Cheetah: Fast Graph Kernel Tracking on Dynamic Graphs

Liangyue Li * Hanghang Tong * Yanghua Xiao† Wei Fan‡

Abstract

Graph kernels provide an expressive approach to measuring the similarity of two graphs, and are key building blocks behind many real-world applications, such as bioinformatics, brain science and social networks . However, current methods for computing graph kernels assume the input graphs are static, which is often not the case in reality. It is highly desirable to track the graph kernels on dynamic graphs evolving over time in a timely manner. In this paper, we propose a family of *Cheetah* algorithms to deal with the challenge. *Cheetah* leverages the low rank structure of graph updates and incrementally updates the eigen-decomposition or SVD of the adjacency matrices of graphs. Experimental evaluations on real world graphs validate our algorithms (1) are significantly faster than alternatives with high accuracy and (b) scale sub-linearly.

1 Introduction

Graph is a natural data structure for modeling a system with interacting objects. It appears in a variety of high-impact application domains, ranging from bioinformatics [3], mobile network [36], brain science [10], transportation networks [8] to social media mining [35]. For instance, in bioinformatics [3], large volume of graph-structured data emerges, e.g., proteins are modeled by graphs comprised of molecules. These graph structures might indicate the function of the proteins. In Internet, the Web itself is a huge graph where nodes are HTML documents and the edges are the hyperlinks. In social media, the graph data is generated at an unprecedented rate. Facebook alone has over 1.32 billion monthly active users [1] . Nodes in social graphs are individuals and edges represent friendship/follow/influence.

Many important graph mining algorithms require a good similarity measure of two graphs. In the above bioinformatics case, protein function prediction can be achieved by comparing to proteins with similar structure and with known function. Graph kernels [32] provide an expressive approach to measuring such similarity. Among others, random walk based graph kernel [12, 19] considers the overall structure of graphs and it works when the node correspondence between two input graphs is unknown (see Section 6 for detailed review). In practice, a major bottleneck for random walk based graph kernel lies in its computational cost, as the exact method takes $O(n^3)$ or $O(m^2)$ time, where n and m are the numbers of nodes and edges of the input graphs, respectively [32]. Many approximate methods exist to speed up its computation. To date, the state-of-the-art algorithm for computing random walk based graph kernel [18] leverages the fact that real world graphs often exhibit much lower intrinsic ranks r compared to their actual size n. Compared with the exact method, it significantly reduces the time complexity to $O(n^2r)$ or $O(mr)$ with a high approximation accuracy.

Nonetheless, all the current methods for computing graph kernels assume the input graphs are static, which is often not the case in reality. For example, hosts on the Web can be down for maintenance, hundreds of thousands new users register on online social networks every day. How to efficiently track the similarity of time evolving graphs is a great challenge. Simply re-calculating the graph kernel at each time step is not realistic for fast decision making. For example, even with the prior fastest method [18], it would still require $O(n^2r)$ or $O(mr)$ at each time stamp to update the graph kernel.

To address the above challenge, we propose a family of fast algorithms (code named *Cheetah*) for tracking the graph kernels of dynamic graphs efficiently. The computational bottleneck of [18] is that the low rank approximation needs to be re-calculated for each new graph, which is very costly in the dynamic setting. We address this by incrementally updating the low rank structure after seeing an incoming graph update. Our algorithms (1) leverage the low rank properties of the graph updates and (2) incrementally and accurately update the low rank approximation in a fast manner. Specifically, for undirected graphs, we propose *Cheetah-U* by incrementally updating the eigenvalue decomposition (EVD); for directed graphs, we design *Cheetah-D* by efficiently updating the singular value decomposition (SVD). The experimental evaluations on real world graphs show that the proposed algorithms (1) are signif-

*Arizona State University. Email: liangyue@asu.edu; hanghang.tong@asu.edu

†Fudan University. Email: shawyh@fudan.edu.cn

‡Big Data Labs - Baidu USA. Email: fanwei03@baidu.com

Table 1: Symbols

Symbols	Definition
$\mathbf{G}$	a graph
$\mathbf{A}^{(t)}$	adjacency matrix at time t
$\mathbf{\Delta A}^{(t)}$	difference matrix of the graph at time t
$\mathbf{U}^{(t)}, \mathbf{\Lambda}^{(t)}$	eigen pair of $\mathbf{A}^{(t)}$
$\text{Ker}^{(t)}(\mathbf{G_1}, \mathbf{G_2})$	exact graph kernel function on graphs $\mathbf{G_1}$ and $\mathbf{G_2}$ at time t
$\hat{\text{Ker}}^{(t)}(\mathbf{G_1}, \mathbf{G_2})$	approximate graph kernel function on graphs $\mathbf{G_1}$ and $\mathbf{G_2}$ at time t
n	number of nodes in a graph
m	number of edges in a graph
c	decay factor in random walk kernel
d_n	number of node labels
r	reduced rank after low rank approximation of $\mathbf{A}^{(t)}$
r'	reduced rank after low rank approximation of $\mathbf{\Delta A}^{(t)}$

icantly faster than the existing alternatives; (2) achieve very high approximation accuracy with proven error bounds and (3) scale sub-linearly.

The main contributions of this paper are summarized as follows:

1. **Problem Definitions**. We define the novel Graph Kernel Tracking problem, to track the kernel of time evolving graphs. To our best knowledge, this is the first effort on this important topic.

2. **Algorithm and Analysis**. We propose a family of fast algorithms (*Cheetah*) for Graph Kernel Tracking and analyze its approximation error bounds as well as the complexity.

3. **Experimental Evaluations**. We perform extensive experiments on real world graphs, to validate the effectiveness and efficiency of our algorithm.

The rest of the paper is organized as follows. Section 2 defines Graph Kernel Tracking . Section 3 and 4 present the proposed *Cheetah* algorithms for both undirected and directed graphs. Section 5 shows the experimental results. After reviewing related work in Section 6, we conclude the paper in Section 7.

2 Problem Definition

Table 1 lists the main symbols used throughout the paper. We use bold upper-case letters for matrices (e.g., $\mathbf{A}$) and bold lower-case letters for vectors (e.g., $\mathbf{v}$). Parenthesized superscript is used to denote time (e.g., $\mathbf{A}^{(t)}$ is the time-aggregate adjacency matrix at time t). For matrix indexing, we use a convention similar to Matlab, e.g., $\mathbf{A}(i, j)$ is the element at the i^{th} row and j^{th} column of the matrix $\mathbf{A}$, and $\mathbf{A}(:, j)$ is the j^{th} column of $\mathbf{A}$, etc. Besides, we use prime for matrix transpose (e.g., $\mathbf{A}'$ is the transpose of $\mathbf{A}$).

For two static graphs $\mathbf{G_1}$ and $\mathbf{G_2}$ with adjacency matrices $\mathbf{A_1}$ and $\mathbf{A_2}$, the random walk based graph kernel between them can be computed as follows [32]:

(2.1)
$$\text{Ker}(\mathbf{G_1}, \mathbf{G_2}) = (\mathbf{q_1}' \otimes \mathbf{q_2}')(\mathbf{I} - c\mathbf{A_1}' \otimes \mathbf{A_2}')^{-1}(\mathbf{p_1} \otimes \mathbf{p_2})$$

where c is a decay factor for discounting longer walks, $\mathbf{p_1}, \mathbf{p_2}$ are starting probabilities for $\mathbf{G_1}, \mathbf{G_2}$ and $\mathbf{q_1}, \mathbf{q_2}$ are ending probabilities for $\mathbf{G_1}, \mathbf{G_2}$. The idea is to sum up all common walks with all possible lengths on the two graphs. The most time consuming part is the matrix inverse. The state-of-the-art algorithm proposed in [18] greatly reduces the computation cost by performing low rank approximation on both $\mathbf{A_1}$ and $\mathbf{A_2}$, following the observation that real world graphs have low intrinsic ranks.

In the dynamic setting, initially at time step $t = 0$, we observe the two graphs $\mathbf{G_1}$ and $\mathbf{G_2}$ and their random walk graph kernel can be computed in the same way as in the static case above. At each time step, both graphs can evolve (e.g., nodes and edges are added/deleted, and edge weight changes). We use $\mathbf{\Delta A}^{(t)}$ to denote such updates of $\mathbf{G}$ at time step t. For example, given a co-author network for an annual conference, $\mathbf{\Delta A}^{(t)}(i, j)$ is the number of papers authors i and j write together for the conference at year t. With these notations, our problem can be formally defined as follows:

Problem 1. Graph Kernel Tracking

Given: *(1) adjacency matrices* $\mathbf{A_1}$ *and* $\mathbf{A_2}$ *of two time-evolving graphs* $\mathbf{G_1}$ *and* $\mathbf{G_2}$ *at initial time step, (2) a sequence of updates* $\mathbf{\Delta A_1}^{(t)}$ *and* $\mathbf{\Delta A_2}^{(t)}, (t = 1, 2, \ldots)$

Track: *the graph kernel* $\text{Ker}^{(t)}(\mathbf{G_1}, \mathbf{G_2}), (t = 1, 2, \ldots)$

As mentioned above, algorithm in [18] speeds up the random walk graph kernel by computing the low rank approximation for both graphs. However, in the dynamic setting, it would be very costly to re-compute the low-rank approximation of the input graphs at each time step. Based on this observation, we devote ourselves to searching for efficient ways to track the low-rank approximation of the input graphs. Depending on whether the input graphs are undirected or directed

graphs, we present two such algorithms in the next two sections, respectively.

3 *Cheetah-U* for Undirected Graphs

In this section, we address GRAPH KERNEL TRACKING for undirected graphs. We first present our proposed algorithm, followed by some analysis in terms of its accuracy as well as complexity.

3.1 The Proposed Algorithm

The heart of our algorithm for undirected graphs is an effective subroutine to track the eigen-decomposition of the adjacency matrices of the two corresponding graphs over time. To be specific, we define the eigen-decomposition tracking problem as follows.

PROBLEM 2. EVD TRACKING

Given: *(1) the adjacency matrix* $\mathbf{A}$ *of a time-evolving undirected graph* $\mathbf{G}$ *at initial time step, (2) a sequence of updates* $\mathbf{\Delta A}^{(t)}, (t = 1, 2, \ldots)$*;*

Track: *the corresponding eigenvectors* $\mathbf{U}^{(t)}$*, and eigenvalues* $\mathbf{\Lambda}^{(t)}, (t = 1, 2, \ldots)$.

Once we have a subroutine *UpdateEigen* to efficiently solve Problem 2, we propose *Cheetah-U* (Algorithm 1) to efficiently solve Problem 1. In *Cheetah-U*, we first obtain the eigen-decomposition of the newly updated graphs using the subroutine *UpdateEigen* without re-calculating the EVD again (line 1,2), which could lead to huge savings in terms of computation time. The new eigen pairs are then used to calculate graph kernel after the updates (line 3-6) using the algorithm in [18]. Notice that in *Cheetah-U* it assumes that the input graphs have no attribute information. We would like to point out that the proposed *Cheetah-U* can be naturally generalized to incorporate the attribute information when it is available.

Now the crucial question becomes how to design an efficient subroutine *UpdateEigen*. In this paper, we propose an effective method for EVD TRACKING , which is summarized in Algorithm 2. The key idea of *UpdateEigen* is as follows. For real world graphs, the updates $\mathbf{\Delta A}$ often have very low ranks (e.g., few nodes being wired to few other nodes), which can be in turn exploited to effectively update EVD. More specifically, we first obtain the eigen-decomposition of the graph update (line 1) and perform a partial QR decomposition on the block matrix composed of original graph's eigenvectors ($\mathbf{U_0}$) and its update matrix's eigenvectors ($\mathbf{X}$) (line 2). Since $\mathbf{U_0}$ is already orthonormal, the QR procedures start from columns in $\mathbf{X}$. We construct a new matrix $\mathbf{Z}$ from the upper triangle matrix of the QR decomposition and the eigenvalues of the graph and its update (line 3) and perform a full EVD of $\mathbf{Z}$ (line 4). $\mathbf{Z}$'s eigenvector rotates the orthonormal basis of the QR decomposition to get the new eigenvectors $\mathbf{U}$ (line 5) and $\mathbf{Z}$'s eigenvalues are the new eigenvalues (line 6).

Notice that there are several alternative choices to update the EVD of a time evolving graph, such as those based on matrix perturbation theory and its high-order variant. However, these methods implicitly assume that the new eigenvectors share the same subspace as that of the old eigenvectors, which could be easily violated in real applications. In contrast, Algorithm 2 does not have such a constraint and thus avoids introducing the additional approximation error during the updating process.

Algorithm 1 *Cheetah-U*: graph kernel tracking for undirected graphs

Input: (1) top r eigen decomposition $\mathbf{U_1}^{(t-1)}$, $\mathbf{\Lambda_1}^{(t-1)}$ and $\mathbf{U_2}^{(t-1)}$, $\mathbf{\Lambda_2}^{(t-1)}$ of $\mathbf{A_1}^{(t-1)}$ and $\mathbf{A_2}^{(t-1)}$; (2) updates $\mathbf{\Delta A_1}^{(t)}$ and $\mathbf{\Delta A_2}^{(t)}$ to $\mathbf{G_1}$ and $\mathbf{G_2}$ at time step t; (3) starting and ending probability $\mathbf{p_1}$ and $\mathbf{q_1}$ for $\mathbf{G_1}$; (4) starting and ending probability $\mathbf{p_2}$ and $\mathbf{q_2}$ for $\mathbf{G_2}$;

Output: graph kernel $\ker^{(t)}(\mathbf{G_1}, \mathbf{G_2})$ at time step t

1: Update eigen decomposition of $\mathbf{A_1}^{(t)}$: $\mathbf{U_1}^{(t)}, \mathbf{\Lambda_1}^{(t)} \leftarrow$ *UpdateEigen*$(\mathbf{U_1}^{(t-1)}, \mathbf{\Lambda_1}^{(t-1)}, \mathbf{\Delta A_1}^{(t)})$
2: Update eigen decomposition of $\mathbf{A_2}^{(t)}$: $\mathbf{U_2}^{(t)}, \mathbf{\Lambda_2}^{(t)} \leftarrow$ *UpdateEigen*$(\mathbf{U_2}^{(t-1)}, \mathbf{\Lambda_2}^{(t-1)}, \mathbf{\Delta A_2}^{(t)})$
3: $\tilde{\mathbf{\Lambda}} \leftarrow ((\mathbf{\Lambda_1}^{(t)} \otimes \mathbf{\Lambda_2}^{(t)})^{-1} - c\mathbf{I})^{-1}$
4: $\mathbf{L} \leftarrow \mathbf{q_1}'\mathbf{U_1}^{(t)} \otimes \mathbf{q_2}'\mathbf{U_2}^{(t)}$
5: $\mathbf{R} \leftarrow \mathbf{U_1}^{(t)'}\mathbf{p_1} \otimes \mathbf{U_2}^{(t)'}\mathbf{p_2}$
6: $\ker^{(t)}(\mathbf{G_1}, \mathbf{G_2}) \leftarrow (\mathbf{q_1}'\mathbf{p_1})(\mathbf{q_2}'\mathbf{p_2}) + c\mathbf{L}\tilde{\mathbf{\Lambda}}\mathbf{R}$

Algorithm 2 *UpdateEigen* (subroutine for EVD Tracking)

Input: Eigen decomposition of $\mathbf{A_0}$: $\mathbf{U_0}, \mathbf{\Lambda_0}$, update $\mathbf{\Delta A}$

Output: Eigen decomposition of $\mathbf{A} = \mathbf{A_0} + \mathbf{\Delta A}$: $\mathbf{U}, \mathbf{\Lambda}$

1: Eigen decomposition of $\mathbf{\Delta A}$: $\mathbf{XYX}' \leftarrow \mathbf{\Delta A}$
2: Perform Partial QR decomposition of $[\mathbf{U_0}, \mathbf{X}]$: $[\mathbf{U_0}, \mathbf{\Delta Q}]\mathbf{R} \leftarrow \mathrm{QR}(\mathbf{U_0}, \mathbf{X})$
3: Set $\mathbf{Z} = \mathbf{R}[\mathbf{\Lambda_0}\ \mathbf{0}; \mathbf{0}\ \mathbf{Y}]\mathbf{R}'$
4: Perform full eigen decomposition of $\mathbf{Z}$: $\mathbf{V\Lambda V}' \leftarrow \mathbf{Z}$
5: Set $\mathbf{U} \leftarrow [\mathbf{U_0}, \mathbf{\Delta Q}]\mathbf{V}$
6: **Return:** $\mathbf{U}$ and $\mathbf{\Lambda}$

3.2 Proofs and Analysis

In this subsection, we provide some analysis of the

proposed *Cheetah-U* algorithm in terms of its accuracy and complexity. Let us start with the accuracy of the subroutine *UpdateEigen*, which is summarized in Lemma 3.1. According to Lemma 3.1, the only place that we might introduce the approximation error is the initial eigen-decomposition for $\mathbf{A}_0$; and updating process itself will not introduce additional error.

LEMMA 3.1. *(Correctness of UpdateEigen). If* $\mathbf{A}_0 = \mathbf{U}_0\mathbf{\Lambda}_0\mathbf{U}_0'$ *holds, algorithm 2 gives the exact eigen-decomposition of the udpated graph* $\mathbf{A}$.

Proof. For a undirected graph, both $\mathbf{A_0}, \mathbf{\Delta A}$ are symmetric, we can write their eigen-decomposition as follows:

$$\mathbf{A_0} = \mathbf{U_0}\mathbf{\Lambda_0}\mathbf{U_0}'$$
$$\mathbf{\Delta A} = \mathbf{X}\mathbf{Y}\mathbf{X}',$$

where $\mathbf{U_0}$, $\mathbf{\Lambda_0}$ are the eigen pairs of $\mathbf{A}_0$ and $\mathbf{X}$, $\mathbf{Y}$ are the eigen pairs of $\mathbf{\Delta A}$. After the update, we have following equation:

$$\begin{aligned}\mathbf{A} &= \mathbf{A_0} + \mathbf{\Delta A} \\ &= \mathbf{U_0}\mathbf{\Lambda_0}\mathbf{U_0}' + \mathbf{X}\mathbf{Y}\mathbf{X}' \\ &= \begin{bmatrix}\mathbf{U_0} & \mathbf{X}\end{bmatrix}\begin{bmatrix}\mathbf{\Lambda_0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Y}\end{bmatrix}\begin{bmatrix}\mathbf{U_0} & \mathbf{X}\end{bmatrix}'\end{aligned}$$

Denoting $[\mathbf{U_0}\ \mathbf{X}]$ by $\tilde{\mathbf{U}}$, we perform a decomposition on $\tilde{\mathbf{U}}$ similar to QR decomposition and have

$$\tilde{\mathbf{U}} = \begin{bmatrix}\mathbf{U_0} & \mathbf{\Delta Q}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{R_1} \\ \mathbf{0} & \mathbf{R_2}\end{bmatrix},$$

where $[\mathbf{U_0}\ \ \mathbf{\Delta Q}]$ is orthonormal and $\begin{bmatrix}\mathbf{I} & \mathbf{R_1} \\ \mathbf{0} & \mathbf{R_2}\end{bmatrix}$ is an upper triangle matrix. Note the difference of the decomposition here from standard QR decomposition is that since $\mathbf{U_0}$ is already orthonormal, we only need to start from the first column of $\mathbf{X}$ to perform the Gram-Schmidt procedure. It follows that

$$\begin{aligned}\mathbf{A} &= \tilde{\mathbf{U}}\begin{bmatrix}\mathbf{\Lambda_0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Y}\end{bmatrix}\tilde{\mathbf{U}}' \\ &= \begin{bmatrix}\mathbf{U_0} & \mathbf{\Delta Q}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{R_1} \\ \mathbf{0} & \mathbf{R_2}\end{bmatrix}\begin{bmatrix}\mathbf{\Lambda_0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Y}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{R_1} \\ \mathbf{0} & \mathbf{R_2}\end{bmatrix}'\begin{bmatrix}\mathbf{U_0}' \\ \mathbf{\Delta Q}'\end{bmatrix}\end{aligned}$$

Denoting $\begin{bmatrix}\mathbf{I} & \mathbf{R_1} \\ \mathbf{0} & \mathbf{R_2}\end{bmatrix}\begin{bmatrix}\mathbf{\Lambda_0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Y}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{R_1} \\ \mathbf{0} & \mathbf{R_2}\end{bmatrix}'$ by $\mathbf{Z}$, we do a full eigen decomposition on it and have $\mathbf{Z} = \mathbf{V}\mathbf{L}\mathbf{V}'$, where $\mathbf{V}$ and $\mathbf{L}$ are its eigen pairs. Therefore, the updated graph $\mathbf{A}$ can be written as

$$\begin{aligned}\mathbf{A} &= \underbrace{\begin{bmatrix}\mathbf{U_0} & \mathbf{\Delta Q}\end{bmatrix}\mathbf{V}}_{\mathbf{U}}\underbrace{\mathbf{L}}_{\mathbf{\Lambda}}\underbrace{\mathbf{V}'\begin{bmatrix}\mathbf{U_0}' \\ \mathbf{\Delta Q}'\end{bmatrix}}_{\mathbf{U}'} \\ &= \mathbf{U}\mathbf{\Lambda}\mathbf{U}',\end{aligned}$$

where $\mathbf{U}$ and $\mathbf{\Lambda}$ will be the new eigen pairs of the updated graph $\mathbf{A}$.

Summarizing the above procedures, we have the exact EVD update algorithm in Algorithm 2.

Next, we analyze the tracking quality of *Cheetah-U*, which is summarized in Theorem 3.1.

THEOREM 3.1. *(Error Bound of Cheetah-U) In Cheetah-U, if we use the subroutine UpdateEigen, the relative error of the approximate random walk kernel after one update is bounded by:*

$$RelErr \leq \frac{f(c,\delta)}{(1-c(\lambda_1^{(1)}+\delta)(\lambda_2^{(1)}+\delta))g(c,\eta)-f(c,\delta)} \tag{3.2}$$

where $RelErr = \frac{|\mathrm{Ker}^{(1)}(\mathbf{G_1},\mathbf{G_2}) - \hat{\mathrm{Ker}}^{(1)}(\mathbf{G_1},\mathbf{G_2})|}{\mathrm{Ker}^{(1)}(\mathbf{G_1},\mathbf{G_2})}$, $f(c,\delta) = c\sum_{(i,j)\notin\mathcal{H}}\lambda_1^{(i)}\lambda_2^{(j)} + c\delta\sum_{i\notin\mathcal{H}}(\lambda_1^{(i)} + \lambda_2^{(i)})$, $g(c,\delta) = \sqrt{c^2(\lambda_1^{(1)}-\delta)^2(\lambda_2^{(1)}-\delta)^2 + n^2}$, $\delta = \max(\|\mathbf{\Delta A_1}\|_F, \|\mathbf{\Delta A_2}\|_F)$,$\lambda_1^{(i)}$ *and* $\lambda_2^{(i)}$ *are the* i*-th largest eigenvalues of* $\mathbf{A_1}$ *and* $\mathbf{A_2}$*, and* $\mathcal{H} = \{(a,b)|a,b \in [1,r]\}$.

Proof. To calculate the exact kernel after one update, we have the following equation:

$$\mathrm{Ker}^{(1)}(\mathbf{G_1},\mathbf{G_2}) = \mathbf{q}'(\mathbf{I} - c(\mathbf{A_1}+\mathbf{\Delta A_1})\otimes(\mathbf{A_2}+\mathbf{\Delta A_2}))^{-1}\mathbf{p} \tag{3.3}$$

where $\mathbf{q}' = \mathbf{q_1}' \otimes \mathbf{q_2}'$ and $\mathbf{p} = \mathbf{p_1}\otimes\mathbf{p_2}$, $\|\mathbf{p}\|_1 = \|\mathbf{q}\|_1 = 1$, $\mathbf{A_1}, \mathbf{A_2}$ are the adjacency matrices of the original two graphs $\mathbf{G_1}, \mathbf{G_2}$ and $\mathbf{\Delta A_1}, \mathbf{\Delta A_2}$ are their updates. Using *UpdateEigen* in *Cheetah-U* by Lemma 3.1, the only error introduced is the low rank approximations of $\mathbf{A_1}$ and $\mathbf{A_2}$. Therefore, our approximated kernel can be computed as:

$$\hat{\mathrm{Ker}}^{(1)}(\mathbf{G_1},\mathbf{G_2}) = \mathbf{q}'(\mathbf{I} - c(\hat{\mathbf{A_1}}+\mathbf{\Delta A_1})\otimes(\hat{\mathbf{A_2}}+\mathbf{\Delta A_2}))^{-1}\mathbf{p} \tag{3.4}$$

where $\hat{\mathbf{A_1}}, \hat{\mathbf{A_2}}$ are rank r-approximations of $\mathbf{A_1}, \mathbf{A_2}$. Let $\mathbf{M}$ be $\mathbf{I} - c(\mathbf{A_1}+\mathbf{\Delta A_1})\otimes(\mathbf{A_2}+\mathbf{\Delta A_2})$ and $\hat{\mathbf{M}}$ be $\mathbf{I} - c(\hat{\mathbf{A_1}}+\mathbf{\Delta A_1})\otimes(\hat{\mathbf{A_2}}+\mathbf{\Delta A_2})$. The Frobenius norm of their difference matrix is upper bounded:

$$\begin{aligned}&\|\mathbf{M} - \hat{\mathbf{M}}\|_F \\ &= \|c(\mathbf{A_1}+\mathbf{\Delta A_1})\otimes(\mathbf{A_2}+\mathbf{\Delta A_2}) \\ &-c(\hat{\mathbf{A_1}}+\mathbf{\Delta A_1})\otimes(\hat{\mathbf{A_2}}+\mathbf{\Delta A_2})\|_F \\ &= \|c(\mathbf{A_1}\otimes\mathbf{A_2} - \hat{\mathbf{A_1}}\otimes\hat{\mathbf{A_2}}) + c(\mathbf{A_1}-\hat{\mathbf{A_1}})\otimes\mathbf{\Delta A_2} \\ &+c\mathbf{\Delta A_1}\otimes(\mathbf{A_2}-\hat{\mathbf{A_2}})\|_F \\ &\leq \|c(\mathbf{A_1}\otimes\mathbf{A_2} - \hat{\mathbf{A_1}}\otimes\hat{\mathbf{A_2}})\|_F + c\|\mathbf{A_1}-\hat{\mathbf{A_1}}\|_F\|\mathbf{\Delta A_2}\|_F \\ &+c\|\mathbf{\Delta A_1}\|_F\|\mathbf{A_2}-\hat{\mathbf{A_2}}\|_F \\ &\leq c\sum_{(i,j)\notin\mathcal{H}}\lambda_1^{(i)}\lambda_2^{(j)} + c\delta\sum_{i\notin\mathcal{H}}(\lambda_1^{(i)}+\lambda_2^{(i)})\end{aligned} \tag{3.5}$$

where $\delta = \max(\|\mathbf{\Delta A_1}\|_F, \|\mathbf{\Delta A_2}\|_F)$.

We know that

$$\begin{aligned}\lambda_{\max}(\mathbf{A_1}+\mathbf{\Delta A_1}) &= \|\mathbf{A_1}+\mathbf{\Delta A_1}\|_2 \\ &\le \|\mathbf{A_1}\|_2 + \|\mathbf{\Delta A_1}\|_2 \\ &\le \lambda_1^{(1)} + \delta\end{aligned} \tag{3.6}$$

Therefore, the condition number of $\mathbf{M}$ is also upper bounded:$\kappa(\mathbf{M}) \le \frac{1}{1-c(\lambda_1^{(1)}+\delta)(\lambda_2^{(1)}+\delta)}$.

On the other hand, by triangle inequality,

$$\begin{aligned}&\|(\mathbf{A_1}+\mathbf{\Delta A_1})\otimes(\mathbf{A_2}+\mathbf{\Delta A_2})\|_F \\ &= \|\mathbf{A_1}+\mathbf{\Delta A_1}\|_F\|\mathbf{A_2}+\mathbf{\Delta A_2}\|_F \\ &\ge (\lambda_1^{(1)}-\delta)(\lambda_2^{(1)}-\delta)\end{aligned} \tag{3.7}$$

Since we don't consider graphs with self-loops, i.e., adjacency matrices here have all zeros on the diagonal, it follows that

$$\|\mathbf{M}\|_F \ge \sqrt{c^2(\lambda_1^{(1)}-\delta)^2(\lambda_2^{(1)}-\delta)^2+n^2}. \tag{3.8}$$

From matrix perturbation analysis [14], we have the upper bound for the relative error:

$$\begin{aligned}&\frac{|\mathrm{Ker}^{(1)}(\mathbf{G_1},\mathbf{G_2})-\hat{\mathrm{Ker}}^{(1)}(\mathbf{G_1},\mathbf{G_2})|}{\mathrm{Ker}^{(1)}(\mathbf{G_1},\mathbf{G_2})} \\ &= \frac{\mathbf{q}'(\mathbf{M}^{-1}-\hat{\mathbf{M}}^{-1})\mathbf{p}}{\mathbf{q}'\hat{\mathbf{M}}^{-1}\mathbf{p}} \\ &\le \frac{\|\mathbf{M}^{-1}-\hat{\mathbf{M}}^{-1}\|_F}{\|\mathbf{M}^{-1}\|_F} \\ &\le \frac{\kappa(\mathbf{M})\frac{\|\mathbf{M}-\hat{\mathbf{M}}\|_F}{\|\mathbf{M}\|_F}}{1-\kappa(\mathbf{M})\frac{\|\mathbf{M}-\hat{\mathbf{M}}\|_F}{\|\mathbf{M}\|_F}} \\ &\le \frac{f(c,\delta)}{(1-c(\lambda_1^{(1)}+\delta)(\lambda_2^{(1)}+\delta))g(c,\eta)-f(c,\delta)}\end{aligned} \tag{3.9}$$

Finally, we analyze the complexities of Algorithm 1 and 2. As can be seen from Theorem 3.2, both algorithms have linear time and space complexities wrt the size of graph n. r and r' are reduced rank of $\mathbf{A}$ and $\mathbf{\Delta A}$ respectively, which are small constants. Therefore, the algorithms are scalable for large graphs.

THEOREM 3.2. *(Complexities of Cheetah-U and UpdateEigen) Algorithm 2 takes $O(n(r^2+r'^2))$ time and $O(n(r+r'))$ space. Algorithm 1 takes $O(n(r^2+r'^2)+r^2)$ time and $O(n(r+r')+r'^2)$ space.*

Proof. Omitted for brevity.

4 *Cheetah-D* for Directed Graphs

In this section, we address GRAPH KERNEL TRACKING for directed graphs. We first present the proposed algorithm, followed by some complexity analysis.

4.1 The Proposed Algorithm

Similar as in the undirected graph case, an effective subroutine to track SVD of the adjacency matrices of the two corresponding directed graphs over time is needed. To be specific, we define the SVD tracking problem as follows:

PROBLEM 3. SVD TRACKING

Given: *(1) the adjacency matrix $\mathbf{A}$ of a time-evolving directed graph $\mathbf{G}$ at initial time step, (2) a sequence of updates $\mathbf{\Delta A}^{(t)}, (t=1,2,\ldots)$;*

Track: *the corresponding left and right singular vectors $\mathbf{U}^{(t)}$,$\mathbf{V}^{(t)}$ and singular values $\mathbf{\Lambda}^{(t)}, (t=1,2,\ldots)$.*

We propose an effective method for SVD TRACKING as summarized in Algorithm 4. The key idea is similar to *UpdateEigen*. The difference is that SVD is used for exploiting the low rank structure of the graph updates instead of EVD. Once we have a subroutine *UpdateSVD* to efficiently solve Problem 3, we propose *Cheetah-D* (Algorithm 3) to efficiently solve Problem 1. In *Cheetah-D*, we first obtain SVD of the updated graphs (line 1,2) and then use that for graph kernel computation using algorithm in [18]. Note that *Cheetah-D* can also be generalized to incorporate the attribute information.

Algorithm 3 *Cheetah-D*: graph kernel tracking for directed graphs

Input: (1) top r SVD $\mathbf{U_1}^{(t-1)}$, $\mathbf{\Lambda_1}^{(t-1)}$, $\mathbf{V_1}^{(t-1)}$ and $\mathbf{U_2}^{(t-1)}$, $\mathbf{\Lambda_2}^{(t-1)}$, $\mathbf{V_2}^{(t-1)}$ of $\mathbf{A_1}^{(t-1)}$ and $\mathbf{A_2}^{(t-1)}$; (2) updates $\mathbf{\Delta A_1}^{(t)}$ and $\mathbf{\Delta A_2}^{(t)}$ to $\mathbf{G_1}$ and $\mathbf{G_2}$ at time step t; (3) starting and ending probability $\mathbf{p_1}$ and $\mathbf{q_1}$ for $\mathbf{G_1}$; (4) starting and ending probability $\mathbf{p_2}$ and $\mathbf{q_2}$ for $\mathbf{G_2}$;

Output: $\ker^{(t)}(\mathbf{G_1},\mathbf{G_2})$ at time step t

1: Update SVD of $\mathbf{A_1}^{(t)}$: $\mathbf{U_1}^{(t)},\mathbf{\Lambda_1}^{(t)},\mathbf{V_1}^{(t)} \leftarrow$ *UpdateSVD*$(\mathbf{U_1}^{(t-1)},\mathbf{\Lambda_1}^{(t-1)},\mathbf{V_1}^{(t-1)},\mathbf{\Delta A_1}^{(t)})$
2: Update SVD of $\mathbf{A_2}^{(t)}$: $\mathbf{U_2}^{(t)},\mathbf{\Lambda_2}^{(t)},\mathbf{V_2}^{(t)} \leftarrow$ *UpdateSVD*$(\mathbf{U_2}^{(t-1)},\mathbf{\Lambda_2}^{(t-1)},\mathbf{V_2}^{(t-1)},\mathbf{\Delta A_2}^{(t)})$
3: $\tilde{\mathbf{\Lambda}} \leftarrow ((\mathbf{\Lambda_1}^{(t)}\otimes\mathbf{\Lambda_2}^{(t)})^{-1} - c(\mathbf{V_1}^{(t)'}\otimes\mathbf{V_2}^{(t)'})(\mathbf{U_1}^{(t)}\otimes\mathbf{U_2}^{(t)}))^{-1}$
4: $\mathbf{L} \leftarrow \mathbf{q_1}'\mathbf{U_1}^{(t)}\otimes\mathbf{q_2}'\mathbf{U_2}^{(t)}$
5: $\mathbf{R} \leftarrow \mathbf{V_1}^{(t)'}\mathbf{p_1}\otimes\mathbf{V_2}^{(t)'}\mathbf{p_2}$
6: $\ker^{(t)}(\mathbf{G_1},\mathbf{G_2}) \leftarrow (\mathbf{q_1}'\mathbf{p_1})(\mathbf{q_2}'\mathbf{p_2}) + c\mathbf{L}\tilde{\mathbf{\Lambda}}\mathbf{R}$

4.2 Proofs and Analysis

In this subsection, we begin with the correctness proof of subroutine *UpdateSVD* summarized in Lemma 4.1, followed by complexity analysis.

LEMMA 4.1. *(Correctness of UpdateSVD). If $\mathbf{A}_0 = \mathbf{U}_0\mathbf{\Lambda}_0\mathbf{V}_0'$ holds, algorithm 4 gives the exact singular value decomposition of the udpated graph $\mathbf{A}$.*

Algorithm 4 *UpdateSVD* (subroutine for SVD Tracking)

Input: SVD of $\mathbf{A_0}$: $\mathbf{U_0}, \mathbf{\Lambda_0}, \mathbf{V_0}$, update $\mathbf{\Delta A}$
Output: SVD of $\mathbf{A} = \mathbf{A_0} + \mathbf{\Delta A}$: $\mathbf{U}, \mathbf{\Lambda}, \mathbf{V}$

1: SVD of $\mathbf{\Delta A}$: $\mathbf{XYZ}' \leftarrow \mathbf{\Delta A}$
2: Perform Partial QR decomposition of $[\mathbf{U_0}, \mathbf{X}]$: $[\mathbf{U_0}, \mathbf{\Delta Q}]\mathbf{S} \leftarrow \text{QR}(\mathbf{U_0}, \mathbf{X})$
3: Perform Partial QR decomposition of $[\mathbf{V_0}, \mathbf{Z}]$: $[\mathbf{V_0}, \mathbf{\Delta Z}]\mathbf{T} \leftarrow \text{QR}(\mathbf{V_0}, \mathbf{Z})$
4: Set $\mathbf{W} = \mathbf{S}[\mathbf{\Lambda_0}\ \mathbf{0}; \mathbf{0}\ \mathbf{Y}]\mathbf{T}'$
5: Perform Full SVD of $\mathbf{W}$: $\mathbf{L\Lambda R}' \leftarrow \mathbf{W}$
6: Set $\mathbf{U} \leftarrow [\mathbf{U_0}, \mathbf{\Delta Q}]\mathbf{L}$
7: Set $\mathbf{V} \leftarrow [\mathbf{V_0}, \mathbf{\Delta Z}]\mathbf{R}$
8: **Return:** $\mathbf{U}, \mathbf{\Lambda}, \mathbf{V}$

Proof. Omitted for brevity.

THEOREM 4.1. *(Complexities of Cheetah-D and UpdateSVD) Algorithm 4 takes $O(n(r^2 + r'^2))$ time and $O(n(r + r'))$ space. Algorithm 3 takes $O(n^2r^4 + n(r^2 + r'^2) + r^6)$ time and $O(n^2r^2 + n(r + r') + r'^2)$ space.*

Proof. Omitted for brevity.

5 Experiment

In this section, we present the experimental results for the proposed *Cheetah*. The experiments are designed to evaluate the following aspects:

- *Effectiveness:* How accurate is our algorithm for tracking graph kernels over time?
- *Efficiency:* How fast is our proposed algorithm?

5.1 Datasets We use two real world dynamic graphs for case study and performance evaluations as follows:

- **MTA bus traffic.** We collect real time bus traffic data in New York City using the API provided at MTA Bus Time [1]. Traffic volume at 30 bus stops on 3 routes are monitored from Monday, March 24, 2014 to Sunday, March 30, 2014. On each day, we first obtain traffic volume within each hour as a time series for each bus stop and then build a causality graph for these 30 stops using Granger causality test [15].
- **AS.** This is the communication network of routers constructed by BGP logs in Autonomous Systems (AS) [21]. The dataset contains 733 daily instances which span an interval of 785 days from November 8, 1997 to January 2, 2000. AS exhibits both addition and deletion of nodes and edges over the time span. The number of nodes ranges from 103 to 6474 and the number of edges ranges from 243 to 13,233.

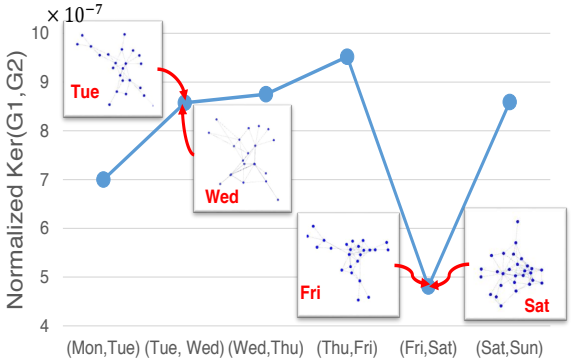

Figure 1: Case study – real time MTA bus traffic. Causality graphs are shown in the small blocks.

5.2 Effectiveness Results

Case study on MTA bus traffic: Normalized graph kernels[2] are computed on two graphs of two consecutive days, e.g., kernels of Monday and Tuesday, Tuesday and Wednesday. Figure 1 shows the trend of kernels over a week. Kernels between weekdays change smoothly. We observe a sharp drop of the kernel between Friday and Saturday, which reflects the fact that traffic patterns on weekdays and weekends are different since MTA runs completely different bus schedules during weekdays and weekend. The kernel goes up on Sunday because Saturday and Sunday share similar traffic patterns.

Accuracy vs. time stamp: In order to evaluate how accurate our method is for tracking graph kernels, we extract two graphs from AS, each of size $n = 3328$. At each time stamp, we randomly pick 50 nodes and add an edge from each to 100 other random nodes. We use relative error computed as below for our evaluation criteria:

$$\text{Relative Error} = \frac{|\text{Ker}(\mathbf{G_1}, \mathbf{G_2}) - \hat{\text{Ker}}(\mathbf{G_1}, \mathbf{G_2})|}{\text{Ker}(\mathbf{G_1}, \mathbf{G_2})} \tag{5.10}$$

Figure 2 shows relative error of *Cheetah-U* at different time stamps with different reduced rank r while r' is fixed. Here r' is the reduced rank of update matrix, i.e., we perform top-r' eigen decomposition on $\mathbf{\Delta A}$ in *UpdateEigen*. Similar trend is seen with different r' while r is fixed. The figure clearly shows (1) the accumulated error of our method grows slowly (sublinearly) over time; and (2) the overall accumulated

[1] Available at `http://bustime.mta.info`

[2] The graph kernel is normalized by the number of edges.

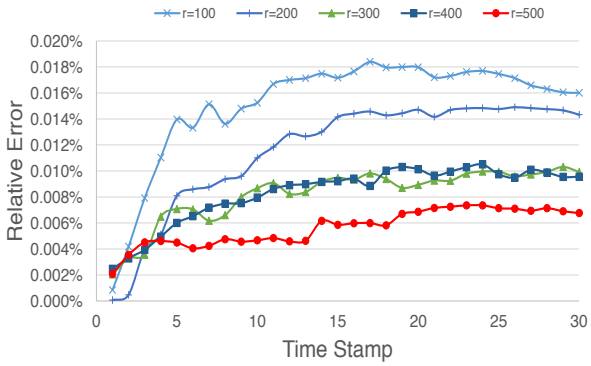

Figure 2: Relative error of *Cheetah-U* via *UpdateEigen* on AS at different time stamp with different r.

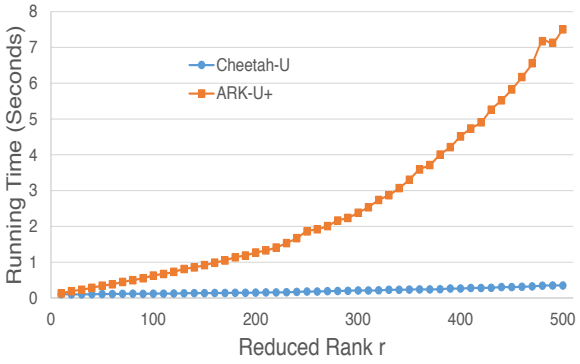

Figure 4: Running time of *Cheetah-U* on AS with different reduced rank r.

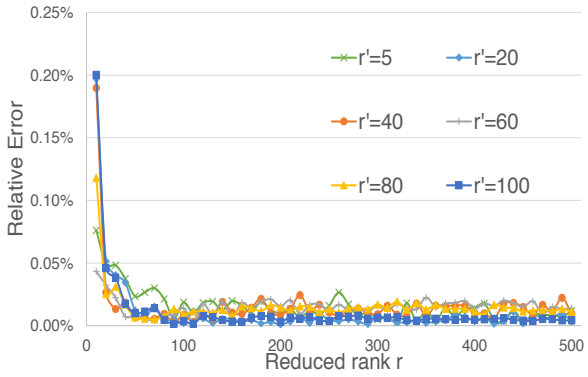

Figure 3: Average error vs. reduced rank r. Each curve has different reduced rank r' for the update matrix in *UpdateEigen*.

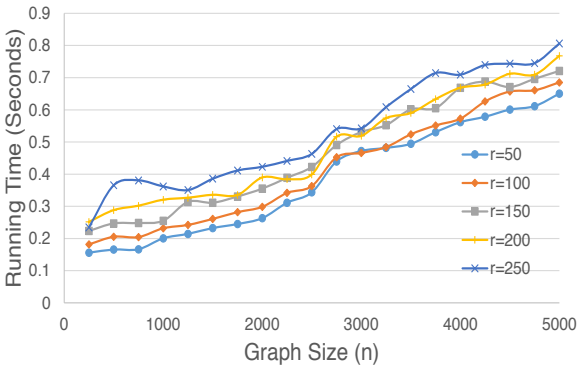

Figure 5: Running time of *Cheetah-U* on AS with different graph size n and reduced rank r.

error is very small (less than 0.02%). Notice that, results using the alternative methods for updating eigen pairs (referred to as 'first-order' and 'second-order') are not shown here since even at $t = 1$ the error is in the order of 10^4.

Accuracy vs. rank: In order to evaluate how accuracy of *Cheetah-U* changes with respect to the reduced rank r, we run the above experiment under different r and average the relative error over 10 time stamps. To see how the approximation of the updates affects the accuracy, we also vary the reduced rank r'. As can be seen from Figure 3, the error quickly drops when r increases.

5.3 Efficiency Results

Running time vs. rank: We compare the speed of *Cheetah-U* with ARK-U+ proposed in [18] varying reduced rank r and average the running time over 10 time stamps. We set reduced rank of update matrix as $r' = 5$. Figure 4 clearly shows that our method is much faster than ARK-U+.

Scalability: In order to evaluate the scalability of our method, we run *Cheetah-U* on graphs with different sizes n. Figure 5 shows the running time under different r while fixing $r' = 5$. Similar trend is seen with different r' while fixing r. From the figure, we can see that the running time grows linearly wrt the size of the input graphs, which is consistent with our complexity analysis in Theorem 3.2.

Quality vs. speed: Finally, we evaluate how the proposed method balances between the quality and speed. In Figure 6, we show relative error vs. running time of different methods. Each dot in the figure is with different reduced rank r. Clearly, our method achieves the best trade-off between quality and time.

6 Related Work

In this section, we review the related work in terms of (a) graph kernel, (b) dynamic graph mining.

Graph Kernel. Graph kernel provides an expressive and non-trivial measure of similarity on graphs

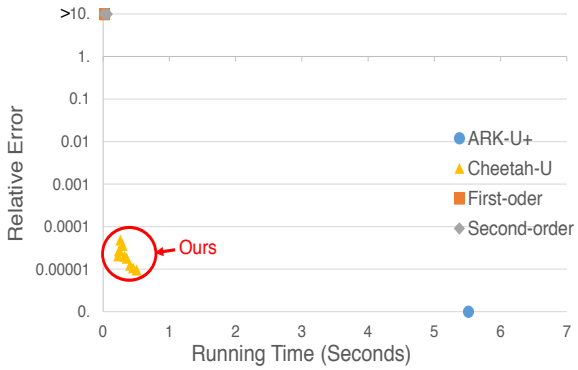

Figure 6: Relative error vs. running time of comparison methods on AS.

(see [4] for a comprehensive review). It has seen applications ranging from automated reasoning [31] to bioinformatics/chemoinformatics [11, 26]. A recent interesting work uses graph kernel to address team member replacement problem [22]. According to what substructures used for comparison in two graphs, graph kernels can be summarized into three categories: kernels based on walks [12, 32, 33, 13, 5], kernels based on limited-sized subgraphs [17, 25, 20] and kernels based on subtree patterns [23, 24, 16]. Among them, graph kernel based on random walk has been successfully applied in many real world scenarios [6]. The idea is to count the number of common walks when simultaneous walks are performed on the two graphs. One challenge of random walk based graph kernel lies in computational cost. The best known time complexity for exact computation is $O(n^3)$ by reducing to the problem of solving a linear system [32, 33]. With low rank approximation, the computation can be further accelerated with high approximation accuracy [18].

Dynamic Graph Mining. Most real world graphs are evolving over time, hence it's of practical value to track some properties of the dynamic graphs, and do it in an efficient way. To track the low-rank approximation of graphs, CMD [28] computes sparse example-based decompositions by sampling from the original matrix without duplications. *Colibri* methods in [29] further speed up the computation by judiciously sampling linearly independent columns. Evolutionary Nonnegative Matrix Factorization (eNMF) [34] incrementally updates the factorized matrices assuming smoothness between two consecutive time stamps. Proximity and centrality are two important measures on graphs. To monitor these, fast algorithms on bipartite graphs are designed [30] by leveraging the fact that rank of graph updates is small. Our work differs from [30] in that we track the similarity of two graphs while authors in [30] focus on similarity of two nodes on one graph.

As for communities in dynamic graphs, work include studying how social groups form and evolve [2], finding communities in dynamic graphs and spotting discontinuity time points [27]. On a single dynamic graph, there are also many work on tracking its spectrum [7, 9].

7 Conclusion

In this paper, we propose *Cheetah* to efficiently track the graph kernels of two time-evolving graphs. To the best of our knowledge, we are the first to study kernel tracking in dynamic setting. The main contributions include:

1. **Problem Definitions.** A novel Graph Kernel Tracking problem is first defined, along with two derivative problems:EVD Tracking and SVD Tracking .
2. **Algorithm and analysis.** A family of *Cheetah* algorithms are proposed to address the above problems. We show the correctness and analyze the complexities of the algorithms.
3. **Experimental Evaluations.** Case study and performance evaluation on real world data present the usefulness and superiority of our algorithms.

Our work can be generalized to attributed graphs while such attribute information remains the same. However, in reality, attributes can also change with time, e.g., in citation network, an author's interest might shift from computer vision to data mining. One future direction is to design algorithms for graph kernel tracking that can also capture such attribute dynamics.

8 Acknowledgment

This material is supported by the National Science Foundation under Grant No. IIS1017415, by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053, by Defense Advanced Research Projects Agency (DARPA) under Contract Number W911NF-11-C-0200 and W911NF-12-C-0028, by National Institutes of Health under the grant number R01LM011986, Region II University Transportation Center under the project number 49997-33 25. Yanghua Xiao was partially supported by the National NSFC(No.61472085, 61171132, 61033010), by National Key Basic Research Program of China under No.2015CB358800, by Shanghai STCF under No.13511505302, by NSF of Jiangsu Prov. under No. BK2010280.

The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] Facebook information. http://newsroom.fb.com/company-info.

[2] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54, 2006.

[3] D. A. Bader and K. Madduri. A graph-theoretic analysis of the human protein-interaction network using multicore parallel algorithms. *Parallel Computing*, 34 (11):627–639, 2008.

[4] K. M. Borgwardt. *Graph kernels*. PhD thesis, Ludwig Maximilians University Munich, 2007.

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

[6] 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.

[7] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and Its Applications*, pages 20–30, 2006.

[8] C. Chen, K. Petty, A. Skabardonis, P. Varaiya, and Z. Jia. Freeway performance measurement system: mining loop detector data. *Transportation Research Record: Journal of the Transportation Research Board*, 1748(1):96–102, 2001.

[9] X. Chen and K. S. Candan. LWI-SVD: low-rank, windowed, incremental singular value decompositions on time-evolving data sets. In *KDD*, pages 987–996, 2014.

[10] C. Faloutsos, D. Koutra, and J. T. Vogelstein. DELTACON: A principled massive-graph similarity function. In *SDM*, pages 162–170, 2013.

[11] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. M. Borgwardt. Scalable kernels for graphs with continuous attributes. In *NIPS*, pages 216–224, 2013.

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

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

[14] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8.

[15] C. W. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica: Journal of the Econometric Society*, pages 424–438, 1969.

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

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

[18] U. Kang, H. Tong, and J. Sun. Fast random walk graph kernel. In *SDM*, pages 828–838, 2012.

[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] J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.

[22] L. Li, H. Tong, N. Cao, K. Ehrlich, Y.-R. Lin, and N. Buchler. Replacing the irreplaceable: Fast algorithms for team member recommendation. *arXiv:1409.5512*, 2014.

[23] 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.

[24] N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. *NIPS*, 2009.

[25] 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.

[26] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

[27] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*, pages 687–696, 2007.

[28] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. *SDM*, 1(1):6–22, 2008.

[29] 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.

[30] H. Tong, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *SDM*, pages 704–715, 2008.

[31] E. Tsivtsivadze, J. Urban, H. Geuvers, and T. Heskes. Semantic graph kernels for automated reasoning. In *SDM*, pages 795–803, 2011.

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

[33] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *The Journal of Machine Learning Research*, 99:1201–1242, 2010.

[34] F. Wang, H. Tong, and C. Lin. Towards evolutionary nonnegative matrix factorization. In *AAAI*, 2011.

[35] R. Zafarani, M. Ali Abbasi, and H. Liu. *Social Media Mining*. Cambridge University Press, 2014.

[36] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *WWW*, pages 791–800. ACM, 2009.