Fast Random Walk Graph Kernel

U Kang
Carnegie Mellon University

Hanghang Tong
IBM T.J. Watson

Jimeng Sun
IBM T.J. Watson

Abstract

Random walk graph kernel has been used as an important tool for various data mining tasks including classification and similarity computation. Despite its usefulness, however, it suffers from the expensive computational cost which is at least $O(n^3)$ or $O(m^2)$ for graphs with n nodes and m edges.

In this paper, we propose ARK, a set of fast algorithms for random walk graph kernel computation. ARK is based on the observation that real graphs have much lower intrinsic ranks, compared with the orders of the graphs. ARK exploits the low rank structure to quickly compute random walk graph kernels in $O(n^2)$ or $O(m)$ time.

Experimental results show that our method is up to 97,865× faster than the existing algorithms, while providing more than 91.3% of the accuracies.

1 Introduction

Many real-world, complex objects with structural properties can be naturally modeled as graphs. For instance, web can be naturally represented as a graph with webpages as nodes and hyperlinks as edges. In the medical domain, the symptom-lab test graph of a given patient, which can be constructed from his/her medical records, provides a good indicator of the structure information of possible disease s/he carries (e.g., the association between a particular symptom and some lab test, the co-occurrence of different symptom). How can we characterize the difference of the current web graph from the one from last year to spot potential abnormal activities? How can we measure the similarities among different patients so that we can segment them into different categories?

Graph kernel [39] provides a natural tool to answer the above questions. Among others, one of the most powerful graph kernel is based on random walks, and has been successfully applied to many real applications (see Section 6 for a review). Despite its success, one main challenge remains open in terms of the scalability. To date, the best known algorithm to compute random walk based graph kernel is *cubic* in terms of the number of the nodes in the graph. Consequently, most, if not all, of the current random walk graph kernel algorithms quickly become computationally infeasible for the graphs with more than *hundreds of nodes*.

To address this issue, we propose a family of fast algorithms to compute random walk graph kernel. The key observation is that many real graphs have much lower intrinsic ranks, compared with the orders of the graphs. The heart of our proposed algorithms is to leverage such low-rank structure as an intermediate step to speed up the computation. Our experimental evaluations on many real graphs show that the new proposed methods (1) are much faster than the existing ones (up to 97,865× speed-up); (2) scale to the graphs with 325,729 nodes that all the existing methods fail to compute; and (3) bear a high approximation accuracy (more than 91.3%).

The main contributions of this paper are as follows.

1. **Algorithms.** We propose ARK, a set of fast algorithms to compute random walk graph kernel, which significantly reduce the time complexity (see Table 2 for a comparison).
2. **Proofs and Analysis.** We show the approximation error bounds as well as the efficiency of our methods.
3. **Experiments.** We perform extensive experiments on many real world graphs, showing the speed-up (maximum 97,865×) and the accuracy (more than 91.3%) of our proposed method.

The rest of paper is organized as follows. Section 2 presents the preliminaries of the standard random walk graph kernel. Sections 3 and 4 describe our proposed ARK algorithms for unlabeled and labeled graphs, respectively. Section 5 presents the experimental results. After reviewing related works in Section 6, we conclude the paper in Section 7.

2 Preliminaries; Random Walk Graph Kernel

In this section, we describe the preliminaries on the random walk graph kernel whose fast algorithms will be proposed in Sections 3 and 4. Table 1 lists the symbols used in this paper.

Symbol	Definition
G	a graph
n	number of nodes in a graph
m	number of edges in a graph
A	adjacency matrix of a graph
$k(G_1, G_2)$	exact graph kernel function on graphs G_1 and G_2
$\hat{k}(G_1, G_2)$	approximate graph kernel function on graphs G_1 and G_2
W	weight matrix in random walk kernel
c	decay factor in random walk kernel
d_n	number of distinct node labels
r	reduced rank after low rank approximation

Table 1: Table of symbols.

2.1 Definition Random walk graph kernel has been used for classification and measuring similarities of graphs [19, 39]. Given two graphs, the random walk graph kernel computes the number of common walks in two graphs. Two walks are common if the lengths of the walks are equal, and the label sequences are the same (for nodes/edges labeled graphs). The computed number of common walks is used to measure the similarity of two graphs.

We derive the random walk graph kernel for the unlabeled and unnormalized case, and generalize the definition to labeled and normalized cases. Given two graphs $G_1 = \{V_1, E_1\}$ and $G_2 = \{V_2, E_2\}$, the direct product graph $G_\times = \{V_\times, E_\times\}$ of G_1 and G_2 is a graph with the node set $V_\times = \{(v_1, v_2)|v_1 \in V_1, v_2 \in V_2\}$, and the edge set $E_\times = \{((v_{11}, v_{21}), (v_{12}, v_{22}))|(v_{11}, v_{12}) \in E_1, (v_{21}, v_{22}) \in E_2\}$. A random walk on the direct product graph $G_\times$ is equivalent to the simultaneous random walks on G_1 and G_2. Let p_1 and p_2 be the starting probabilities of the random walks on G_1 and G_2, respectively. The stopping probabilities q_1 and q_2 are defined similarly. Then, the number of length l common walks on the direct product graph $G_\times$ is given by $(q_1 \otimes q_2)(W_1^T \otimes W_2^T)^l(p_1 \otimes p_2)$, where W_1 and W_2 are the adjacency matrices of G_1 and G_2, respectively [39]. Discounting the longer walks by the decay factor c, and summing up all the common walks for all different lengths, we derive the equation for the random walk graph kernel:

$$\begin{aligned} k(G_1, G_2) = & \sum_{l=0}^{\infty}(q_1 \otimes q_2)(W_1^T \otimes W_2^T)^l(p_1 \otimes p_2) \\ = & (q_1 \otimes q_2)(I - c(W_1^T \otimes W_2^T))^{-1}(p_1 \otimes p_2). \end{aligned}$$

More generally, the random walk graph kernel is defined as follows.

Definition 1. (Random Walk Graph Kernel) *Let G_1 and G_2 be two graphs. Let p_1 and p_2 be the starting probabilities of the random walks on G_1 and G_2, respectively. The stopping probabilities q_1 and q_2 are defined similarly. The random walk graph kernel $k(G_1, G_2)$ is determined by*

$$k(G_1, G_2) := q^T(I - cW)^{-1}p, \tag{2.1}$$

where W is a weight matrix, c is a decay factor, $p = p_1 \otimes p_2$, and $q = q_1 \otimes q_2$. □

The weight matrix W is determined by two factors: normalization and labels on nodes/edges.

Normalization. Let A_1 and A_2 be the adjacency matrices of G_1 and G_2, respectively. For unnormalized case, the weight matrix is given by $W = A_1^T \otimes A_2^T$. For normalized case, the weight matrix is given by $W = A_1^T D_1^{-1} \otimes A_2^T D_2^{-1}$, where D_1 and D_2 are diagonal matrices whose ith diagonal elements are given by $\sum_j A_1(i, j)$ and $\sum_j A_2(i, j)$, respectively.

Labels. Nodes and edges can be labeled. First, we consider node labeled graphs. Let G_1 have n_1 nodes and G_2 have n_2 nodes. Let l_1 and l_2 be the node label vectors of G_1 and G_2, respectively. The $((i-1)\cdot n_2 + j)$th row of the weight matrix W are zeroed out if the ith element $l_1(i)$ of l_1 and the jth element $l_2(j)$ of l_2 do not have the same labels. Second, we consider edge labeled graphs. Let W_1 and W_2 be the normalized or unnormalized adjacency matrices of G_1 and G_2, respectively. The $((i_1 - 1)\cdot n_2 + i_2, (j_1 - 1)\cdot n_2 + j_2)$th element of W is 1 if and only if the edge labels of $W_1^T(i_1, j_1)$ and $W_2^T(i_2, j_2)$ are the same.

2.2 Computing Random Walk Graph Kernel We describe methods for computing the random walk graph kernel. For simplicity, we assume that both the graphs G_1 and G_2 have n nodes and m edges.

Naive Method. The naive algorithm is to compute the Equation (2.1) by inverting the $n^2 \times n^2$ matrix W. Since inverting a matrix takes time proportional to the cube of the number of rows/columns, the running time is $O(n^6)$.

Sylvester Method. If the weight matrix can be decomposed into one or two sums of Kronecker products, Sylvester method solves the Equation (2.1) in $O(n^3)$ time [39]. However, there are two drawbacks in Sylvester method. First, the method requires the two graphs to have the same number of nodes, which is often not true. Second, the theoretical running time of Sylvester method on the weight matrix composed of more than two Kronecker products is unknown.

Spectral Decomposition Method. For unlabeled and unnormalized matrices, spectral decomposition method runs in $O(n^3)$ time. The problem of spectral decomposition method is that it can't run on the labeled graph or normalized matrix.

Conjugate Gradient Method. Conjugate gradient (CG) method is used to solve linear systems efficiently. To use CG for computing random walk graph kernel, we first solve $(I - cW)x = p$ for x using CG, and compute $q^T x$. Each iteration of CG takes $O(m^2)$ since the most expensive operation is the matrix-vector multiplication. Thus CG takes $O(m^2 i_F)$ time where i_F denote the number of iterations. A problem of the CG method is its high memory requirement: it requires $O(m^2)$ memory.

Fixed Point Iteration Method. Fixed point iteration method first solves $(I - cW)x = p$ for x by iterative matrix-vector multiplications. Note that the fixed point iteration method converges only when the decay factor c is smaller than $|\xi_1|^{-1}$ where ξ_1 is the largest magnitude eigenvalue of W. Similar to CG, the fixed point iteration method takes $O(m^2 i_F)$ time for i_F iterations, and has the same problems of requiring $O(m^2)$ memory.

Since the Sylvester method and the spectral decomposition method cannot be used for kernels on general graphs, we use the conjugate gradient and the fixed point iterations method for experimental evaluations in Section 5.

3 Proposed Approximation: Unlabeled Graphs

As we saw in Section 2, the exact algorithms to compute the random walk graph kernel take too much time or require too much memory. To solve the problem, we propose Ark (Approximate Random walk Kernel), a set of approximation algorithms to compute the random walk graph kernel in this and the next sections. Table 2 shows the summary of the running time comparison of our Ark and the exact algorithms. This section addresses unlabled graphs which corresponds to the cases (a) and (b) in Table 2. The node labeled graphs, which corresponds to the cases (c) and (d) in Table 2, are handled in the next section.

3.1 Asymmetric W (Ark-U) We first consider node unlabeled graphs with the normalized weight matrix, which corresponds to the case (a) in Table 2. Let two graphs G_1 and G_2 have the adjacency matrices A_1 and A_2, respectively. Let $W_1 = A_1 D_1^{-1}$ and $W_2 = A_2 D_2^{-1}$ be the row normalized adjacency matrix of G_1 and G_2, where D_1 and D_2 are diagonal matrices whose ith diagonal elements are given by $\sum_j A_1(i,j)$ and $\sum_j A_2(i,j)$, respectively. In this setting, the weight matrix W is given by

$$W = W_1^T \otimes W_2^T. \tag{3.2}$$

Since the W matrix is large, we approximate W using its low-rank approximation. More precisely, we define the r-approximation of a matrix as follows.

Definition 2. (r-Approximation of a Matrix) *Given a matrix A, the r-approximation of A is a matrix $\hat{A}$ satisfying the following equation:*

$$||A - \hat{A}||_F \leq min_{Z|rank(Z)=r}||A - Z||_F, \tag{3.3}$$

meaning $\hat{A}$ provides better approximation to A than the best rank-r approximation.

□

Our goal is an approximate random walk kernel which is defined as follows.

Definition 3. (Approx. Random Walk Kernel) *Given a random walk graph kernel function $k(G_1, G_2) := q^T(I - cW)^{-1}p$, the approximate random walk graph kernel $\hat{k}(G_1, G_2)$ is given by*

$$\hat{k}(G_1, G_2) := q^T(I - c\hat{W})^{-1}p, \tag{3.4}$$

where $\hat{W}$ is a low rank approximation of W.

We want the $\hat{W}$ matrix to be as close as possible to W, while preserving a low rank. Ideally, $\hat{W}$ can be an r-approximation of W. It is well known that the singular value decomposition (SVD) gives the best low rank approximation [37]. Thus, a simple but naive approach to get the r-approximation of W is to use rank-r SVD of W. However, such method is inefficient since the running time is $O(m^2 r)$, and the W matrix needs to be explicitly constructed. Our proposed idea is to use the SVD of W_1^T and W_2^T to compute the r-approximation of the weight matrix W. This approach has another advantage of not needing to explicitly construct the W matrix. We first give the algorithm, called Ark-U, and prove its correctness. Algorithm 1 shows our approximation algorithm.

We show that Algorithm 1 is correct.

Theorem 3.1. (Correctness of Ark-U) *The Algorithm 1 (Ark-U) gives the approximate random walk kernel*

$$\hat{k}(G_1, G_2) = q^T(I - c\hat{W})^{-1}p, \tag{3.5}$$

where $\hat{W}$ is the r-approximation of $W = W_1 \otimes W_2$.

Case	Normalization of W	# of Node Labels	Exact	Ark
(a)	Normalized	1	$O(n^3)$	$O(n^2r^4 + r^6 + mr)$
(b)	Unnormalized	1	$O(n^3)$	$O((m+n)r + r^2)$
(c)	Normalized	d_n	$O(m^2 i_F)$	$O(d_n n^2 r^4 + r^6 + mr)$
(d)	Unnormalized	d_n	$O(m^2 i_F)$	$O(d_n n^2 r^4 + r^6 + mr)$

Table 2: Summary of the running time comparison of our ARK and the exact algorithms. n is the number of nodes, m is the number of edges, r is the reduced rank after low rank approximation, d_n is the number of node labels, and i_F is the number of iterations in the conjugate gradient or the fixed point iteration methods. Note our proposed ARK is faster than the exact method, since $n >> r$. ARK-U handles case (a) in Section 3.1. ARK-U+ addresses case (b) in Section 3.2, and ARK-L deals with case (c) and (d) in Section 4.

Algorithm 1 ARK-U: approximate random walk kernel for unlabeled nodes and asymmetric W

Input: Adjacency matrix A_1 of a graph G_1,
adjacency matrix A_2 of a graph G_2,
starting and ending probabilities p_1 and q_1 for G_1,
starting and ending probabilities p_2 and q_2 for G_2,
decay factor c.
Output: Approx. random walk kernel $\hat{k}(G_1, G_2)$

1: $W_1 \leftarrow A_1 D_1^{-1}$; // row normalize A_1
2: $W_2 \leftarrow A_2 D_2^{-1}$; // row normalize A_2
3: $U_1 \Lambda_1 V_1^T \leftarrow W_1^T$; //SVD on W_1^T
4: $U_2 \Lambda_2 V_2^T \leftarrow W_2^T$; //SVD on W_2^T
5: $\tilde{\Lambda} \leftarrow ((\Lambda_1 \otimes \Lambda_2)^{-1} - c(V_1^T \otimes V_2^T)(U_1 \otimes U_2))^{-1}$;
6: $L \leftarrow (q_1^T U_1 \otimes q_2^T U_2)$;
7: $R \leftarrow (V_1^T p_1 \otimes V_2^T p_2)$;
8: $\hat{k}(G_1, G_2) \leftarrow (q_1^T p_1)(q_2^T p_2) + cL\tilde{\Lambda}R$;

Proof. Let $W_1^T = U_1 \Lambda_1 V_1^T$ and $W_2^T = U_2 \Lambda_2 V_2^T$ be the top r singular value decompositions of W_1^T and W_2^T, respectively. From the standard result of linear algebra,

$$\hat{W} = (U_1 \otimes U_2)(\Lambda_1 \otimes \Lambda_2)(V_1^T \otimes V_2^T)$$

is a singular value decomposition. The $\hat{W}$ satisfies $||W - \hat{W}||_F \leq min_{Z|rank(Z)=r}||W - Z||_F$ since the diagonal elements of the matrix $\Lambda_1 \otimes \Lambda_2$ contain the top r largest eigenvalues of $W_1^T \otimes W_2^T$.

Thus,

$$\begin{aligned}
&q^T(I - cW)^{-1}p \\
&= q^T(I - c(U_1 \otimes U_2)(\Lambda_1 \otimes \Lambda_2)(V_1^T \otimes V_2^T))^{-1}p \\
&= q^T(I + c(U_1 \otimes U_2)\tilde{\Lambda}(V_1^T \otimes V_2^T))p \\
&= q^T p + cq^T(U_1 \otimes U_2)\tilde{\Lambda}(V_1^T \otimes V_2^T)p \\
&= (q_1^T p_1)(q_2^T p_2) + c(q_1^T U_1 \otimes q_2^T U_2)\tilde{\Lambda}(V_1^T p_1 \otimes V_2^T p_2),
\end{aligned}$$

where the second equality comes from the Sherman-Morrison Lemma [28]. □

We show the time and the space complexities of Algorithm 1. Note that the time complexity $O(n^2r^4 + mr + r^6)$ of ARK-U is smaller than the best exact algorithm's complexity $O(n^3)$, since r is very small (6 is enough, as we see in Section 5) and considered as a constant.

THEOREM 3.2. (TIME COMPLEXITY OF ARK-U) ARK-U *takes* $O(n^2r^4 + mr + r^6)$ *time.*

Proof. The top r decompositions in lines 3 and 4 cost $O(mr)$. Computing $\tilde{\Lambda}$ in line 5 takes $O(n^2r^4 + r^6)$. Computing line 6, 7 and 8 takes $O(nr + r^4)$. □

THEOREM 3.3. (SPACE COMPLEXITY OF ARK-U) ARK-U *requires* $O(m + n^2r^2)$ *space.*

Proof. The storage of W_1 and W_2 require $O(m)$ space. The top r decompositions in lines 3 and 4 require $O(nr)$. Line 5 to 8 require $O(n^2r^2)$ space, thus making the total space complexity $O(m + n^2r^2)$. □

3.2 Symmetric W (Ark-U+) ARK-U can be used for both the symmetric and the asymmetric weight matrices. For symmetric weight matrix, however, we propose ARK-U+, an even faster approximation algorithm. ARK-U+ handles the case (b) in Table 2.

We first describe the weight matrix W in this setting. Assume two graphs G_1 and G_2 have the symmetric adjacency matrices A_1 and A_2, respectively. Then, the weight matrix W is given by

$$W = A_1^T \otimes A_2^T, \tag{3.6}$$

where W is also symmetric by the nature of Kronecker products [22]. Our proposed idea is to use the eigen decomposition, instead of SVD, to compute the r-approximation of W. Since the eigen decomposition and the SVD on symmetric matrices are different only

up to signs, the eigen decomposition gives the correct r-approximation. The computational advantage is that we need to store only one $n \times r$ eigenvectors, instead of two $n \times r$ singular vectors. Algorithm 2 shows our ARK-U+ algorithm for symmetric W.

Algorithm 2 ARK-U+: approximate random walk kernel for unlabeled nodes and symmetric W

Input: Adjacency matrix A_1 of a graph G_1,
adjacency matrix A_2 of a graph G_2
starting and ending probabilities p_1 and q_1 for G_1,
starting and ending probabilities p_2 and q_2 for G_2,
decay factor c.

Output: Approx. random walk kernel $\hat{k}(G_1, G_2)$

1: $U_1 \Lambda_1 U_1^T \leftarrow A_1^T$; //eigen decomposition on W_1
2: $U_2 \Lambda_2 U_2^T \leftarrow A_2^T$; //eigen decomposition on W_2
3: $\tilde{\Lambda} \leftarrow ((\Lambda_1 \otimes \Lambda_2)^{-1} - cI)^{-1}$;
4: $L \leftarrow (q_1^T U_1 \otimes q_2^T U_2)$;
5: $R \leftarrow (U_1^T p_1 \otimes U_2^T p_2)$;
6: $\hat{k}(G_1, G_2) \leftarrow (q_1^T p_1)(q_2^T p_2) + cL\tilde{\Lambda}R$;

We show that Algorithm 2 is correct.

THEOREM 3.4. (CORRECTNESS OF ARK-U+) *The Algorithm 2 (ARK-U+) gives the approximate random walk kernel*

$$\hat{k}(G_1, G_2) = q^T (I - c\hat{W})^{-1} p, \tag{3.7}$$

where $\hat{W}$ is the r-approximation of $W = W_1 \otimes W_2$.

Proof. Let $A_1^T = U_1 \Lambda_1 U_1^T$ and $A_2^T = U_2 \Lambda_2 U_2^T$ be the top r singular value decompositions of A_1 and A_2, respectively. From the standard result of linear algebra,

$$\hat{W} = (U_1 \otimes U_2)(\Lambda_1 \otimes \Lambda_2)(U_1^T \otimes U_2^T) \tag{3.8}$$

is a singular value decomposition. The $\hat{W}$ satisfies $||W - \hat{W}||_F \leq min_{Z|rank(Z)=r} ||W - Z||_F$ since the diagonal elements of the matrix $\Lambda_1 \otimes \Lambda_2$ contain the top r largest eigenvalues of $A_1^T \otimes A_2^T$.

Thus,

$$\begin{aligned}
&q^T (I - cW)^{-1} p \\
&= q^T (I - c(U_1 \otimes U_2)(\Lambda_1 \otimes \Lambda_2)(U_1^T \otimes U_2^T))^{-1} p \\
&= q^T (I + c(U_1 \otimes U_2)\tilde{\Lambda}(U_1^T \otimes U_2^T))p \\
&= q^T p + cq^T (U_1 \otimes U_2)\tilde{\Lambda}(U_1^T \otimes U_2^T)p \\
&= (q_1^T p_1)(q_2^T p_2) + c(q_1^T U_1 \otimes q_2^T U_2)\tilde{\Lambda}(U_1^T p_1 \otimes U_2^T p_2),
\end{aligned}$$

where the second equality comes from the Sherman-Morrison Lemma [28]. □

We show the time and the space complexities of Algorithm 2. Note that the time and the space complexities of ARK-U+ are smaller than those of ARK-U due to the exploitation of the symmetricity.

THEOREM 3.5. (TIME COMPLEXITY OF ARK-U+) ARK-U+ *takes $O((m+n)r + r^2)$ time.*

Proof. The top r decompositions in lines 1 and 2 cost $O(mr)$. Computing $\tilde{\Lambda}$ in line 3 takes $O(r^2)$ since $\tilde{\Lambda}$ is a diagonal matrix with $\frac{\lambda_1^i \lambda_2^j}{1 - c\lambda_1^i \lambda_2^j}$, for $1 \leq i, j \leq r$, as its elements. Computing line 4, 5 and 6 takes $O(nr + r^2)$. □

THEOREM 3.6. (SPACE COMPLEXITY OF ARK-U+) ARK-U *requires $O(m + nr + r^2)$ space.*

Proof. The storage of W_1 and W_2 require $O(m)$ space. The top r decompositions in lines 3 and 4 require $O(nr)$. Line 5 to 8 require $O(nr + r^2)$ space, thus making the total space complexity $O(m + nr + r^2)$. □

3.3 Error Bound How close is the approximate random walk kernel $\hat{k}(G_1, G_2)$ to the exact kernel $k(G_1, G_2)$? The analysis for general cases is difficult, but for ARK-U+ which handles symmetric W we have the following error bound.

THEOREM 3.7. *In ARK-U+, the difference of the exact and the approximate random walk kernel is bounded by*

$$|k(G_1, G_2) - \hat{k}(G_1, G_2)| \leq \sum_{(i,j) \notin F} |\frac{c\lambda_1^{(i)} \lambda_2^{(j)}}{1 - c\lambda_1^{(i)} \lambda_2^{(j)}}|, \tag{3.9}$$

where $\lambda_1^{(i)}$ and $\lambda_2^{(i)}$ are the ith largest eigenvalue of Λ_1 and Λ_2, respectively, and $F = \{(a, b) | a, b \in [1, k]\}$ is the set of pairs (a, b) where both a and b are in the range of $[1, k]$.

Proof. Let $W = A_1^T \otimes A_2^T$. Then, $(U_1 \otimes U_2)(\Lambda_1 \otimes \Lambda_2)(U_1^T \otimes U_2^T)$ is an eigen decomposition of W which includes top k largest eigenvalues of W. Let $u_1^{(i)}$ and $u_2^{(i)}$ be the ith column of U_1 and U_2, respectively. Then, $\tilde{u}^{(i,j)} := u_1^{(i)} \otimes u_2^{(j)}$ is the eigenvector of W with the corresponding eigenvalue $\lambda_1^{(i)} \lambda_2^{(j)}$. It follows that

$$\begin{aligned}
&(I - cW)^{-1} \\
&= I + c(U_1 \otimes U_2)\tilde{\Lambda}(U_1^T \otimes U_2^T) \\
&= I + \sum_{i,j \in [1,n]} \tilde{\lambda}^{(i,j)} \tilde{u}^{(i,j)} (\tilde{u}^{(i,j)})^T,
\end{aligned}$$

where $\tilde{\Lambda} := ((\Lambda_1 \otimes \Lambda_2)^{-1} - cI)^{-1}$, and $\tilde{\lambda}^{(i,j)} := \frac{c\lambda_1^{(i)} \lambda_2^{(j)}}{1 - c\lambda_1^{(i)} \lambda_2^{(j)}}$.

Now, we consider our approximation. Let $\hat{W}$ be the approximation of the W matrix from the top k low rank approximations of W_1 and W_2, as shown in Equation (3.8). Then,

$$(I - c\hat{W})^{-1} = I + \sum_{i,j \in [1,k]} \tilde{\lambda}^{(i,j)} \tilde{u}^{(i,j)} (\tilde{u}^{(i,j)})^T.$$

Thus,

$$\begin{aligned}
&|k(G_1, G_2) - \hat{k}(G_1, G_2)| \\
&= |q^T (I - cW)^{-1} p - q^T (I - c\hat{W})^{-1} p| \\
&= |q^T (\sum_{(i,j) \notin F} \frac{c\lambda_1^{(i)} \lambda_2^{(j)}}{1 - c\lambda_1^{(i)} \lambda_2^{(j)}} \tilde{u}^{(i,j)} (\tilde{u}^{(i,j)})^T) p| \\
&\leq ||q^T||_2 \cdot || \sum_{(i,j) \notin F} \frac{c\lambda_1^{(i)} \lambda_2^{(j)}}{1 - c\lambda_1^{(i)} \lambda_2^{(j)}} \tilde{u}^{(i,j)} (\tilde{u}^{(i,j)})^T ||_F \cdot ||p||_2 \\
&\leq \sum_{(i,j) \notin F} | \frac{c\lambda_1^{(i)} \lambda_2^{(j)}}{1 - c\lambda_1^{(i)} \lambda_2^{(j)}} |,
\end{aligned}$$

where in the last inequality we used the fact that $||q^T||_2 \leq ||q^T||_1 = 1$, $||p||_2 \leq ||p||_1 = 1$, and $|| \sum_i a_i u_i u_i^T ||_F = \sqrt{\mathtt{tr}(\sum_i a_i^2 u_i u_i^T)} = \sqrt{\sum_i a_i^2 \cdot \mathtt{tr}(u_i u_i^T)} = \sqrt{\sum_i a_i^2} \leq \sum_i |a_i|$ for any real numbers a_i and orthonormal vectors u_i. □

4 Proposed Approximation: Labeled Graphs

In this section, we describe Ark-L, an approximation algorithm to compute the random walk graph kernel on node labeled graphs. As discussed in the beginning of Section 3, Ark-L addresses the cases (c) and (d) in Table 2.

4.1 Weight Matrix As we saw in Section 2, the weight matrix W for node labeled graphs is constructed by zeroing out rows of the Kronecker products of normalized or unnormalized matrices. Specifically, given the normalized or unnormalized adjacency matrices W_1 and W_2 of G_1 and G_2, respectively, the weight matrix W is given by

$$W = \tilde{L}(W_1^T \otimes W_2^T), \tag{4.10}$$

where $\tilde{L}$ is a diagonal matrix whose (i,i)th element is 0 if the ith row of $(W_1^T \otimes W_2^T)$ is zeroed out due to label inconsistency, or 1 otherwise. Let $L_1^{(j)}$ be a diagonal label matrix whose ith element is 1 if the node i of the graph G_1 has the label j, and 0 otherwise. $L_2^{(j)}$ is defined similarly for the graph G_2. Then, $\tilde{L}$ is expressed by the sums of Kronecker products:

$$\tilde{L} = \sum_{j=1}^{d_n} L_1^{(j)} \otimes L_2^{(j)},$$

where d_n is the number of distinct node labels.

4.2 Approximation (Ark-L) We first show the approximation algorithm, Ark-L, for random walk kernel on node labeled graphs, and show its correctness. Algorithm 3 shows our approximation algorithm. We assume that W_1 and W_2 can be either row-normalized or unnormalized adjacency matrix of G_1 and G_2, respectively.

Algorithm 3 Ark-L: approximate random walk kernel for labeled nodes

Input: Weight matrix W_1 of a graph G_1,
weight matrix W_2 of a graph G_2,
label matrices $L_1^{(1)}$ to $L_1^{(d_n)}$ of G_1,
label matrices $L_2^{(1)}$ to $L_2^{(d_n)}$ of G_2,
starting and ending probability p_1 and q_1 for G_1,
starting and ending probability p_2 and q_2 for G_2,
decay factor c.

Output: Approx. random walk kernel $\hat{k}(G_1, G_2)$

1: $U_1 \Lambda_1 V_1^T \leftarrow W_1^T$; //SVD on W_1^T
2: $U_2 \Lambda_2 V_2^T \leftarrow W_2^T$; //SVD on W_2^T
3: $\tilde{\Lambda} \leftarrow ((\Lambda_1 \otimes \Lambda_2)^{-1} - c(\sum_{j=1}^{d_n} V_1^T L_1^{(j)} U_1 \otimes V_2^T L_2^{(j)} U_2))^{-1}$;
4: $L \leftarrow (\sum_{j=1}^{d_n} q_1^T L_1^{(j)} U_1 \otimes q_2^T L_2^{(j)} U_2)$;
5: $R \leftarrow (\sum_{j=1}^{d_n} V_1^T L_1^{(j)} p_1 \otimes V_2^T L_2^{(j)} p_2)$;
6: $\hat{k}(G_1, G_2) \leftarrow (\sum_{j=1}^{d_n} (q_1^T L_1^{(j)} p_1)(q_2^T L_2^{(j)} p_2)) + cL\tilde{\Lambda}R$;

We show that Algorithm 3 is correct.

Theorem 4.1. (Correctness of Ark-L) *The Algorithm 3 (*Ark-L*) gives the approximate random walk kernel*

$$\hat{k}(G_1, G_2) = q^T (I - c\hat{W})^{-1} p, \tag{4.11}$$

where $\hat{W} = \tilde{L} W_r$, and W_r is the r-approximation of $W_1 \otimes W_2$.

Proof. Let $W_1^T = U_1 \Lambda_1 V_1^T$ and $W_2^T = U_2 \Lambda_2 V_2^T$ be the top r singular value decompositions of W_1 and W_2, respectively. From the standard result of linear algebra,

$$\hat{W} = (U_1 \otimes U_2)(\Lambda_1 \otimes \Lambda_2)(V_1^T \otimes V_2^T)$$

is a singular value decomposition. The $\hat{W}$ satisfies $||W - \hat{W}||_F \leq min_{Z|rank(Z)=r} ||W - Z||_F$ since the

diagonal elements of the matrix $\Lambda_1 \otimes \Lambda_2$ contain top r largest eigenvalues of $A_1 \otimes A_2$.

Thus,

$$
\begin{aligned}
& q^T(I - c\tilde{L}(W_1^T \otimes W_2^T))^{-1}\tilde{L}p \\
& = q^T(I - c\tilde{L}(U_1 \otimes U_2)(\Lambda_1 \otimes \Lambda_2)(V_1^T \otimes V_2^T))^{-1}\tilde{L}p \\
& = q^T(I + c\tilde{L}(U_1 \otimes U_2)\tilde{\Lambda}(V_1^T \otimes V_2^T))\tilde{L}p \\
& = q^T\tilde{L}p + cq^T(\sum_{j=1}^{d_n} L_1^j U_1 \otimes L_2^j U_2)\tilde{\Lambda}(V_1^T \otimes V_2^T)\tilde{L}p \\
& = (\sum_{j=1}^{d_n}(q_1^T L_1^{(j)} p_1)(q_2^T L_2^{(j)} p_2)) + c\cdot \\
& (\sum_{j=1}^{d_n} q_1^T L_1^{(j)} U_1 \otimes q_2^T L_2^{(j)} U_2)\tilde{\Lambda}(\sum_{j=1}^{d_n} V_1^T L_1^{(j)} p_1 \otimes V_2^T L_2^{(j)} p_2)
\end{aligned}
$$

where the second equality comes from the Sherman-Morrison Lemma [28]. □

We show the time and the space complexities of Algorithm 3. Note that the time complexity $O(d_n n^2 r^4 + mr + r^6)$ of Ark-L is smaller than the best exact algorithm's complexity $O(m^2 i_F)$ since $n >> r$ and $n >> d_n$.

Theorem 4.2. (Time complexity of Ark-L) Ark-L *takes* $O(d_n n^2 r^4 + mr + r^6)$ *time.*

Proof. The top r decompositions in lines 1 and 2 cost $O(mr)$. Computing $\tilde{\Lambda}$ in line 3 takes $O(d_n n^2 r^4 + r^6)$. Computing line 4, 5 and 6 takes $O(d_n nr + d_n r^2 + r^4)$. □

Theorem 4.3. (Space complexity of Ark-L) Ark-L *requires* $O(m + n^2 r^2)$ *space.*

Proof. The storage of W_1 and W_2 require $O(m)$ space. The top r decompositions in lines 1 and 2 require $O(nr)$. Line 5 to 8 require $O(n^2 r^2)$ space, thus making the total space complexity $O(m + n^2 r^2)$. □

5 Experiments

We perform experiments to answer the following questions.

- Q1 How fast are our Ark algorithms compared to exact methods?
- Q2 What are the accuracies of our Ark algorithms compared to exact methods?
- Q3 How do the accuracies of Ark change with the number of eigenvalues?

Q1 is answered in Section 5.2, and Q(2,3) are answered in Section 5.3.

5.1 Experimental Setting For the exact methods, we run both the conjugate gradient and the fixed point iterations, and choose the one with the smaller running time. We use the graphs in Table 3 with the following details.

- WWW-Barabasi: a Web graph snapshot of nd.edu domain.
- HEP-TH: a citation network in the area of theoretical high energy physics.
- AS-Oregon: a router connection graph.

We use the decay factor $c = 0.1$ for Ark-U and Ark-L. For Ark-U+, we choose $c = 0.0001$ since it was the largest c that allows the fixed point iteration method to converge (see Section 2.2 for more information on the convergence of the fixed point iteration method). All the experiments were performed in a Linux machine with 48 GB memory, and quad-core AMD 2400 MHz CPUs.

Name	Nodes	Edges
WWW-Barabasi	325,729	2,207,671
HEP-TH	27,400	704,036
AS-Oregon	13,579	74,896

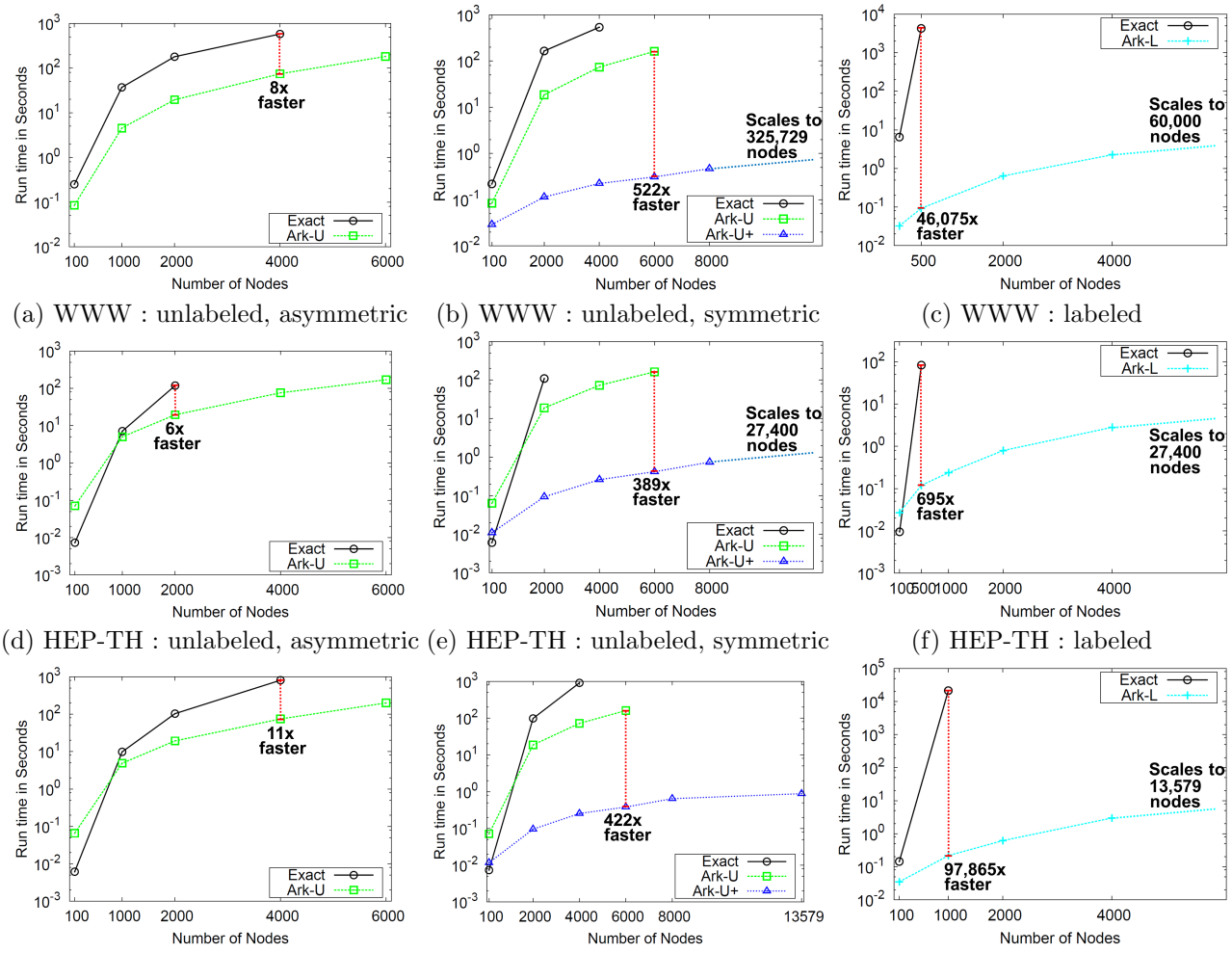
Table 3: Summary of graphs used.

5.2 Scalability We first present the scalability results. For each graph, we extract the principal submatrices (=upper, left part of the adjacency matrix) of different lengths, and compute the graph kernel using the two copies of the extracted subgraph. Figure 1 shows the running time comparison of our approximation vs. exact methods for real world graphs.

Ark-U. In the first column of Figure 1, Ark-U is compared against the exact method on unlabeled, asymmetric graphs. Note that for all the graphs, Ark-U is 6× to 11× faster than the exact method. The exact method is not plotted for all the number of nodes since it failed with the 'out of memory' error.

Ark-U+. In the second column of Figure 1, Ark-U+ is compared against the exact method and Ark-U on unlabeled, symmetric graphs. Note that for all the graphs, Ark-U+ is 389× to 522× faster than the exact and Ark-U method. The exact and Ark-U method is not plotted for all the number of nodes since they failed with the 'out of memory' error.

Ark-L. Finally, in the third column of Figure 1, Ark-L is compared against the exact method. Note that we omitted the plots for exact method beyond 500 data points since they took more than 5 hours.



(a) WWW : unlabeled, asymmetric (b) WWW : unlabeled, symmetric (c) WWW : labeled

(d) HEP-TH : unlabeled, asymmetric (e) HEP-TH : unlabeled, symmetric (f) HEP-TH : labeled

(g) Oregon : unlabeled, asymmetric (h) Oregon : unlabeled, symmetric (i) Oregon : labeled

Figure 1: Running time comparison of our approximation vs. exact methods on real world graphs. The Y axes are in log scale. **(a,d,g)**: ARK-U is 6× to 11× faster than the exact method. The exact method is not plotted for all the number of nodes due to 'out of memory' error. **(b,e,h)**: ARK-U+ is 389× to 522× faster than the exact and ARK-U method. The exact and the ARK-U method is not plotted fully due to 'out of memory' error. **(c,f,i)**: ARK-L is 695× to 97,865× faster than the exact method. We didn't plot the exact method beyond 500 nodes since they took more than 5 hours.

	WWW			HEP-TH			Oregon		
Nodes	**Ark-U**	**Ark-U+**	**Ark-L**	**Ark-U**	**Ark-U+**	**Ark-L**	**Ark-U**	**Ark-U+**	**Ark-L**
100	0.959	0.999	0.980	0.999	0.999	0.999	0.998	0.999	0.999
500	0.957	0.999	0.984	0.977	0.999	0.995	0.959	0.999	0.980
1000	0.930	0.999	*	0.962	0.999	*	0.939	0.999	*
1500	0.920	0.999	*	0.952	0.999	*	0.934	0.999	*
2000	0.913	0.999	*	0.946	0.998	*	0.928	0.999	*

Table 4: Accuracy vs. the number of nodes in our approximation. *: could not compute accuracy since the exact method didn't finish. We fixed the number of eigenvalues to 6. Note the high accuracies for all the graphs, despite the small number of eigenvalues.

	WWW			HEP-TH			Oregon		
r*	**Ark-U**	**Ark-U+**	**Ark-L**	**Ark-U**	**Ark-U+**	**Ark-L**	**Ark-U**	**Ark-U+**	**Ark-L**
1	0.930	0.993	0.971	0.977	0.999	0.996	0.943	0.999	0.970
2	0.930	0.999	0.971	0.977	0.999	0.996	0.947	0.999	0.974
3	0.950	0.999	0.981	0.977	0.999	0.996	0.950	0.999	0.975
4	0.951	0.999	0.981	0.977	0.999	0.996	0.952	0.999	0.976
5	0.953	0.999	0.982	0.977	0.999	0.995	0.954	0.999	0.977
6	0.957	0.999	0.984	0.977	0.999	0.995	0.959	0.999	0.980
7	0.961	0.999	0.986	0.977	0.999	0.995	0.961	0.999	0.981
8	0.962	0.999	0.987	0.977	0.999	0.995	0.961	0.999	0.981
9	0.962	0.999	0.987	0.977	0.999	0.995	0.964	0.999	0.983
10	0.963	0.999	0.987	0.977	0.999	0.994	0.966	0.999	0.984
11	0.965	0.999	0.989	0.977	0.999	0.994	0.967	0.999	0.984
12	0.966	0.999	0.989	0.977	0.999	0.994	0.968	0.999	0.985
13	0.967	0.999	0.989	0.977	0.999	0.994	0.969	0.999	0.986
14	0.969	0.999	0.991	0.977	0.999	0.994	0.970	0.999	0.986
15	0.971	0.999	0.991	0.977	0.999	0.994	0.971	0.999	0.986
16	0.971	0.999	0.991	0.977	0.999	0.994	0.972	0.999	0.987
17	0.972	0.999	0.992	0.977	0.999	0.993	0.974	0.999	0.988
18	0.972	0.999	0.992	0.977	0.999	0.993	0.976	0.999	0.989
19	0.972	0.999	0.992	0.977	0.999	0.993	0.977	0.999	0.989
20	0.973	0.999	0.992	0.977	0.999	0.993	0.977	0.999	0.990

Table 5: Accuracy vs. the number of eigenvalues used for our approximation on real world graphs. **r***: number of eigenvalues for approximation. Note that first few eigenvalues give more than 90% of the accuracy.

Again, ARK-L is 695× to 97,865× faster than the exact method.

5.3 Accuracy We present the accuracy of ARK. The accuracy is defined by the relative error of our approximation with regard to the exact kernel:

$$accuracy = \frac{|\hat{k}(G_1, G_2) - k(G_1, G_2)|}{k(G_1, G_2)}.$$

Accuracy with the Number of Nodes. We fix the number of eigenvalues to 6, and show the accuracy by increasing the number of nodes. Table 4 shows the result. Note that for all the graphs, ARK gives more than 90% accuracies. Note also that only top 6 eigenvalues for 2,000 node graph resulted in more than 91.3% accuracies.

Accuracy with the Number of Eigenvalues. We fix the number of nodes to 500, and show the accuracy by increasing the number of eigenvalues used for the approximation. Table 5 shows the result. Note that for all the graphs, ARK gives more than 90% accuracies. Note also that increasing the number of eigenvalues increase the accuracy.

6 Related Work

The related works branch into three: graph kernel/similarity, node similarity, and low-rank approximation.

Graph Kernel/Similarity. There have been interesting researches in the recent years to classify or measure the similarity between two graphs by kernels. Applications include chemical and biological data classification [19, 29, 31], and outlier detection [14]. Kernels for discrete objects have been studied by Haussler [13], and many graph kernels have been proposed after that. There are three major groups in graph kernels: kernels based on walks and paths based kernels [8, 19, 9, 3, 38, 39], kernels based on limited-size subgraphs [15, 32, 20], and kernels based on subtree patterns [24, 31, 14]. Among them, random walk based kernels have been proved to be one of the most successful methods [5, 4, 30]. Vishwanathan et al [39] proposed a unified theoretic framework for various random walk based kernels. However, all the existing algorithms for random walk based kernel require at least *cubic* running time, which is infeasible for large graphs. Other remotely related work includes [27], which explored different heuristic to measure web graph similarities.

Node Similarity. There are also many works on measuring the similarity between nodes on the *same* graph, e.g., random walk with restart [26, 35], sink-augmented delivered current [7], cycle free effective conductance [21], survivable network [12], direction-aware proximity [34], ObjectRank [2], RelationalRank [10], SimRank [17] and its fast approximation [23]

Low-Rank Approximation. Many real graphs have low intrinsic ranks. Low rank approximation [11, 6, 1] plays a very important role in mining such graphs, e.g., community detection, anomaly detection, etc. For static graphs, the most popular choices include SVD/PCA [11, 18] and random projection [16]. For dynamic graphs, a lot of SVD based techniques have been proposed, such as dynamic tensor analysis [33], incremental spectral clustering [25] etc. More recent work in this line includes example-based low-rank approximation [36], non-negative matrix factorization [40], etc.

7 Conclusions

In this paper, we propose ARK, fast algorithms for computing random walk kernels. The main contributions are the followings.

1. **Algorithms.** We carefully design our algorithms to significantly reduce the time complexity.
2. **Proofs and Analysis.** We give theoretical analysis about the error bound as well as the the correctness and the efficiency of our methods.
3. **Experiments.** We perform numerous experiments on real world graphs, and show that our methods lead to up to 97,865× speed-up with more than 91.3% accuracy.

Future research directions include fast random walk kernel computation on time evolving graphs, and designing parallel, distributed algorithms for very large graphs which do not fit into the memory of a single machine.

Acknowledgments

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] D. Achlioptas and F. McSherry. Fast computation of low-rank matrix approximations. *J. ACM*, 54(2), 2007.

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

[3] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *ICDM*, pages 74–81, 2005.

[4] K. M. Borgwardt, H.-P. Kriegel, S. V. N. Vishwanathan, and N. Schraudolph. Graph kernels for disease outcome prediction from protein-protein interaction networks. In *Pacific Symposium on Biocomputing*, 2007.

[5] K. M. Borgwardt, S. V. N. Vishwanathan, and H.-P. Kriegel. Class prediction from time series gene expression profiles using dynamical systems kernels. In *Pacific Symposium on Biocomputing*, pages 547–558, 2006.

[6] P. Drineas, R. Kannan, and M. W. Mahoney. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. *SIAM Journal of Computing*, 2005.

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

[8] T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *COLT*, pages 129–143, 2003.

[9] T. Gärtner, J. W. Lloyd, and P. A. Flach. Kernels and distances for structured data. *Machine Learning*, 57(3):205–232, 2004.

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

[11] G. H. Golub and C. F. Van-Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2nd edition, 1996.

[12] M. Grötschel, C. L. Monma, and M. Stoer. Design of survivable networks. In *Handbooks in Operations Research and Management Science 7: Network Models*. North Holland, 1993.

[13] D. Haussler. Convlution kernels on discrete structures. *Technical Report*, 1999.

[14] S. Hido and H. Kashima. A linear-time graph kernel. In *ICDM*, pages 179–188, 2009.

[15] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD*, pages 158–167, 2004.

[16] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, pages 189–197, 2000.

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

[18] K. V. R. Kanth, D. Agrawal, and A. K. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD Conference*, pages 166–176, 1998.

[19] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *ICML*, pages 321–

328, 2003.

[20] R. I. Kondor, N. Shervashidze, and K. M. Borgwardt. The graphlet spectrum. In *ICML*, page 67, 2009.

[21] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD*, pages 245–255, 2006.

[22] A. J. Laub. Matrix analysis for scientists and engineers. *SIAM*, 2004.

[23] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu. Fast computation of simrank for static and dynamic information networks. In *EDBT*, pages 465–476, 2010.

[24] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Graph kernels for molecular structure-activity relationship analysis with support vector machines. *Journal of Chemical Information and Modeling*, 45(4):939–951, 2005.

[25] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang. Incremental spectral clustering with application to monitoring of evolving blog communities. In *SDM*, 2007.

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

[27] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, Volume 1(1):19–30, May 2010.

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

[29] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.

[30] H. Saigo, M. Hattori, H. Kashima, and K. Tsuda. Reaction graph kernels predict ec numbers of unknown enzymatic reactions in plant secondary metabolism. *BMC Bioinformatics*, 11(S-1):31, 2010.

[31] N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. *Advances in Neural Information Processing Systems*, 2009.

[32] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. *Journal of Machine Learning Research - Proceedings Track*, 5:488–495, 2009.

[33] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *KDD*, pages 374–383, 2006.

[34] H. Tong, C. Faloutsos, and Y. Koren. Fast direction-aware proximity for graph mining. In *KDD*, pages 747–756, 2007.

[35] H. Tong, C. Faloutsos, and J.-Y. Pan. Random walk with restart: Fast solutions and applications. *Knowledge and Information Systems: An International Journal (KAIS)*, 2008.

[36] H. Tong, S. Papadimitriou, J. Sun, P. S. Yu, and C. Faloutsos. Colibri: fast mining of large static and dynamic graphs. In *KDD*, pages 686–694, 2008.

[37] L. N. Trefethen and D. B. III. Numerical linear algebra. *SIAM*, 1997.

[38] S. V. N. Vishwanathan, K. M. Borgwardt, and N. N. Schraudolph. Fast computation of graph kernels. In *NIPS*, pages 1449–1456, 2006.

[39] S. V. N. Vishwanathan, N. N. Schraudolph, R. I. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

[40] D. Wang, T. Li, and C. H. Q. Ding. Weighted feature subset non-negative matrix factorization and its applications to document understanding. In *ICDM*, pages 541–550, 2010.