavelet texture feature. We use exactly the same method as in [14] to construct the weighted graph matrix $\mathbf{W}$, which contains $5,000$ nodes and $\approx 774K$ edges

- CoMMG

This dataset is used in [22], which contains around 7,000 captioned images, each with about 4 captioned terms. There are in total 160 terms for captioning. In our experiments, 1,740 images are set aside for testing. The graph matrix $\mathbf{W}$ is constructed exactly as in [22], which contains $54,200$ nodes and $\approx 354K$ edges.

- AP

The author-paper information of DBLP dataset [4] is used to construct the weighted graph $\mathbf{W}$ as in equation (5): every author is denoted as a node in $\mathbf{W}$, and the edge weight is the number of co-authored papers between the corresponding two authors. On the whole, there are $\approx 315K$ nodes and $\approx 1,834K$ non-zero edges in $\mathbf{W}$.

- AC

The author-conference information of DBLP dataset [4] is used to construct the bipartite graph $\mathbf{M}$: each row corresponds to an author and each column corresponds to a conference; and the edge weight $\mathbf{M}_{i,j}$ is the number of papers that the i^{th} author publishes in j^{th} conference. On the whole, there are $\approx 291K$ nodes ($\approx 288K$ authors and $\approx 3K$ conferences) and $\approx 661K$ non-zero edges in $\mathbf{M}$.

All the above datasets are summarized in table(6):

Table 6. Summary of data sets

dataset	number of nodes	number of edges
CoIR	$5K$	$\approx 774K$
CoMMG	$\approx 52K$	$\approx 354K$
AP	$\approx 315K$	$\approx 1,834K$
AC	$\approx 291K$	$\approx 661K$

4.1.2 Applications

As mentioned before, many applications can be built upon random walk with restart. In this paper, we test the following applications:

- Center-piece subgraph discovery (CePs) [26]
- Content based image retrieval (CBIR) [14]
- Cross-modal correlation discovery (CMCD), including automatic captioning of images [22]
- neighborhood formulation (NF) for both uni-partite graph and bipartite graph [25]

The typical datasets for these applications in the past years are summarized in table(4.1.2)

4.1.3 Parameter Setting

The proposed methods are compared with *OnTheFly*, *Pre-Compute* and *Blk*. All these methods share 3 parameters: c, m and ξ_1. we use the same parameters for CBIR as [14], that is $c = 0.95$, $m = 50$ and $\xi_1 = 0$. For the rest applications, we use the same setting as [22] for simplicity, that is $c = 0.9$, $m = 80$ and $\xi_1 = 10^{-8}$.

Table 7. Summary of typical applications with different datasets

	CBIR	CMCD	Ceps	NF
CoIR	√			√
CoMMG		√		
AP			√	
AC				√

For B_LIN and NB_LIN, we take $\xi_2 = 10^{-4}$ to sparsify $\mathbf{Q}_1$, $\mathbf{U}$, and $\mathbf{V}$ which further reduces storage cost. We evaluate different choices for the remaining parameters. For clarification, in the following experiments, B_LIN is further referred as B_LIN(k, t, Eig/Part), where k is the number of partition, t is the target rank of the low-rank approximation, and "Eig/Part" denotes the specific method for doing low-rank approximation – "Eig" for eigen-value decomposition and "Part" for partition-based low-rank approximation. Similarly, NB_LIN is further referred as NB_LIN(t, Eig/Part), and *Blk* is further referred as *Blk*(k).

For the datasets with groundtruth (CoIR and CoMMG), we use the relative accuracy $RelAcu$ as the evaluation criterion:

$$RelAcu = \frac{\widehat{Acu}}{Acu} \tag{13}$$

where $\widehat{Acu}$ and Acu are the accuracy values by the evaluated method and by *PreCompute*, respectively.

Another evaluation criterion is $RelScore$,

$$RelScore = \frac{\widehat{tScr}}{tScr}, \tag{14}$$

where $\widehat{tScr}$ and $tScr$ are the total relevance scores captured by the evaluated method and by *PreCompute*, respectively.

All the experiments are performed on the same machine with 3.2GHz CPU and 2GB memory.

4.2 CoIR Results

100 images are randomly selected from the original dataset as the query images and the precision vs. scope is reported. The user feedback process is simulated as follows. In each round of relevance feedback (RF), 5 images that are most relevant to the query based on the current retrieval result are fed back and examined. It should be pointed out that the initial retrieval result is equivalent to that for neighborhood formulation (NF). $RelAcu$ is evaluated on the first 20 retrieved images, that is, the precision within the first 20 retrieved images. In figure (3), the results are evaluated from three perspectives: accuracy vs. query time (QT), accuracy vs. pre-computational time (PT) and accuracy vs. pre-storage cost (PS). In the figure, the QT, PT and PS costs are in log-scale. Note that pre-computational time and storage cost are the same for both initial retrieval and relevance feedback, therefore, we only report accuracy vs. pre-computational time and accuracy vs. pre-storage cost for initial retrieval.

It can be seen that in all the figures, B_LIN and NB_LIN always lie in the upper-left zone, which indicates that the proposed methods achieve a good balance between on-line response quality and off-line processing cost. Both B_LIN and NB_LIN 1) achieve about one order of magnitude speedup (compared with *OnTheFly*); and 2) save one order of magnitude on pre-computational and storage cost. For example, B_LIN(50, 300, Eig) preserves 95%+ accuracy for both initial retrieval and relevance feedback, while it 1) achieves 32x speedup for on-line response (0.09Sec/2.91Sec), compared with *OnTheFly*; and 2)save 8x on storage (21M/180M) and 161x on pre-computational cost (90Sec/14,500Sec), compared with *PreCompute*. NB_LIN(600,Eig) preserves 93%+ accuracy for both initial retrieval and relevance feedback, while it 1) achieves 97x speedup for on-line response (0.03Sec/2.91Sec), compared with *OnTheFly*; and 2)saves 10x on storage(17M/180M) and 48x on pre-computational cost (303Sec/14,500Sec), compared with *PreCompute*.[7].

For the task of neighborhood formation (NF), figure (4) shows the result of RelScore vs. scope. It can been seen that by exploring both the block-wise and linear correlations structure simultaneously, 1) both *Blk*(50) and NB_LIN(50, Eig) capture most neighborhood information (for example, they both capture about 90% score for the precision on the first 10 retrieved images), and 2) B_LIN(50, 300, Eig) captures 95%+ score over the whole scope. (The improvement becomes even more significant with the increase of the scope).

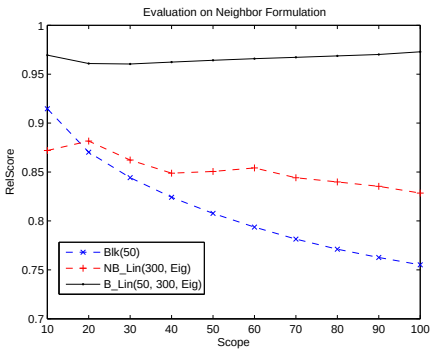
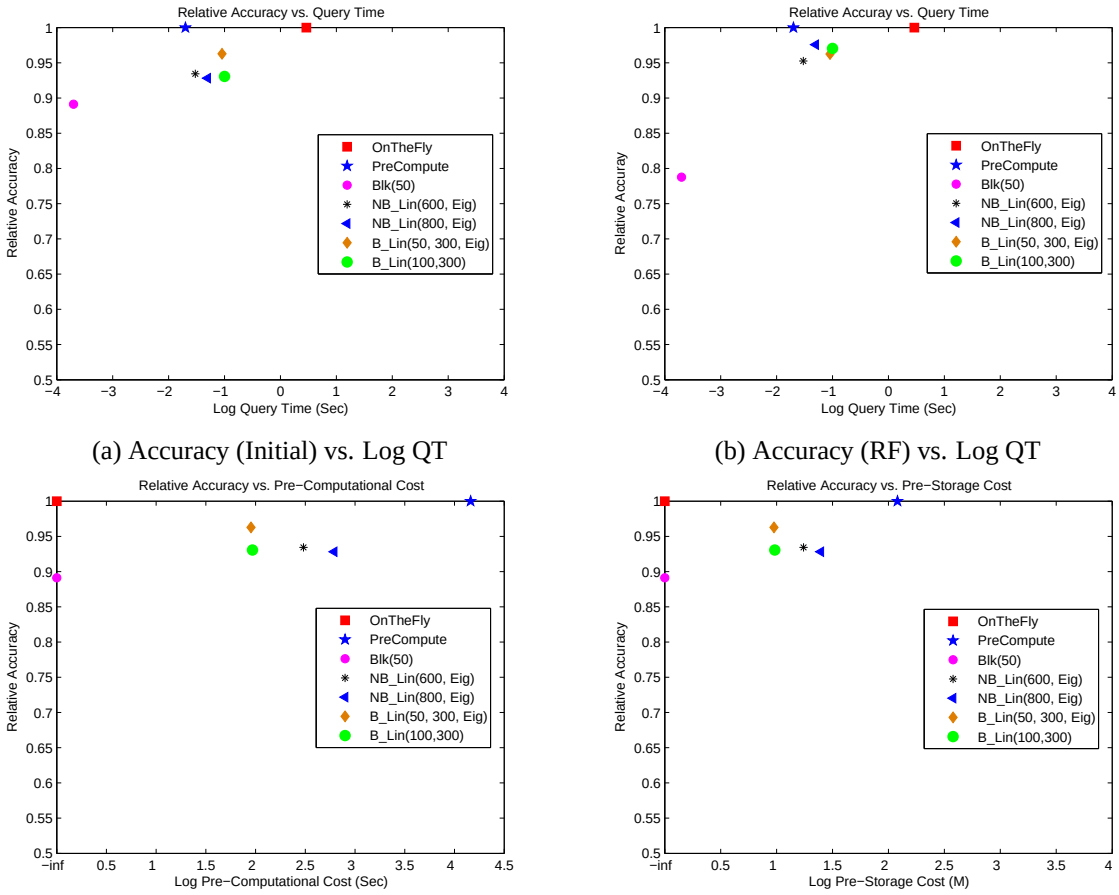


Figure 4. Evaluation on CoIR for NF

[7] We also perform experiment on BlockRank [18]. However, the result is similar with *OnTheFly*. Thus, we do not present it in this paper.



(a) Accuracy (Initial) vs. Log QT

(b) Accuracy (RF) vs. Log QT

(c)Accuracy (Initial) vs. Log PT

(d) Accuracy (Initial) vs. Log PS

Figure 3. Evaluation on CoIR for CBIR

4.3 CoMMG Results

For this dataset, we only compare NB_LIN with *OnTheFly* and *PreCompute*. The results are shown in figure (6). The x-axis of figure (6) is plotted in log-scale. Again, NB_LIN lies in the upper-left zone in all the figures, which means that NB_LIN achieves a good balance between on-line quality and off-line processing cost. For example, NB_LIN(100, Eig) preserves 91.3% quality, while it 1) achieves 154x speedup for on-line response (0.029/4.50Sec), compared with *OnTheFly*; 2) saves 868x on storage (281/243,900M) and 479x on pre-computational cost (46/21,951Sec), compared with *PreCompute*. The relative precision/recall vs. scope is shown in figure (5).

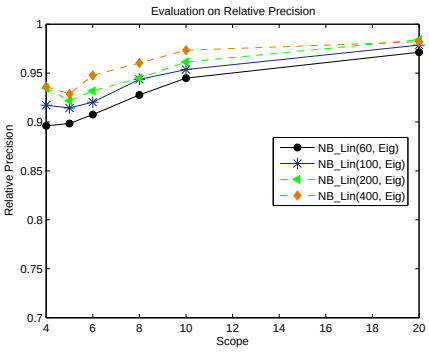
4.4 AP Results

This dataset is used to evaluate Ceps as in [26]. B_LIN is used to generate 1000 candidates, which are further fed to the original Ceps Algorithm [26] to generate the final center-piece subgraphs. We fix the number of query nodes to be 3 and the size of the subgraph to be 20. RelScore is measured by "Important Node Score" as in [26]. The result is shown in figure (7).

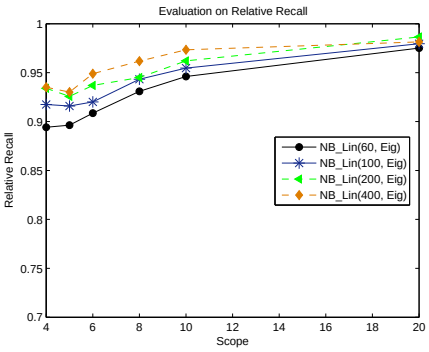
Again, B_LIN lies in the upper-left zone in all the figures, which means that B_LIN achieves a good balance between on-line quality and off-line processing cost. For example, B_LIN(100, 4000, Part) preserves 98.9% quality, while it 1) achieves 27x speedup for on-line response (9.45/258.2Sec), compared with *OnTheFly*; 2) saves 2264x on storage (269/609,020M) and 214x on pre-computational cost (8.7/1875Hour), compared with *PreCompute*.

4.5 AC Results

For this dataset, the number of conferences ($3K$) is much less than that of the authors ($228K$). We evaluate BB_LIN for the following four tasks:



(a) Relative precision



(b) Relative recall

Figure 5. Precision/recall for CMCD

- **C_C:** Given a conference, find its most related conferences
- **C_A:** Given a conference, find its most related authors
- **A_A:** Given an author, find its most related authors
- **A_C:** Given an author, find its most related conferences

On this application, BB_LIN preserves 100% accuracy for all the tasks. Thus, in table (8), we only report Query time (QT), Pre-computational time (PT), and Pre-storage cost (PS). Note that the query time for BB_LIN might differ for the different tasks. For clarification, BB_LIN is further referred as BB_LIN(C/A C/A). (For example, BB_LIN(C, A) denotes using BB_LIN for C_A task.)

As shown in table (8), BB_LIN can achieve up to 3 orders of magnitude speedup, with light off-line computational and storage cost (20.5Sec for pre-computation and 56M for pre-storage). For example, it achieves 180x speedup for **A_A** (0.13/23.98Sec) and 1,800 speedup for **C_C**(0.013/23.98Sec).

Table 8. Evaluation on AC for NF

Method	QT(Sec)	PT(Sec)	PS(M)
OnTheFly	23.97	0	6.7
PreCompute	0.001	6,990,648	626,250
BB_LIN(C, A)	0.097	20.50	56
BB_LIN(C, C)	0.013	20.50	56
BB_LIN(A, C)	0.035	20.50	56
BB_LIN(A, A)	0.13	20.50	56

5 Related work

In this Section, we briefly review related work, which can be categorized into three groups: 1) random walk related methods; 2) graph partitioning methods and 3) the methods for low-rank approximation.

Random walk related methods. There are several methods similar to RWR, including electricity-based method [28], graph-based Semi-supervised learning [27] [7] and so on. Exact solution of these methods usually requires the inversion of a matrix which is often diagonal dominant and of big size. Other methods sharing this requirement include regularized regression, Gaussian process regression [24], and so on. Existing fast solution for RWR include Hub-vector decomposition based [16]; block structure based [18] [25]; fingerprint based [9], and so on. Many applications take random walk and related methods as the building block, including PageRank [21], personalized PageRank [13], SimRank [15], neighborhood formulation in bipartite graphs [25], content-based image retrieval [14], cross modal correlation discovery [22], the BANKS system [2], ObjectRank [3], RalationalRank [10], and so on.

Graph partition and clustering. Several algorithms have been proposed for graph partition and clustering, e.g. METIS [19], spectral clustering [20], flow simulation [8], co-clustering [6], and the betweenness based method [11]. It should be pointed out that the proposed method is orthogonal to the partition method.

Low-rank approximation: One of the widely used techniques is singular vector decomposition (SVD) [12], which is the base for a lot of powerful tools, such as latent semantic index (LSI) [5], principle component analysis (PCA) [17], and so on. For symmetric matrices, a complementary technique is the eigen-value decomposition [12]. More recently, CUR decomposition has been proposed for sparse matrices [1].

6 Conclusions

In this paper, we propose a fast solution for computing the random walk with restart. The main contributions of the paper are as follows:

- The design of B_LIN and its derivative, NB_LIN. These methods take advantages of the block-wise structure and linear correlations in the adjacency matrix of real graphes, using the Sherman-Morrison Lemma.
- The proof of an error bound for NB_LIN. To our knowledge, this is the first attempt to derive an error bound for fast random walk with restart.
- Extensive experiments are performed on several real datasets, on typical applications. The results demonstrate that our proposed algorithm can nicely balance the off-line processing cost and the on-line response quality. In most cases, our methods preserve 90%+ quality, with dramatic savings on the pre-computation cost and the query time.
- A fast solution (BB_LIN) for one particular class of bipartite graphs. Our method achieves up to 1,800x speedup with light pre-computational and storage cost, without suffering quality loss.

Future work includes exploring error bounds for the general case of B_LIN, and performing comparative experiments with other candidate solutions, such as [16] and [9].

A Appendix

Sherman-Morrison Lemma [23]: if $\mathbf{X}^{-1}$ exists, then:

$$(\mathbf{X} - \mathbf{USV})^{-1} = \mathbf{X}^{-1} + \mathbf{X}^{-1}\mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{V}\mathbf{X}^{-1}$$

where $\tilde{\mathbf{\Lambda}} = (\mathbf{S}^{-1} - \mathbf{V}\mathbf{X}^{-1}\mathbf{U})^{-1}$

References

[1] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximation. In *STOC*, 2001.

[2] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, and S. S. Parag. Banks: Browsing and keyword searching in relational databases. In *VLDB*, pages 1083–1086, 2002.

[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[4] http://www.informatik.uni-trier.de/ ley/db/.

[5] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[6] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 03)*, Washington, DC, August 24-27 2003.

[7] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*, pages 118–127, 2004.

[8] G. Flake, S. Lawrence, and C. Giles. Efficient identification of web communities. In *KDD*, pages 150–160, 2000.

[9] D. Fogaras and B. Racz. Towards scaling fully personalized pagerank. In *Proc. WAW*, pages 105–117, 2004.

[10] F. Geerts, H. Mannila, and E. Terzi. Relational link-based ranking. In *VLDB*, pages 552–563, 2004.

[11] M. Girvan and M. E. J. Newman. Community structure is social and biological networks.

[12] G. Golub and C. Loan. *Matrix Computation*. Johns Hopkins, 1996.

[13] T. H. Haveliwala. Topic-sensitive pagerank. *WWW*, pages 517–526, 2002.

[14] J. He, M. Li, H. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *ACM Multimedia*, pages 9–16, 2004.

[15] G. Jeh and J. Widom. Simrank: A measure of structural-context similarity. In *KDD*, pages 538–543, 2002.

[16] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, 2003.

[17] I. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[18] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing pagerank. In *Stanford University Technical Report*, 2003.

[19] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning for irregular graphs. *SIAM Review*, 41(2):278–300, 1999.

[20] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856, 2001.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998. Paper SIDL-WP-1999-0120 (version of 11/11/1999).

[22] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.

[23] W. Piegorsch and G. E. Casella. Inverting a sum of matrices. In *SIAM Review*, 1990.

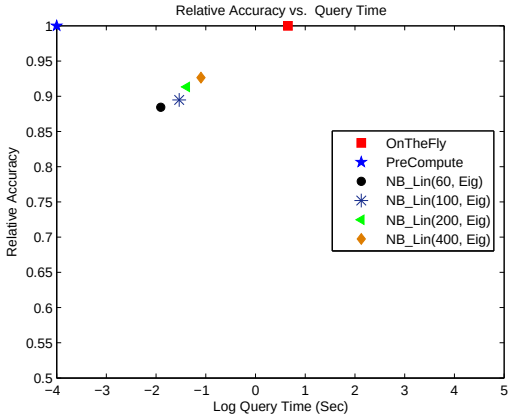
[24] C. E. Rasmusen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[25] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, pages 418–425, 2005.

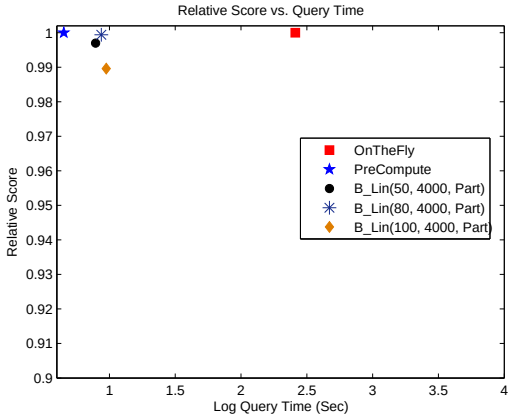
[26] H. Tong and C. Faloutsos. Center-piece subgraphs: Problem definition and fast solutions. In *KDD*, 2006.

[27] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Scholkopf. Learning with local and global consistency. In *NIPS*, 2003.

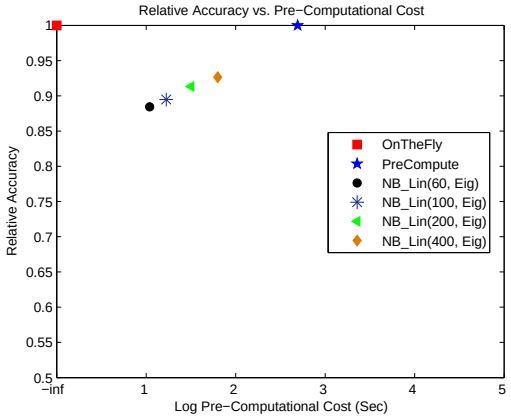
[28] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian field and harmonic functions. In *ICML*, pages 912–919, 2003.



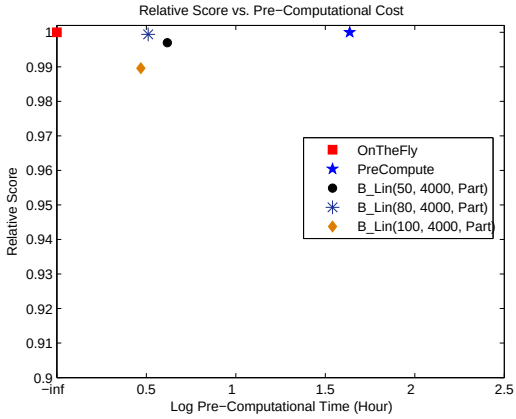
(a) Accuracy vs. Log QT



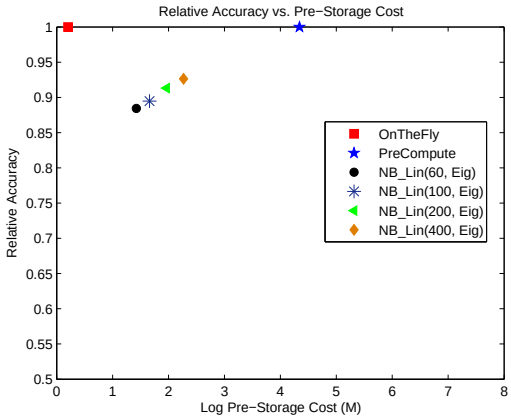
(a) Accuracy vs. Log QT



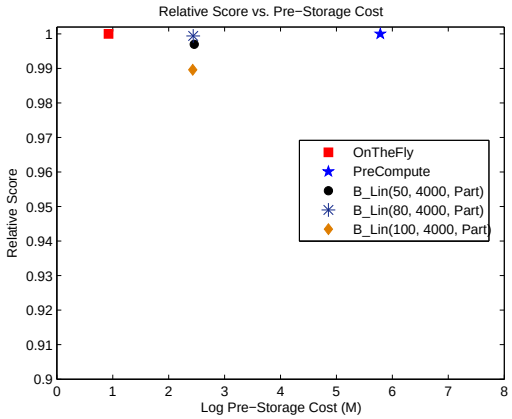
(b) Accuracy vs. Log PT



(b) Accuracy vs. Log PT



(c) Accuracy vs. Log PS



(c) Accuracy vs. Log QS

Figure 6. Evaluation on CoMMG for CMCD

Figure 7. Evaluation on AP for Ceps