Fast and Accurate Random Walk with Restart on Dynamic Graphs with Guarantees

Minji Yoon
Seoul National University
riin55@snu.ac.kr

Woojeong Jin
Seoul National University
woojung211@snu.ac.kr

U Kang
Seoul National University
ukang@snu.ac.kr

ABSTRACT

Given a time-evolving graph, how can we track similarity between nodes in a fast and accurate way, with theoretical guarantees on the convergence and the error? Random Walk with Restart (RWR) is a popular measure to estimate the similarity between nodes and has been exploited in numerous applications. Many real-world graphs are dynamic with frequent insertion/deletion of edges; thus, tracking RWR scores on dynamic graphs in an efficient way has aroused much interest among data mining researchers. Recently, dynamic RWR models based on the propagation of scores across a given graph have been proposed, and have succeeded in outperforming previous other approaches to compute RWR dynamically. However, those models fail to guarantee exactness and convergence time for updating RWR in a generalized form.

In this paper, we propose OSP, a fast and accurate algorithm for computing dynamic RWR with insertion/deletion of nodes/edges in a directed/undirected graph. When the graph is updated, OSP first calculates offset scores around the modified edges, propagates the offset scores across the updated graph, and then merges them with the current RWR scores to get updated RWR scores. We prove the exactness of OSP and introduce OSP-T, a version of OSP which regulates a trade-off between accuracy and computation time by using error tolerance ϵ. Given restart probability c, OSP-T guarantees to return RWR scores with $O(\epsilon/c)$ error in $O(\log_{(1-c)}(\frac{\epsilon}{2}))$ iterations. Through extensive experiments, we show that OSP tracks RWR exactly up to 4605× faster than existing static RWR method on dynamic graphs, and OSP-T requires up to 15× less time with 730× lower $L1$ norm error and 3.3× lower rank error than other state-of-the-art dynamic RWR methods.

CCS CONCEPTS

• **Networks → Online social networks**;

KEYWORDS

Random Walk with Restart, Online algorithm

ACM Reference Format:
Minji Yoon, Woojeong Jin, and U Kang. 2018. Fast and Accurate Random Walk with Restart on Dynamic Graphs with Guarantees. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178876.33186107

WWW 2018, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5639-8/18/04..
https://doi.org/10.1145/3178876.33186107

1 INTRODUCTION

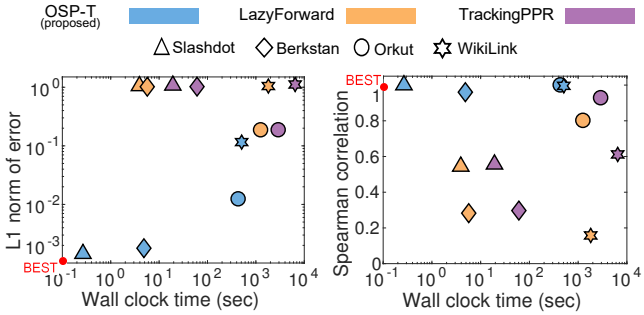
Identifying similarity score between two nodes in a graph has been recognized as a fundamental tool to analyze the graph [15, 21, 22, 25, 28] and has been exploited in various graph mining tasks [2, 5, 8, 11, 19]. Among numerous methods [9, 16, 18] to measure the similarity, random walk with restart (RWR) [18] has aroused considerable attention due to its ability to account for the global network structure from a particular user's point of view. To avoid expensive costs incurred by RWR computation, various methods have been proposed to calculate RWR scores efficiently, and the majority of them have focused on static graphs [12–14, 20, 24]. However, many real-world graphs are dynamic. For example, in an online social network of Facebook which has more than 1.3 billion users, 5 new users are added every second [17]. Thus it is an indispensable task to track RWR scores on time-evolving graphs.

Various approaches have been proposed to handle dynamic RWR problem efficiently. Chien et al. [6] introduced an approximate aggregation/disaggregation method which updates RWR scores only around modified edges. Bahmani et al. [4] applied the Monte-Carlo method [10] on the dynamic RWR problem. Recently, score propagation models were proposed by Ohsaka et al. [17] and Zhang et al. [27]; Ohsaka et al. proposed TrackingPPR which propagates scores using Gauss-Southwell algorithm; Zhang et al. proposed LazyForward which optimizes the initial step from TrackingPPR and propagates scores using Forward Push algorithm. They succeed in outperforming the previous approaches in both running time and accuracy [17, 27]. However, they fail to provide theoretical analysis of accuracy bound. Furthermore, they narrow down the scope of their analyses on time complexity to graph modifications only with insertion of edges or graph modifications on undirected graphs.

In this paper, we propose OSP (Offset Score Propagation for RWR), a fast and accurate method model for computing RWR scores on dynamic graphs. OSP is based on cumulative power iteration (CPI) [26] which interprets an RWR problem as propagation of scores from a seed node across a graph in an iterative matrix-vector multiplication form. When the graph is updated, OSP first calculates offset scores made around the updated edges, and then propagates the offset scores across the modified graph using CPI. The small size of the offset scores leads to fast convergence. Then OSP merges the result of the propagation with the current RWR scores to get an updated RWR scores. We also propose OSP-T, a version of OSP, with provable error bound and running time: given restart probability c and error tolerance ϵ, OSP-T computes RWR scores with $O(\epsilon/c)$ error in $O(\log_{(1-c)}(\frac{\epsilon}{2}))$ iterations in dynamic graphs. Through extensive experiments with various real-world graphs, we demonstrate the superior performance of OSP and OSP-T over existing methods. Table 1 and Figure 1 show a comparison of OSP,

Table 1: Comparison of our proposed OSP, OSP-T and existing methods for RWR computation on dynamic graphs. OSP computes dynamic RWR exactly with reasonable time, while OSP-T shows the best trade-off between speed and accuracy among approximate methods. OSP and OSP-T apply to the most general settings with guarantees.

Method	Speed	Accuracy	Coverage	Accuracy Bound	Time complexity model
TrackingPPR [17]	Fast	Low	Undirected graph	No	Only with insertion of edges
LazyForward [27]	Fast	Low	Undirected graph	No	Only with undirected graph
OSP	**Medium**	**High**	**Directed/Undirected graph**	**Yes**	**General**
OSP-T	**Faster**	**Medium**	**Directed/Undirected graph**	**Yes**	**General**



(a) Accuracy on L1 norm of error (b) Accuracy on Rank

Figure 1: Trade-off between accuracy and time: OSP-T shows the best trade-off between speed and accuracy among approximate methods for dynamic RWR.

OSP-T, and existing methods. The main contributions of this paper are the followings:

- **Algorithm.** We introduce OSP, a fast and accurate method to compute RWR on dynamic graphs. We also propose OSP-T, a version of OSP which regulates a trade-off between accuracy and computation time by using an error tolerance parameter.
- **Analysis.** We present a theoretical analysis on exactness of OSP, and time complexity and error bound of OSP-T. Our analysis is applicable to general dynamic graphs: insertion / deletion of nodes / edges in directed / undirected graphs.
- **Experiment.** We present extensive empirical evidences for the performance of OSP and OSP-T using various dynamic real-world graphs. We show that OSP tracks RWR exactly up to 4605× faster than existing static RWR method, and OSP-T requires up to 15× less time with 730× lower $L1$ norm error and 3.3× lower rank error than other dynamic RWR methods.

The code of our method and datasets used in the paper are available at http://datalab.snu.ac.kr/osp. The rest of the paper is organized as follows. In Section 2, we describe preliminaries on RWR and CPI. In Section 3, we present the proposed model OSP and OSP-T in detail along with theoretical analyses. After presenting our experimental results in Section 4, we provide a review on related works in Section 5 and conclude in Section 6. The symbols frequently used in this paper are summarized in Table 2.

2 PRELIMINARIES

In this section, we briefly review RWR [23], and explain CPI [26], an iterative method for RWR computation.

2.1 Random Walk with Restart

Random walk with restart (RWR) [23] estimates each node's relevance with regard to a given seed node s in a graph. RWR assumes

Table 2: Table of symbols.

Symbol	Definition
G	input graph
ΔG	update in graph
n, m	numbers of nodes and edges in G
s	seed node (= query node, source node)
c	restart probability
ϵ	error tolerance
$\mathbf{q}$	$(n \times 1)$ starting vector whose sth element is 1 and other elements are 0
$\tilde{\mathbf{A}}$	$(n \times n)$ row-normalized adjacency matrix of G
$\tilde{\mathbf{B}}$	$(n \times n)$ row-normalized adjacency matrix of $G + \Delta G$
$\Delta \mathbf{A}$	$(n \times n)$ difference between $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ $(= \tilde{\mathbf{B}} - \tilde{\mathbf{A}})$
$\mathbf{r}_{\text{old}}$	$(n \times 1)$ RWR vector on G
$\mathbf{r}_{\text{new}}$	$(n \times 1)$ updated RWR vector on $G + \Delta G$
$\mathbf{q}_{\text{offset}}$	$(n \times 1)$ offset seed vector
$\mathbf{x}^{(i)}_{\text{offset}}$	$(n \times 1)$ interim offset score vector at ith iteration in OSP
$\mathbf{r}_{\text{offset}}$	$(n \times 1)$ offset score vector

a random surfer who starts at node s. In each step, the surfer follows edges with probability $1 - c$, or jumps to the seed node with probability c. The surfer chooses an edge to move on with uniform probability among all current outgoing edges. The vector $\mathbf{r}$ representing each node's visiting probability satisfies the following equation:

$$\mathbf{r} = (1 - c)\tilde{\mathbf{A}}^\top \mathbf{r} + c\mathbf{q} \tag{1}$$

where $\tilde{\mathbf{A}}$ is a row-normalized adjacency matrix and $\mathbf{q}$ is a starting vector whose sth element is 1 and other elements are 0.

2.2 CPI: Cumulative Power Iteration

Cumulative power iteration (CPI) [26] interprets an RWR problem as a propagation of scores across a graph in an iterative matrix-vector multiplication form: at first, a score c is generated from a seed node; at each iteration, scores are divided and propagated evenly into out-edges with decaying coefficient $1-c$. $\mathbf{x}^{(i)}$ is an interim score vector computed from the iteration i and has scores propagated across nodes at ith iteration as entries. When multiplied with $(1 - c)\tilde{\mathbf{A}}^\top$, scores in $\mathbf{x}^{(i)}$ are propagated into their outgoing neighbors, and the propagated scores are stored in $\mathbf{x}^{(i+1)}$. CPI accumulates interim score vectors $\mathbf{x}^{(i)}$ to get the final RWR score vector $\mathbf{r}_{\text{CPI}}$ as follows.

$$\mathbf{x}^{(0)} = \mathbf{q}_{\text{CPI}}$$

$$\mathbf{x}^{(i)} = (1 - c)\tilde{\mathbf{A}}^\top \mathbf{x}^{(i-1)} = \left((1 - c)\tilde{\mathbf{A}}^\top\right)^i \mathbf{q}_{\text{CPI}}$$

$$\mathbf{r}_{\text{CPI}} = \sum_{i=0}^{\infty} \mathbf{x}^{(i)} = \sum_{i=0}^{\infty} \left((1 - c)\tilde{\mathbf{A}}^\top\right)^i \mathbf{q}_{\text{CPI}}$$

$\mathbf{q}_{\text{CPI}}$ is a seed vector which contains initial scores for propagation. To satisfy Equation 1, $\mathbf{q}_{\text{CPI}}$ needs to be set with $c\mathbf{q}$ where an initial score c is located at seed index s. In other words, setting the seed

vector $\mathbf{q}_{\text{CPI}}$ with other values leads to wrong RWR values in CPI. Unlike other propagation methods such as Gauss-Southwell algorithm [17] and Forward Local Push algorithm [27], CPI computes RWR with accuracy assurance and general time complexity model. Thus we propose dynamic RWR method OSP and OSP-T using CPI to provide theoretical guarantees on the error and the convergence.

3 PROPOSED METHOD

In this section, we describe our proposed method OSP for tracking RWR on dynamic graphs, and introduce OSP-T, a version of OSP which regulates a trade-off between accuracy and computation time on dynamic graphs.

3.1 OSP: Offset Score Propagation

In CPI, scores are propagated following underlying edges, and the score accumulated in each node becomes RWR score of the node in a given graph. In other words, RWR scores of nodes are determined by arrangement of edges. In this sense, when the graph G is updated with an insertion/deletion of edges (ΔG), propagation of scores around ΔG is changed: with insertion of an edge $e = (u, v)$, score x_u from the node u would be propagated into each out-edges with smaller scores $\frac{x_u}{d_u+1}$ than previous propagated scores $\frac{x_u}{d_u}$ where d_u is the number of out-edges of node u before ΔG; with deletion of an edge $e = (u, v)$, score x_u from the node u would be propagated into each out-edges with higher scores $\frac{x_u}{d_u-1}$. Then, these changes are propagated, affect the previous propagation pattern across the whole graph, and finally lead to a different RWR vector r_{new} of the updated graph $G + \Delta G$ from r_{old} of the original graph G. Based on this observation, OSP first calculates an offset seed vector $\mathbf{q}_{\text{offset}}$ and propagates the offset scores across the updated graph $G + \Delta G$ using CPI to get an offset score vector $\mathbf{r}_{\text{offset}}$. Finally, OSP adds up $\mathbf{r}_{\text{old}}$ and $\mathbf{r}_{\text{offset}}$ to get the final RWR score vector $\mathbf{r}_{\text{new}}$ as follows:

$$\mathbf{q}_{\text{offset}} \leftarrow (1-c)(\tilde{\mathbf{B}}^\top - \tilde{\mathbf{A}}^\top)\mathbf{r}_{\text{old}} = (1-c)(\Delta\mathbf{A})^\top \mathbf{r}_{\text{old}}$$
$$\mathbf{x}^{(i)}_{\text{offset}} \leftarrow ((1-c)\tilde{\mathbf{B}}^\top)^i \mathbf{q}_{\text{offset}}$$
$$\mathbf{r}_{\text{offset}} \leftarrow \sum_{i=0}^{\infty} \mathbf{x}^{(i)}_{\text{offset}} = \sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i \mathbf{q}_{\text{offset}}$$
$$\mathbf{r}_{\text{new}} \leftarrow \mathbf{r}_{\text{old}} + \mathbf{r}_{\text{offset}}$$

where $\tilde{\mathbf{A}}$ is a row-normalized adjacency matrix of G, $\tilde{\mathbf{B}}$ is a row-normalized adjacency matrix of $G + \Delta G$, and $\Delta\mathbf{A} = \tilde{\mathbf{B}} - \tilde{\mathbf{A}}$ is the difference between $\tilde{\mathbf{B}}$ and $\tilde{\mathbf{A}}$. Before proving the exactness of $\mathbf{r}_{\text{new}}$ computed by OSP, we first show the convergence of $\mathbf{r}_{\text{offset}}$ in Lemma 3.1.

Lemma 3.1 (Convergence of $\mathbf{r}_{\text{offset}}$). *$\mathbf{r}_{offset}$ converges to a constant value.*

Proof. $\mathbf{r}_{\text{offset}}$ is represented in OSP as follows:

$$\begin{aligned}\mathbf{r}_{\text{offset}} &= \sum_{i=0}^{\infty} \mathbf{x}^{(i)}_{\text{offset}} = \sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i \mathbf{q}_{\text{offset}} \\ &= \sum_{i=0}^{\infty}(1-c)^i(\tilde{\mathbf{B}}^\top)^i(1-c)(\tilde{\mathbf{B}}^\top - \tilde{\mathbf{A}}^\top)\mathbf{r}_{\text{old}} \\ &= \sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^{(i+1)}\mathbf{r}_{\text{old}} - \sum_{i=0}^{\infty}(1-c)^{(i+1)}(\tilde{\mathbf{B}}^\top)^i\tilde{\mathbf{A}}^\top\mathbf{r}_{\text{old}}\end{aligned}$$

Note that $\|\mathbf{r}_{\text{old}}\|_1 = \|\tilde{\mathbf{A}}^\top\|_1 = \|\tilde{\mathbf{B}}^\top\|_1 = 1$ since $\mathbf{r}_{\text{old}}$ is an RWR score vector, and $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ are row-normalized stochastic matrices. Then $\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^{(i+1)}r_{\text{old}}$ and $\sum_{i=0}^{\infty}(1-c)^{(i+1)}(\tilde{\mathbf{B}}^\top)^i\tilde{\mathbf{A}}^\top\mathbf{r}_{\text{old}}$ converge, and thus, $\mathbf{r}_{\text{offset}}$ also converges to a constant value. □

In Theorem 3.2, we show that the sum of $\mathbf{r}_{\text{old}}$ and $\mathbf{r}_{\text{offset}}$ becomes the exact RWR score vector of the updated graph $G+\Delta G$. Our result is the first exactness guarantee for propagation approaches [17, 27] on dynamic graphs.

Theorem 3.2 (Exactness of OSP). *$\mathbf{r}_{new}$ computed by OSP is the exact RWR score vector of the updated graph $G + \Delta G$.*

Proof. For brevity, let $\bar{\mathbf{A}} \leftarrow (1-c)\tilde{\mathbf{A}}, \bar{\mathbf{B}} \leftarrow (1-c)\tilde{\mathbf{B}}, \bar{\mathbf{q}} \leftarrow c\mathbf{q}$ during this proof. Thus, the spectral radii of $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ become less than 1 during this proof.

$$\begin{aligned}\mathbf{r}_{\text{offset}} &= \sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^i\mathbf{q}_{\text{offset}} \\ &= \sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^i(\bar{\mathbf{B}}^\top - \bar{\mathbf{A}}^\top)\mathbf{r}_{\text{old}} \\ &= \sum_{i=0}^{\infty}\left((\bar{\mathbf{B}}^\top)^i(\bar{\mathbf{B}}^\top - \bar{\mathbf{A}}^\top)\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}}\right) \\ &= \sum_{i=0}^{\infty}\left((\bar{\mathbf{B}}^\top)^{i+1}\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}} - (\bar{\mathbf{B}}^\top)^i\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^{k+1}\bar{\mathbf{q}}\right)\end{aligned}$$

Note that $\mathbf{r}_{\text{old}}$ is represented as $\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}}$ in CPI.

The third summation $\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^{k+1}\bar{\mathbf{q}}$ in the last equation is expressed as follows:

$$\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^{k+1}\bar{\mathbf{q}} = (\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}}) - \bar{\mathbf{q}}$$

Using this equation, $\mathbf{r}_{\text{offset}}$ is

$$\begin{aligned}\mathbf{r}_{\text{offset}} &= \sum_{i=0}^{\infty}\left((\bar{\mathbf{B}}^\top)^{i+1}\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}} - (\bar{\mathbf{B}}^\top)^i\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}} + (\bar{\mathbf{B}}^\top)^i\bar{\mathbf{q}}\right) \\ &= \sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^{i+1}\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}} - \sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^i\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}} + \sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^i\bar{\mathbf{q}}\end{aligned}$$

The first term of the last equation, $\sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^{i+1}\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}}$ is expressed as follows:

$$\sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^{i+1}\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}} = \sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^i\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}} - \sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}}$$

Then $\mathbf{r}_{\text{offset}}$ is expressed as follows:

$$\mathbf{r}_{\text{offset}} = -\sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}} + \sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^i\bar{\mathbf{q}}$$

Note $\mathbf{r}_{\text{old}} = \sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}}$ in CPI. Then $\mathbf{r}_{\text{new}}$ becomes as follows:

$$\begin{aligned}\mathbf{r}_{\text{new}} &= \mathbf{r}_{\text{old}} + \mathbf{r}_{\text{offset}} \\ &= \sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}} - \sum_{k=0}^{\infty}(\bar{\mathbf{A}}^\top)^k\bar{\mathbf{q}} + \sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^i\bar{\mathbf{q}} \\ &= \sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^i\bar{\mathbf{q}}\end{aligned}$$

Note that RWR score vector of the updated graph $G + \Delta G$ is expressed as $\sum_{i=0}^{\infty}(\bar{\mathbf{B}}^\top)^i\bar{\mathbf{q}}$ in CPI. □

Algorithm 1: OSP and OSP-T Algorithm

Require: previous RWR score vector: $\mathbf{r}_{\text{old}}$, row-normalized adjacency matrix: $\tilde{\mathbf{A}}$, update in $\tilde{\mathbf{A}}$: $\Delta\mathbf{A}$, restart probability: c, error tolerance: ϵ
Ensure: updated RWR score vector: $\mathbf{r}_{\text{new}}$
1: set seed offset vector $\mathbf{q}_{\text{offset}} = (1-c)(\Delta\mathbf{A})^\top \mathbf{r}_{\text{old}}$
2: set $\mathbf{r}_{\text{offset}} = \mathbf{0}$ and $\mathbf{x}^{(0)}_{\text{offset}} = \mathbf{q}_{\text{offset}}$
3: **for** iteration $i = 1$; $\|\mathbf{x}^{(i)}_{\text{offset}}\|_1 > \epsilon$; i++ **do**
4: compute $\mathbf{x}^{(i)}_{\text{offset}} \leftarrow (1-c)(\tilde{\mathbf{A}} + \Delta\mathbf{A})^\top \mathbf{x}^{(i-1)}_{\text{offset}}$
5: compute $\mathbf{r}_{\text{offset}} \leftarrow \mathbf{r}_{\text{offset}} + \mathbf{x}^{(i)}_{\text{offset}}$
6: **end for**
7: $\mathbf{r}_{\text{new}} \leftarrow \mathbf{r}_{\text{old}} + \mathbf{r}_{\text{offset}}$
8: **return** $\mathbf{r}_{\text{new}}$

Algorithm 1 describes how OSP works. OSP first calculates a seed offset vector $\mathbf{q}_{\text{offset}}$ (line 1). Then OSP initializes RWR score vector $\mathbf{r}_{\text{offset}}$ and $\mathbf{x}^{(0)}_{\text{offset}}$ using the offset vector $\mathbf{q}_{\text{offset}}$ (line 2). In ith iteration, scores in $\mathbf{x}^{(i-1)}_{\text{offset}}$ from the previous iteration $(i-1)$ are propagated through $\tilde{\mathbf{A}} + \Delta\mathbf{A}$ with decaying coefficient $1-c$ (line 4). Then, interim score vector $\mathbf{x}^{(i)}_{\text{offset}}$ is accumulated in $\mathbf{r}_{\text{offset}}$ (line 5). OSP stops when $\|\mathbf{x}^{(i)}_{\text{offset}}\|_1 \le \epsilon$ which is a condition for the final score vector $\mathbf{r}_{\text{offset}}$ to converge (line 3). Finally, OSP sums up $\mathbf{r}_{\text{old}}$ and $\mathbf{r}_{\text{offset}}$ (line 7). To retrieve exact RWR scores, OSP sets error tolerance ϵ to a very small value such as 10^{-9}. Using higher values for ϵ, we propose an approximate method OSP-T which trades off the accuracy against the running time for users who put more priority on speed than accuracy in the following section.

3.2 OSP-T: OSP with Tradeoff

OSP-T is an approximate method for dynamic RWR computation which is based on OSP. As described in Algorithm 1, the algorithm of OSP-T is the same as OSP, but OSP-T regulates its accuracy and speed using error tolerance parameter ϵ. In the following, we analyze how much OSP-T sacrifices its accuracy and increases its speed when an error tolerance ϵ is given.

Theorem 3.3 (Time Complexity of OSP-T). *With error tolerance ϵ, OSP-T takes $O(m \log_{(1-c)}(\frac{\epsilon}{2}))$ where m is the number of nonzeros in $\tilde{\mathbf{A}} + \Delta\mathbf{A}$.*

Proof. $\tilde{\mathbf{B}}$ denotes $\tilde{\mathbf{A}} + \Delta\mathbf{A}$, the row-normalized matrix for the updated graph. In each iteration, OSP-T computes $\mathbf{x}^{(i)}_{\text{offset}} = (1-c)\tilde{\mathbf{B}}^\top \mathbf{x}^{(i-1)}_{\text{offset}}$, and takes $O(m)$ time where m is the number of nonzeros in $\tilde{\mathbf{B}}^\top$. It also means the upper bound of number of edges visited in each iteration. OSP-T stops the iteration with error tolerance ϵ when $\|\mathbf{x}^{(i)}_{\text{offset}}\|_1 = \|((1-c)\tilde{\mathbf{B}}^\top)^i \mathbf{q}_{\text{offset}}\|_1 = (1-c)^i \|\mathbf{q}_{\text{offset}}\|_1 \le \epsilon$. Note that $\tilde{\mathbf{B}}^\top$ is a column stochastic matrix and $\|\tilde{\mathbf{B}}^\top\|_1 = 1$. Then the number of iterations to be converged is $\log_{(1-c)}(\frac{\epsilon}{\|\mathbf{q}_{\text{offset}}\|_1})$ and total computation time becomes $O(m \log_{(1-c)}(\frac{\epsilon}{\|\mathbf{q}_{\text{offset}}\|_1}))$. The upper bound of $\|\mathbf{q}_{\text{offset}}\|_1$ is presented as follows:

$$\begin{aligned}\|\mathbf{q}_{\text{offset}}\|_1 &= \|(1-c)(\tilde{\mathbf{B}}^\top - \tilde{\mathbf{A}}^\top)\mathbf{r}_{\text{old}}\|_1 \\ &= (1-c)\|(\tilde{\mathbf{B}}^\top - \tilde{\mathbf{A}}^\top)\|_1 \\ &\le (1-c)(\|\tilde{\mathbf{B}}^\top\|_1 + \|\tilde{\mathbf{A}}^\top\|_1) = 2(1-c)\end{aligned}$$

where $\|\mathbf{r}_{\text{old}}\|_1 = \|\tilde{\mathbf{A}}^\top\|_1 = \|\tilde{\mathbf{B}}^\top\|_1 = 1$. Then upper bounds of number of iterations and time are as follows:

$$O(iteration) = \log_{(1-c)}(\frac{\epsilon}{2(1-c)}) < \log_{(1-c)}(\frac{\epsilon}{2})$$
$$O(time) = m \log_{(1-c)}(\frac{\epsilon}{2(1-c)}) < m \log_{(1-c)}(\frac{\epsilon}{2})$$

Note that the upper bound of $\|\mathbf{q}_{\text{offset}}\|_1$ determines the upper bound of time complexity: $\|\mathbf{q}_{\text{offset}}\|_1$ is a denominator in $\log_{(1-c)}(\frac{\epsilon}{\|\mathbf{q}_{\text{offset}}\|_1})$ and the base of the logarithm is $1-c$ which is smaller than 1. □

Fast convergence. From Theorem 3.3, OSP and OSP-T share the same upper bound $O(m)$ for the number of visited edges per iteration with their baseline method CPI [26]. In practice, OSP and OSP-T visit only small portion of edges since the starting vector $\mathbf{q}_{\text{offset}}$ of OSP and OSP-T has a small $L1$ length: $\mathbf{q}_{\text{offset}} = (1-c)(\Delta\mathbf{A})^\top \mathbf{r}_{\text{old}}$, and thus $\|\mathbf{q}_{\text{offset}}\|_1 \le (1-c)\|(\Delta\mathbf{A})^\top\|_1$ with a unit RWR score vector $\mathbf{r}_{\text{old}}$; then, with small update ΔG, $(\Delta\mathbf{A})^\top$ is a sparse matrix with small $L1$ length and leads to a small value of $\|\mathbf{q}_{\text{offset}}\|_1$. When $(\tilde{\mathbf{A}} + \Delta\mathbf{A})$ is multiplied with $\mathbf{q}_{\text{offset}}$, only small number of edges in $G + \Delta G$ would be visited. Table 3 shows the $L1$ length of the starting vector ($\|\mathbf{q}_{\text{offset}}\|_1$), the total number of iterations, and the total number of visited edges of CPI, OSP, and OSP-T varying the number of deleted edges from the LiveJournal dataset. The error tolerance ϵ is set to 10^{-9},10^{-9}, and 5×10^{-3} for CPI, OSP, and OSP-T, respectively. OSP and OSP-T have smaller size of the starting vector $\mathbf{q}_{\text{offset}}$ than CPI, resulting in fewer numbers of iterations and visited edges. Considering the total number of edges (m) of the LiveJournal dataset is 34, 681, 189, OSP and OSP-T visit only small portion of edges in the graph thus converge much faster than CPI does. OSP-T converges faster than OSP by trading off accuracy. As size of ΔG increases, numbers of iterations and visited edges, and $L1$ errors all increase. The reason is analyzed theoretically in Section 3.3.

According to Theorem 3.3, error tolerance ϵ determines the computation cost of OSP-T. With error tolerance ϵ and restart probability c, we show that error bound of OSP-T is $O(\frac{\epsilon}{c})$ in the following theorem.

Theorem 3.4 (Error bound of OSP-T). *When OSP-T converges under error tolerance ϵ, error bound of RWR score vector $\mathbf{r}_{new}$ computed by OSP-T is $O(\frac{\epsilon}{c})$.*

Proof. When OSP-T iterates until $(k-1)$th iteration, error bound is presented as follows:

$$\begin{aligned}O(error) &= \|\sum_{i=k}^{\infty} x^{(i)}_{\text{offset}}\|_1 = \|\sum_{i=k}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i \mathbf{q}_{\text{offset}}\|_1 \\ &= \|\sum_{i=k}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i (1-c)(\tilde{\mathbf{B}}^\top - \tilde{\mathbf{A}}^\top)\mathbf{r}_{\text{old}}\|_1 \\ &\le \sum_{i=k}^{\infty}(1-c)^{i+1}\|(\tilde{\mathbf{B}}^\top)^i\|_1 \|\tilde{\mathbf{B}}^\top - \tilde{\mathbf{A}}^\top\|_1 \|\mathbf{r}_{\text{old}}\|_1 \\ &\le \sum_{i=k}^{\infty} 2(1-c)^{i+1} = \frac{2}{c}(1-c)^{k+1}\end{aligned}$$

From the proof of Theorem 3.3, $k = \log_{(1-c)}(\frac{\epsilon}{2(1-c)})$. Then error is bounded as:

$$O(error) = \frac{2}{c}\frac{\epsilon}{2(1-c)}(1-c) = \frac{\epsilon}{c}$$

□

Table 3: Practical performance of OSP and OSP-T on the LiveJournal dataset. Even though CPI, OSP and OSP-T share the same theoretical upper bound $O(m)$ of the number of visited edges per iteration, they show the different numbers of iterations and visited edges due to the different starting vectors q and error tolerance ϵ. OSP converges faster than CPI with the help of smaller size of the starting vector, OSP-T ($\epsilon = 5 \times 10^{-3}$) converges faster than OSP ($\epsilon = 10^{-9}$) with the help of higher error tolerance. Note that the total number of edges of LiveJournal is 34, 681, 189.

#modified edges	CPI			OSP			OSP-T			
	$\|\mathbf{q}_{\text{CPI}}\|_1$	# iter	#visited edges($\times 10^3$)	$\|\mathbf{q}_{\text{offset}}\|_1$	# iter	#visited edges($\times 10^3$)	$\|\mathbf{q}_{\text{offset}}\|_1$	# iter	#visited edges($\times 10^3$)	L1 norm error
1	1	116	3, 910, 864	2.60×10^{-9}	2	2, 145	2.60×10^{-9}	1	25	2.84×10^{-8}
10	1	116	3, 910, 863	1.51×10^{-7}	14	405, 717	1.51×10^{-7}	1	147	3.42×10^{-7}
10^2	1	116	3, 910, 858	2.19×10^{-6}	26	839, 137	2.19×10^{-6}	1	788	1.77×10^{-6}
10^3	1	116	3, 910, 808	2.31×10^{-5}	35	1, 169, 546	2.31×10^{-5}	1	4, 098	1.64×10^{-5}
10^4	1	116	3, 910, 300	2.30×10^{-4}	47	1, 604, 965	2.30×10^{-4}	2	44, 960	1.11×10^{-4}
10^5	1	116	3, 905, 224	2.05×10^{-3}	61	2, 104, 446	2.05×10^{-3}	4	130, 470	7.51×10^{-4}

From Theorem 3.3 and Theorem 3.4, OSP-T trades off the running time and accuracy using ϵ. We show the effects of ϵ on the experimental performance of OSP-T in Section 4.4.

3.3 Effects of ΔG

Including our model, propagation-based methods [17, 27] for dynamic RWR computation have sporadically observed long running time which is considerably longer than the average in real-world graphs. Previous works [17] detect the fact but do not provide any further investigation on reasons. Based on OSP-T, we analyze the root cause of the long running time occasionally happened in the propagation models.

From the proof of Theorem 3.3, the running time of OSP-T is determined by the $L1$ length of its seed vector $\mathbf{q}_{\text{offset}}$. When $\mathbf{D}$ is a diagonal matrix where $\mathbf{D}_{ii} = \sum_{j=1}^{n} |\Delta\mathbf{A}_{ij}|$, $\Delta\tilde{\mathbf{A}} = \mathbf{D}^{-1}\Delta\mathbf{A}$ is a row-normalized matrix and $\Delta\tilde{\mathbf{A}}^\top$ is a column stochastic matrix. Then $\mathbf{q}_{\text{offset}}$ and its $L1$ length are presented as follows:

$$\begin{aligned}
\mathbf{q}_{\text{offset}} &= (1-c)(\Delta\mathbf{A})^\top \mathbf{r}_{\text{old}} \\
&= (1-c)(\mathbf{D}\mathbf{D}^{-1}\Delta\mathbf{A})^\top \mathbf{r}_{\text{old}} \\
&= (1-c)(\mathbf{D}\Delta\tilde{\mathbf{A}})^\top \mathbf{r}_{\text{old}} \\
&= (1-c)\Delta\tilde{\mathbf{A}}^\top(\mathbf{D}\mathbf{r}_{\text{old}}) \\
\|\mathbf{q}_{\text{offset}}\|_1 &= (1-c)\|\Delta\tilde{\mathbf{A}}^\top(\mathbf{D}\mathbf{r}_{\text{old}})\|_1 \\
&\leq (1-c)\|\mathbf{D}\mathbf{r}_{\text{old}}\|_1
\end{aligned}$$

Then $\|\mathbf{D}\mathbf{r}_{\text{old}}\|_1$ is a decisive factor for running time. When edges are inserted to node i or deleted from node i, ith row in $\tilde{\mathbf{B}}$ is updated from ith row in $\tilde{\mathbf{A}}$; then ith row in $\Delta\mathbf{A} = \tilde{\mathbf{B}} - \tilde{\mathbf{A}}$ has nonzero values; finally, (i, i)th element in $\mathbf{D}$ has a nonzero value. In summary, $\mathbf{D}$ is a sparse diagonal matrix which has nonzero values at (i, i)th element when node i is modified by a graph modification ΔG. Then, there are two main components in determining the value of $\|\mathbf{D}\mathbf{r}_{\text{old}}\|_1$: 1) how many nodes are modified (i.e. the number of nonzeros in $\mathbf{D}$), 2) which nodes are modified (i.e. the location of nonzeros in $\mathbf{D}$).

3.3.1 Size of ΔG. When there are many nodes affected by ΔG, many nonzero values are located in $\mathbf{D}$'s diagonal and multiplied with $\mathbf{r}_{\text{old}}$. This leads to a high value of $\|\mathbf{D}\mathbf{r}_{\text{old}}\|_1$. In other words, $\|\mathbf{D}\mathbf{r}_{\text{old}}\|_1$ is determined by the size of ΔG. Intuitively, when the scope of a graph modification gets larger, the computation time for updating RWR takes longer time. As shown in Table 3, as ΔG increases, $\|\mathbf{q}_{\text{offset}}\|_1$ increases in OSP and OSP-T, then total numbers of iterations and visited edges until convergence also increase. $L1$ error also increases since the error bound is also determined by $\|\mathbf{q}_{\text{offset}}\|_1$ as shown in the proof of Theorem 3.4. In Section 4.5.1, we show the effects of size of ΔG on performance of OSP-T in real-world graphs.

3.3.2 Location of ΔG. When the number of nonzeros is fixed, the location of nonzeros in $\mathbf{D}$ is a crucial factor for determining the value of $\|\mathbf{D}\mathbf{r}_{\text{old}}\|_1$. When nonzeros in $\mathbf{D}$ are multiplied with high scores in $\mathbf{r}_{\text{old}}$, the product becomes large. Otherwise, when nonzeros in $\mathbf{D}$ are multiplied with low scores in $\mathbf{r}_{\text{old}}$, the product becomes small. In other words, location of ΔG determines the running time of OSP-T. When ΔG appears around high RWR score nodes, running time skyrockets. On the other hand, when ΔG appears around low score nodes, OSP-T converges quickly. Note that most real-world graphs follow power-law degree distribution [7] with few nodes having high RWR scores and majority of nodes having low scores. Thus ΔG is less likely to happen around high RWR score nodes. This leads to sporadic occurrence of long running time in propagation models. In Section 4.5.2, we show the effects of location of ΔG on running time of OSP-T in real-world graphs.

3.4 Discussions

In addition to edge insertion/deletion, OSP (and OSP-T) easily handles node insertion/deletion. We also show how OSP handles dead-end nodes efficiently.

3.4.1 Node insertion/deletion. OSP easily handles both node insertion and deletion. When a node is inserted with its edges, OSP adds one column and one row, respectively, to the previous $(n \times n)$ matrix $\tilde{\mathbf{A}}$ with all zero values. In $(n+1)$th row of the updated matrix $\tilde{\mathbf{B}}$, edge distribution of the new node is stored. Likewise, OSP adds one row to the $(n \times 1)$ vector $\mathbf{r}_{\text{old}}$ with a zero value and stores an RWR score of the new node in $(n+1)$th row of $\mathbf{r}_{\text{new}}$. On the other hand, when a node is deleted from a given graph, the corresponding row in $\tilde{\mathbf{B}}$ is simply set to all zero values to express the deletion. The remaining process is the same as that of edge insertion/deletion as described in Algorithm 1.

3.4.2 Dead-end. Dead-ends which do not have any out-edges cause scores to leak out. Without handling dead-ends, total sum of RWR scores across a given graph would be less than 1. One common

way [27] to tackle the leakage problem is inserting edges from dead-end nodes to a seed node. However, inserting edges for every dead-end node leads to explosive computation time as the given graph gets larger proportional to the number of dead-ends. Thus RWR experiments have been frequently conducted without handling dead-ends [17, 27]. In this section, we introduce an efficient dead-end handling for OSP which brings the same effect as inserting new edges: we just need to scale $L1$ length of an RWR vector, computed from OSP, to 1 at the end. We prove the exactness of this dead-end handling in Theorem 3.5.

Theorem 3.5 (Exactness of Dead-end handling). *When $\mathbf{r}_{temp}$ denotes a result score vector in OSP without any handling for dead-ends, scaling $\mathbf{r}_{temp}$ to $\frac{\mathbf{r}_{temp}}{\|\mathbf{r}_{temp}\|_1}$ results in an exact RWR score vector $\mathbf{r}_{final}$ which resolves a score leakage problem caused by dead-ends.*

Proof. As proved in Theorem 3.2, the resulting score vector $\mathbf{r}_{\text{temp}}$ computed by OSP is presented as $c\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q}$. Let d_{total_1} be the whole score handed over to the seed node from dead-ends after propagating the initial score c from the seed. Then d_{total_1} is propagated across the graph to fill the leaked scores. We do not need to perform long calculation to propagate d_{total_1} from the seed node: $\mathbf{r}_{\text{temp}} = c\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q}$, and thus scaling $\mathbf{r}_{\text{temp}}$ with (d_{total_1}/c) results in $d_{\text{total}_1}\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q}$ which is the result of propagating score d_{total_1} from the seed. At this time, the propagation of d_{total_1} is leaked out again from the dead-ends. Repeatedly, we collect the leaked scores $(d_{\text{total}_2}, d_{\text{total}_3}, \cdots)$ which are inserted into the dead-ends, and propagate them from the seed. However, we do not need to do long calculation to collect the leaked scores $d_{\text{total}_1}, d_{\text{total}_2}, \cdots$ and propagate them from the seed again and again, since we know that

$$\|\mathbf{r}_{\text{temp}}\|_1 = c\|\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q}\|_1$$
$$\|\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q}\|_1 = \frac{\|\mathbf{r}_{\text{temp}}\|_1}{c}$$

Then the final RWR vector $\mathbf{r}_{\text{final}}$ with dead-end handling is presented as follows:

$$\begin{aligned}\mathbf{r}_{\text{final}} &= c\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q} + d_{\text{total}_1}\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q} \\ &\quad + d_{\text{total}_2}\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q} + \cdots \\ &= (c + d_{\text{total}_1} + d_{\text{total}_2} + d_{\text{total}_3} + \cdots)\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q}\end{aligned}$$
$$\|\mathbf{r}_{\text{final}}\|_1 = (c + d_{\text{total}_1} + d_{\text{total}_2} + d_{\text{total}_3} + \cdots)\|\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q}\|_1 = 1$$

Then $(c + d_{\text{total}_1} + d_{\text{total}_2} + d_{\text{total}_3} + \cdots)$ is expressed as follows:

$$c + d_{\text{total}_1} + d_{\text{total}_2} + d_{\text{total}_3} + \cdots = 1/\|\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q}\|_1 = \frac{c}{\|\mathbf{r}_{\text{temp}}\|_1}$$

Table 4: Dataset statistics

Dataset	Nodes	Edges	Direction	Error tolerance (OSP-T)
WikiLink[1]	12,150,976	378,142,420	Directed	10^{-2}
Orkut[2]	3,072,441	117,185,083	Undirected	5×10^{-3}
LiveJournal[2]	3,997,962	34,681,189	Undirected	5×10^{-3}
Berkstan[2]	685,230	7,600,595	Directed	10^{-4}
DBLP[2]	317,080	1,049,866	Undirected	10^{-4}
Slashdot[2]	82,144	549,202	Directed	10^{-4}

[1] http://konect.uni-koblenz.de/networks/
[2] http://snap.stanford.edu/data/

Thus we get $\mathbf{r}_{\text{final}}$ by scaling $\mathbf{r}_{\text{temp}}$ with $1/\|\mathbf{r}_{\text{temp}}\|_1$ as follows:

$$\begin{aligned}\mathbf{r}_{\text{final}} &= (c + d_{\text{total}_1} + d_{\text{total}_2} + d_{\text{total}_3} + \cdots)\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q} \\ &= \frac{c}{\|\mathbf{r}_{\text{temp}}\|_1}\sum_{i=0}^{\infty}((1-c)\tilde{\mathbf{B}}^\top)^i\mathbf{q} \\ &= \frac{1}{\|\mathbf{r}_{\text{temp}}\|_1}\mathbf{r}_{\text{temp}}\end{aligned}$$

□

4 EXPERIMENTS

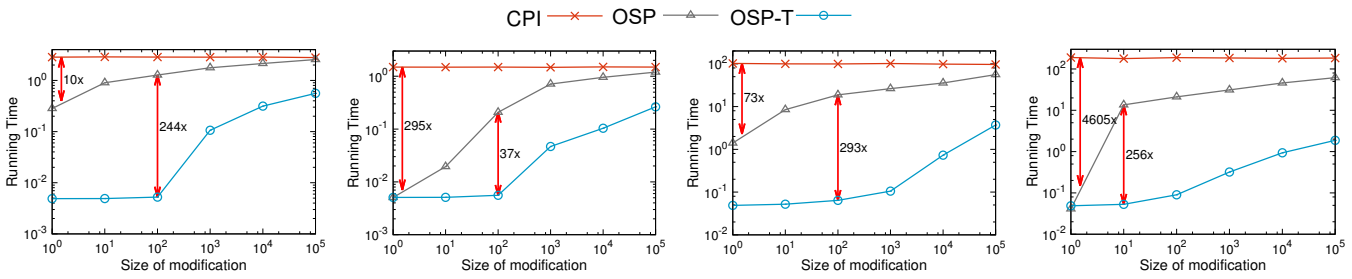
In this section, we experimentally evaluate the performance of OSP and OSP-T compared to other dynamic RWR methods. We aim to answer the following questions:

- **Q1 Performance of OSP.** How much does OSP improve performance for dynamic RWR computation from the baseline static method CPI? (Section 4.2)
- **Q2 Performance of OSP-T.** How much does OSP-T enhance computation efficiency, accuracy and scalability compared with its competitors? (Section 4.3)
- **Q3 Effects of ϵ, error tolerance.** How does the error tolerance ϵ affect the accuracy and the speed of OSP-T? (Section 4.4)
- **Q4 Effects of ΔG, a graph modification.** How does the size of ΔG affect the performance of OSP-T? (Section 4.5.1) How does the location of ΔG in the given graph G affect the performance of OSP-T? (Section 4.5.2)

4.1 Setup

4.1.1 Datasets. We use 6 real-world graphs to evaluate the effectiveness and efficiency of our methods. The datasets and their statistics are summarized in Table 4. Among them, Orkut, LiveJournal, and Slashdot are social networks, whereas DBLP is a collaboration network, and WikiLink and Berkstan are hyperlink networks.

4.1.2 Environment. All experiments are conducted on a workstation with a single core Intel(R) Xeon(R) CPU E5-2630 @ 2.2GHz and 512GB memory. We compare OSP with its baseline static method CPI, and compare OSP-T with two state-of-the-art approximate methods for dynamic RWR, TrackingPPR [17] and LazyForward [27], all of which are described in Section 5. All these methods including OSP and OSP-T are implemented in C++. We set the restart probability c to 0.15 as in the previous works [17, 26]. For each dataset, we measure the average value for 30 random seed nodes. CPI [26] is used to provide the exact RWR values in all experiments.



(a) DBLP (b) Berkstan (c) LiveJournal (d) Orkut

Figure 2: Performance of OSP and OSP-T: OSP computes the exact RWR faster than CPI on dynamic graphs. OSP-T achieves faster speed than OSP while sacrificing accuracy using higher error tolerance ϵ. ϵ is set to 10^{-9} for CPI and OSP, and set to higher values for OSP-T as described in Table 4. Experiments for accuracy of OSP-T are presented in Section 4.5.1.

4.2 Performance of OSP

We evaluate performance of OSP by measuring computation time for tracking RWR exactly on a dynamic graph G varying the size of ΔG. We set the initial graph G as a graph with all its edges and modify G by deleting edges. The size of ΔG varies from one edge to 10^5 edges. For space efficiency, we show the results on Orkut, LiveJournal, Berkstan, and DBLP; results on other graphs are similar. Error tolerance for CPI and OSP is set to 10^{-9} for all datasets, and error tolerance for OSP-T on each dataset is described in Table 4. From Figure 2, OSP tracks the exact RWR on dynamic graphs up to 4605× faster than CPI with the help of small size of the starting vector $\mathbf{q}_{\text{offset}}$. Note that CPI is initialized with $\mathbf{q}_{\text{CPI}} = c\mathbf{q}$. OSP-T trades off accuracy against speed using higher error tolerance ϵ, thus results in superior speed than CPI and OSP. Note that CPI and OSP compute the exact RWR scores while OSP-T results in the approximate RWR scores on dynamic graphs. As size of ΔG becomes larger, computation time of OSP and OSP-T increases while CPI maintains similar computation time. The effects of size of ΔG on computation time of OSP and OSP-T are discussed concretely in Section 4.5.1.

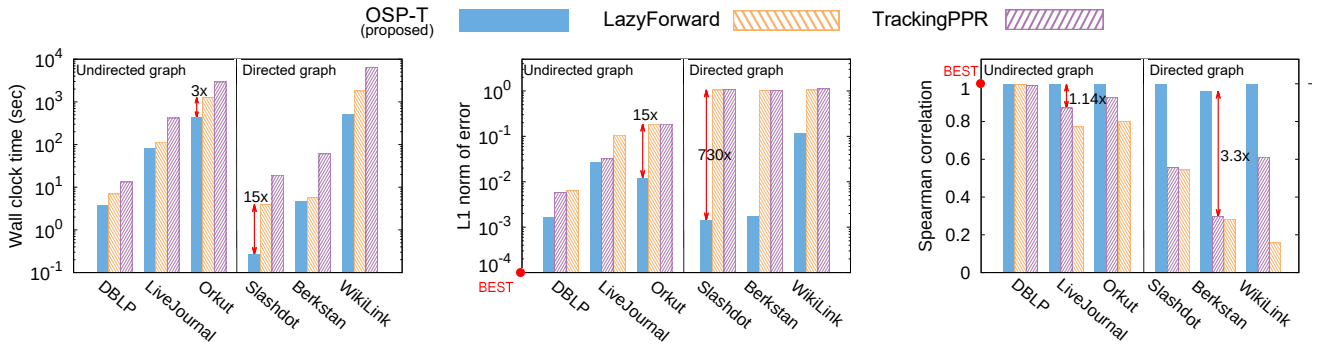
4.3 Performance of OSP-T

From each dataset, we generate a uniformly random edge stream and divide the stream into two parts. We extract 10 snapshots from the second part of the edge stream. At first, we initialize each graph with the first part of the stream, and then update the graph for each new snapshot arrival. At the end of the updates, we compare the performance of each algorithm.

4.3.1 Computational Efficiency. We evaluate the computational efficiency of OSP-T in terms of running time when error tolerance is given. Error tolerance used in TrackingPPR and LazyForward means maximum permissible $L1$ error per node, while error tolerance in OSP-T indicates maximum permissible $L1$ error per RWR score vector. For TrackingPPR and LazyForward, we set error tolerance to 10^{-8} across all datasets. For OSP-T, we set error tolerance close to $10^{-8} \times (\#nodes)$ for each graph, respectively, to give the same error tolerance effect with the competitors. Error tolerance ϵ for each dataset is described in Table 4. The wall-clock running time is shown in Figure 3(a). OSP-T runs faster than other methods by up to 15× while maintaining higher accuracy as shown in Figures 3(b) and 3(c). This performance difference comes from the different definitions of error tolerance which are described earlier. In OSP-T, error tolerance works per RWR score vector, and thus convergence condition is checked after all nodes that could be reached in one hop are updated. On the other hand, error tolerance works per node in TrackingPPR and LazyForward, and thus convergence condition is checked every time a node is updated.

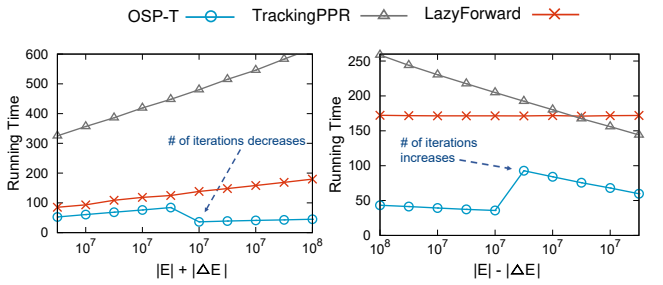
4.3.2 Accuracy. After all updates with snapshots, we get an approximate RWR vector for a given graph from each method. We compare $L1$ norm error between an approximate RWR vector and its exact RWR vector. To measures the rank accuracy, we use Spearman correlation [3]. Note that the higher the Spearman correlation, the higher is the rank accuracy. As shown in Figures 3(b) and 3(c), OSP-T outperforms other state-of-the-art methods with higher accuracy by up to 730× in $L1$ norm and 3.3× in ranking. This difference in accuracy comes from the different propagation models used in OSP-T and its competitors. Gauss-Southwell algorithm and Forward Local Push algorithm used in TrackingPPR and LazyForward, respectively, propagate scores toward the top-ranked node in terms of residual scores, while CPI used in OSP-T propagates scores toward the whole nodes which could be reached in one hop. Note that OSP-T maintains high accuracy on overall graphs, whereas LazyForward and TrackingPPR show different accuracy on directed and undirected graphs; since Gauss-Southwell algorithm and Forward Local Push algorithm are based on Local Push algorithm [1] which is designed for undirected graphs, they both show considerably lower accuracy on directed graphs than on undirected graphs.

4.3.3 Scalability. In this experiment, we estimate scalability of each method by comparing running time when a given graph incrementally grows/shrinks by inserting/deleting ΔG of a fixed size. For brevity, we show the result on Orkut; results on other graphs are similar. $|\Delta G|$ is fixed to 5×10^6 edges and the error tolerance for each method is the same as that in Section 4.3.1. When the graph incrementally grows, OSP-T shows different tendency compared to other methods. While all methods take a longer time as the graph grows, OSP-T occasionally shows a sudden drop in time as shown in Figure 4(a). Similarly, OSP-T shows a sudden jump when the given graph incrementally shrinks as shown in Figure 4(b). From the proof of Theorem 3.3, time complexity of OSP-T is $O(m \log_{(1-c)}(\frac{\epsilon}{\|\mathbf{q}_{\text{offset}}\|_1}))$. The first term m indicates the upper bound of number of edges which could be visited in each iteration. When a graph grows, the number of visited edges in each iteration would increase as shown in Figure 5(a). The second term



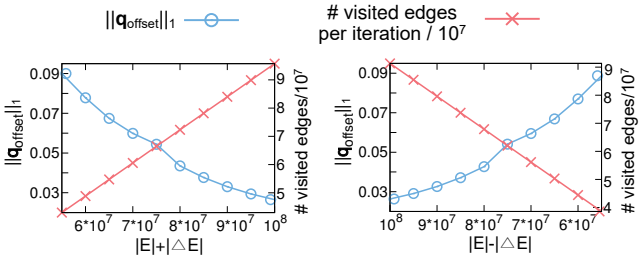
(a) Running time (b) Accuracy on L1 norm of error (c) Accuracy on Rank

Figure 3: Performance of OSP-T: (a) compares the running time among dynamic RWR methods; (b) and (c) compare the L1 norm of error and the rank accuracy of RWR scores of OSP-T and other methods with those of the exact RWR scores, respectively. While other methods show different performance on directed/undirected graphs, OSP-T maintains superior performance on overall graphs.



(a) Insertion on Orkut (b) Deletion on Orkut

Figure 4: Scalability: OSP-T shows the best scalability among the competitors.



(a) Insertion on Orkut (b) Deletion on Orkut

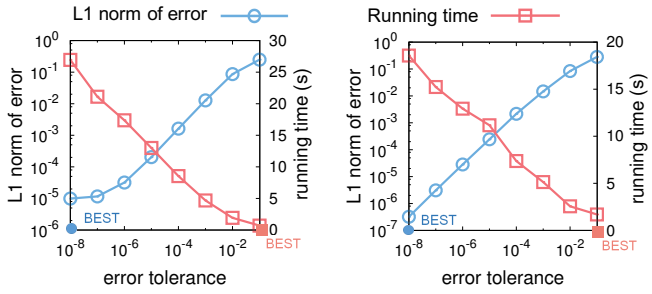
Figure 5: Two deciding factors for running time: $\|\mathbf{q}_{\text{offset}}\|_1$ and the number of visited edges per iteration show different tendencies when a graph grows/shrinks. Note that $|\Delta E|$ is fixed.

$\log_{(1-c)}(\frac{\epsilon}{\|\mathbf{q}_{\text{offset}}\|_1})$ indicates the number of iterations needed to converge, and is positively correlated with $\|\mathbf{q}_{\text{offset}}\|_1$. From Section 3.3, $\|\mathbf{q}_{\text{offset}}\|_1$ is decided by RWR scores of updated nodes: as RWR scores of updated nodes increase, $\|\mathbf{q}_{\text{offset}}\|_1$ increases. Assume that ΔG of a fixed size updates k nodes in the graph. As the graph grows, the average RWR score of the k nodes decreases since the total number of nodes in the graph increases while the total sum of RWR scores among nodes is always 1. As a result, as the graph grows, $\|\mathbf{q}_{\text{offset}}\|_1$ becomes smaller as shown in Figure 5(a). This leads to fewer iterations for convergence. In Figure 4(a), the number of iterations changes from 3 to 2 when $|E|+|\Delta E|$ is 8×10^7, thus the running time suddenly drops. When the number of iterations is consistent ($5.5\times10^7 < |E|+|\Delta E| < 7.5\times10^7$ and $8\times10^7 < |E|+|\Delta E| < 10^8$), the running time is decided by the first term m thus increasing constantly as the graph grows. In case of deletion, the opposite process is applied. As the graph shrinks, the average RWR score of updated nodes increases, thus $\|\mathbf{q}_{\text{offset}}\|_1$ becomes larger as shown in Figure 5(b). This leads to increased number of iterations for convergence. The number of iterations changes from 2 to 3 when $|E|-|\Delta E|$ is 7.5×10^7 with a sudden jump in running time. When the number of iterations is consistent, the running time decreases constantly as the graph shrinks. In LazyForward, time complexity of an undirected graph is $O(|\Delta E| + 1/(\bar{d}\delta))$ [27] where δ is the error tolerance per node and $\bar{d}$ is the average degree [1] of the graph which increases as the graph grows. In the time complexity model, running time would decrease as the graph grows, and increase as the graph shrinks. However, running time constantly increases when the graph grows in Figure 4(a) and maintains a constant value when the graph shrinks in Figure 4(b). Thus, the time complexity model fails to explain scalability of LazyForward. Since TrackingPPR [17] does not provide a time complexity model for general insertion/deletion, we could not investigate further. Only OSP-T succeeds in analyzing its scalability based on its time complexity model.

4.4 Effects of ϵ, error tolerance

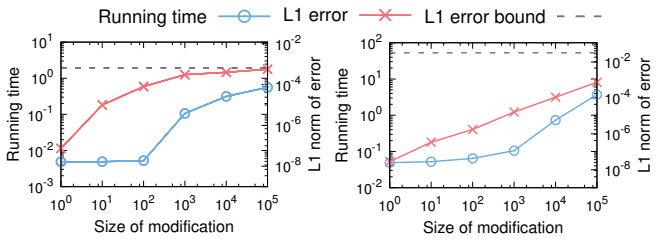
To examine the effects of error tolerance ϵ on the performance of OSP-T, we check the $L1$ error and running time varying ϵ. We report results on DBLP and Berkstan for brevity; results on other graphs are similar. As shown in Figure 6, as ϵ increases, the running time of OSP-T decreases and $L1$ error increases across all datasets. Note that we theoretically proved O(running time) is proportional to $log(\epsilon)$ in Theorem 3.3, and O($L1$ error) is proportional to ϵ in Theorem 3.4. We verify those theorems experimentally in this experiment. In Figure 6, error tolerance and $L1$ error are plotted in log scale while running time is plotted in linear scale. Relations among them are near linear. This shows that the theorems describing the relations among $L1$ error, running time and ϵ work in real-world datasets.

[1] In [27], the original time complexity is $O(|\Delta E|+1/\bar{\delta})$ where $\bar{\delta}$ is a degree-normalized error tolerance per node such that $|\mathbf{r}^{(i)}(t)|/d(t) < \bar{\delta}$ for residuals r with any node t at ith iteration. To consider the effect of changes in degree, we present δ as $\bar{d}\bar{\delta}$ with the average degree $\bar{d}$.



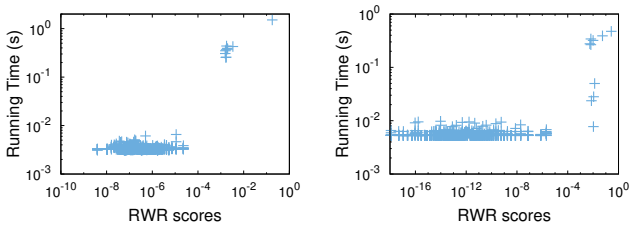
(a) DBLP (b) Berkstan

Figure 6: Effects of error tolerance ϵ on OSP-T: as ϵ increases, the $L1$ error increases while the running time decreases.



(a) DBLP (b) LiveJournal

Figure 7: Effects of size of ΔG on OSP-T: as the size of ΔG becomes bigger, both computation time and $L1$ error increase.



(a) DBLP (b) Berkstan

Figure 8: Effects of location of ΔG: when ΔG happens around high RWR score nodes, OSP-T takes long running time. However, this rarely happens since there are only few nodes with high RWR score in a graph.

4.5 Effects of ΔG, a graph modification

4.5.1 Size of ΔG. In Figure 7, as the size of ΔG increases, the running time and $L1$ error of OSP-T also increase. Larger size of ΔG leads to longer $L1$ length of the starting vector $\mathbf{q}_{\text{offset}}$ with longer computation time. Likewise, longer $L1$ length of $\mathbf{q}_{\text{offset}}$ leads to higher $L1$ error as discussed in Section 3.3. Still, the whole $L1$ norm errors are under the error bound $O(\epsilon/c)$ where ϵ is the error tolerance and c is the restart probability.

4.5.2 Location of ΔG. To show the effects of location of ΔG on the performance of OSP-T, we estimate the running time varying the location of an edge to be deleted in a given graph. We first calculate RWR scores among nodes and divide nodes evenly into 100 groups in the order of RWR scores. Then, we randomly sample 10 nodes from each group. For each sampled node u, we estimate the running time of OSP-T after deleting an edge (u, v). As shown in Figure 8, when a modification happens around nodes with high RWR scores, OSP-T takes a long running time, but this rarely happens. Intuitively, when a modification happens around high RWR score nodes, the higher offset scores are propagated, then the more steps would be needed to satisfy the given error tolerance. This intuition is consistent with the theoretical result we showed in Section 3.3. Sparse distribution around high RWR scores with long running time coincides with sporadic long running time observed by Ohsaka et al. [17].

5 RELATED WORKS

In this section, we review previous approaches to handle dynamic RWR problem. Chien et al. [6] proposed the approximate aggregation/disaggregation method. The method takes a small subset S that contains the updated edge, and then renew RWR scores only in S. Bahmani et al. [4] applied the Monte-Carlo method [10] on the dynamic RWR problem. Their algorithm maintains R random-walk segments, and reconstructs any segments related to a graph modification. Recently, score propagation models TrackingPPR and LazyForward were proposed by Ohsaka [17] and Zhang [27], respectively: when a given graph is updated, scores that complement the changes are calculated at first, then propagated across the graph. Although sharing the same intuition, they differ in the initialization step and the propagation method. While TrackingPPR propagates the scores immediately to all neighboring nodes, LazyForward modifies RWR values of the updated nodes at first then propagates the scores. In OSP-T, our proposed method, calculating offset seed vector $\mathbf{q}_{\text{offset}}$ is at the initialization step. The propagation methods used in the two models are Gauss-Southwell algorithm and Forward Local Push algorithm, respectively. In each iteration, Gauss-Southwell algorithm and Forward Local Push algorithm propagate scores stored in a vertex which has the largest score, while OSP-T propagates scores across whole vertices that could be reached in one hop. TrackingPPR and LazyForward succeed in outperforming the previous approaches [4, 6] in both computation time and accuracy [17, 27], and show the effectiveness of propagation model on dynamic graphs. However, none of them provide the guarantee of exactness or rigid analysis of error bound. Furthermore, Ohsaka et al. analyzed the running time only when edges are randomly and sequentially inserted, while Zhang et al. analyzed the running time only for undirected graphs. Note that we provide exactness of OSP, and time complexity and error bound for OSP-T in a generalized form. OSP-T outperforms TrackingPPR and LazyForward in terms of speed and accuracy as shown in our experiments (Section 4.3).

6 CONCLUSION

We propose OSP, a fast and accurate method for tracking RWR scores on a dynamic graph. When the graph is updated, OSP first calculates offset scores around the modified edges, propagates the offset scores across the updated graph, and then merges them with the current RWR scores to get updated RWR scores. We also propose OSP-T, a version of OSP which regulates a trade-off between accuracy and computation time. Among numerous dynamic RWR models based on score propagation, OSP is the first model with rigid analysis of accuracy and running time in a generalized form. Through intensive experiments, we show that OSP and OSP-T outperform other state-of-the-art methods in terms of accuracy and computation time.

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.R0190-15-2012, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development). U Kang is the corresponding author.

REFERENCES

[1] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*. IEEE, 475–486.

[2] I Antonellis, H Garcia-Molina, and C-C Simrank+ Chang. 2007. Query rewriting through link analysis of the click graph. *Proceedings of VLDB (Dec 2008)* (2007), 408–421.

[3] R Artusi, P Verderio, and E Marubini. 2002. Bravais-Pearson and Spearman correlation coefficients: meaning, test of hypothesis and confidence interval. *Int J Biol Markers* 17, 2 (2002), 148–151.

[4] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment* 4, 3 (2010), 173–184.

[5] Soumen Chakrabarti, Amit Pathak, and Manish Gupta. 2011. Index design and query processing for graph conductance search. *The VLDB Journal* 20, 3 (2011), 445–470.

[6] Steve Chien, Cynthia Dwork, Ravi Kumar, Daniel R Simon, and D Sivakumar. 2004. Link evolution: Analysis and algorithms. *Internet mathematics* 1, 3 (2004), 277–304.

[7] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. 1999. On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, Vol. 29. ACM, 251–262.

[8] Yasuhiro Fujiwara, Makoto Nakatsuji, Makoto Onizuka, and Masaru Kitsuregawa. 2012. Fast and exact top-k search for random walk with restart. *Proceedings of the VLDB Endowment* 5, 5 (2012), 442–453.

[9] Glen Jeh and Jennifer Widom. 2002. SimRank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 538–543.

[10] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*. ACM, 271–279.

[11] Woojeong Jin, Jinhong Jung, and U Kang. 2017. Supervised and Extended Restart in Random Walks for Ranking and Link Prediction in Networks. *arXiv preprint arXiv:1710.06609*.

[12] Jinhong Jung, Woojeong Jin, Lee Sael, and U. Kang. 2016. Personalized Ranking in Signed Networks Using Signed Random Walk with Restart. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*. 973–978.

[13] Jinhong Jung, Namyong Park, Lee Sael, and U Kang. 2017. BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart. In *SIGMOD*.

[14] Jinhong Jung, Kijung Shin, Lee Sael, and U. Kang. 2016. Random Walk with Restart on Large Graphs Using Block Elimination. *ACM Trans. Database Syst.* 41, 2 (2016), 12. https://doi.org/10.1145/2901736

[15] U Kang, Mikhail Bilenko, Dengyong Zhou, and Christos Faloutsos. 2012. Axiomatic Analysis of Co-occurrence Similarity Functions. *CMU-CS-12-102* (2012).

[16] Zhenjiang Lin, Michael R Lyu, and Irwin King. 2009. Matchsim: a novel neighbor-based similarity measure with maximum neighborhood matching. In *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 1613–1616.

[17] Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. 2015. Efficient PageRank tracking in evolving networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 875–884.

[18] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. 2004. Automatic multimedia cross-modal correlation discovery. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 653–658.

[19] Haekyu Park, Jinhong Jung, and U. Kang. 2017. A comparative study of matrix factorization and random walk with restart in recommender systems. In *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*. 756–765.

[20] Kijung Shin, Jinhong Jung, Sael Lee, and U Kang. 2015. BEAR: Block Elimination Approach for Random Walk with Restart on Large Graphs. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 1571–1585.

[21] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. 2005. Neighborhood formation and anomaly detection in bipartite graphs. In *Data Mining, Fifth IEEE International Conference on*. IEEE, 8–pp.

[22] Hanghang Tong and Christos Faloutsos. 2006. Center-piece subgraphs: problem definition and fast solutions. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 404–413.

[23] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. (2006).

[24] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: effective indexing for approximate personalized pagerank. *Proceedings of the VLDB Endowment* 10, 3 (2016), 205–216.

[25] Joyce Jiyoung Whang, David F Gleich, and Inderjit S Dhillon. 2013. Overlapping community detection using seed set expansion. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM, 2099–2108.

[26] Minji Yoon, Jinhong Jung, and U Kang. 2018. TPA: Fast, Scalable, and Accurate method for Approximate Random Walk with Restart on Billion Scale Graphs. In *IEEE 34th International Conference on Data Engineering, ICDE 2018, April 16-20, 2018, Paris, France*.

[27] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate Personalized PageRank on Dynamic Graphs. *arXiv preprint arXiv:1603.07796* (2016).

[28] Zeyuan Allen Zhu, Silvio Lattanzi, and Vahab S Mirrokni. 2013. A Local Algorithm for Finding Well-Connected Clusters.. In *ICML (3)*. 396–404.