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 \( \epsilon \). Given restart probability \( c \), OSP-T guarantees to return RWR scores with \( O(\epsilon/c) \) error in \( O(\log_{1-\epsilon}(\frac{1}{\epsilon})) \) iterations. Through extensive experiments, we show that OSP tracks RWR exactly up to 4605\times faster than existing static RWR method on dynamic graphs, and OSP-T requires up to 15\times less time with 730\times lower \( L1 \) norm error and 3.3\times 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

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 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 \( \epsilon \), OSP-T computes RWR scores with \( O(\epsilon/c) \) error in \( O(\log_{1-\epsilon}(\frac{1}{\epsilon})) \) 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Speed</th>
<th>Accuracy</th>
<th>Coverage</th>
<th>Accuracy Bound</th>
<th>Time complexity model</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrackingPPR</td>
<td>Fast</td>
<td>Low</td>
<td>Undirected graph</td>
<td>No</td>
<td>Only with insertion of edges</td>
</tr>
<tr>
<td>LazyForward</td>
<td>Fast</td>
<td>Low</td>
<td>Undirected graph</td>
<td>No</td>
<td>Only with undirected graph</td>
</tr>
<tr>
<td>OSP</td>
<td>Medium</td>
<td>High</td>
<td>Directed/Undirected graph</td>
<td>Yes</td>
<td>General</td>
</tr>
<tr>
<td>OSP-T</td>
<td>Faster</td>
<td>Medium</td>
<td>Directed/Undirected graph</td>
<td>Yes</td>
<td>General</td>
</tr>
</tbody>
</table>

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 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 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 r representing each node’s visiting probability satisfies the following equation:

\[ r = (1-c)\tilde{A}^\top r + cq \]  

where \( \tilde{A} \) is a row-normalized adjacency matrix and 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. \( x^{(i)} \) is an interim score vector computed from the iteration i and has scores propagated across nodes at ith iteration as entries. When multiplied with \((1-c)\tilde{A}^\top\), scores in \( x^{(i)} \) are propagated into their outgoing neighbors, and the propagated scores are stored in \( x^{(i+1)} \). CPI accumulates interim score vectors \( x^{(i)} \) to get the final RWR score vector \( r_{\text{CPI}} \) as follows.

\[ x^{(0)} = q_{\text{CPI}} \]
\[ x^{(i)} = (1-c)\tilde{A}^\top x^{(i-1)} = ((1-c)\tilde{A}^\top)^t q_{\text{CPI}} \]
\[ r_{\text{CPI}} = \sum_{t=0}^{\infty} x^{(t)} = \sum_{t=0}^{\infty} ((1-c)\tilde{A}^\top)^t q_{\text{CPI}} \]

\( q_{\text{CPI}} \) is a seed vector which contains initial scores for propagation. To satisfy Equation 1, \( q_{\text{CPI}} \) needs to be set with cq where an initial score c is located at seed index s. In other words, setting the seed vector \( q_{\text{CPI}} \) with other values leads to wrong RWR values in CPI.

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.

Table 2: Table of symbols.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>input graph</td>
</tr>
<tr>
<td>( \Delta G )</td>
<td>update in graph</td>
</tr>
<tr>
<td>( n, m )</td>
<td>numbers of nodes and edges in G</td>
</tr>
<tr>
<td>( s )</td>
<td>seed node (= query node, source node)</td>
</tr>
<tr>
<td>( c )</td>
<td>restart probability</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>error tolerance</td>
</tr>
<tr>
<td>( q )</td>
<td>((n \times 1)) starting vector whose sth element is 1 and other elements are 0</td>
</tr>
<tr>
<td>( \tilde{A} )</td>
<td>((n \times n)) row-normalized adjacency matrix of G</td>
</tr>
<tr>
<td>( \tilde{B} )</td>
<td>((n \times n)) row-normalized adjacency matrix of G + ( \Delta G )</td>
</tr>
<tr>
<td>( \Delta A )</td>
<td>((n \times n)) difference between ( \tilde{A} ) and ( \tilde{B} = \tilde{B} - \tilde{A} )</td>
</tr>
<tr>
<td>( r_{\text{old}} )</td>
<td>((n \times 1)) RWR vector on G</td>
</tr>
<tr>
<td>( r_{\text{new}} )</td>
<td>((n \times 1)) updated RWR vector on G + ( \Delta G )</td>
</tr>
<tr>
<td>( q_{\text{offset}} )</td>
<td>((n \times 1)) offset seed vector</td>
</tr>
<tr>
<td>( x_{\text{offset}}^{(i)} )</td>
<td>((n \times 1)) interim offset score vector at ith iteration in OSP</td>
</tr>
<tr>
<td>( r_{\text{offset}} )</td>
<td>((n \times 1)) offset score vector</td>
</tr>
</tbody>
</table>
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 $x_u$ than previous propagated scores $\tilde{x}_u$, 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 $x_{u,v}$.

Then, these changes are propagated, affect the previous propagation pattern across the whole graph, and finally lead to a different RWR vector $r_{\text{new}}$ of the updated graph $G + \Delta G$ from $r_{\text{old}}$ of the original graph G. Based on this observation, OSP first calculates an offset seed vector $q_{\text{offset}}$ and propagates the offset scores across the updated graph $G + \Delta G$ using CPI to get an offset score vector $r_{\text{offset}}$. Finally, OSP adds up $r_{\text{old}}$ and $r_{\text{offset}}$ to get the final RWR score vector $r_{\text{new}}$ as follows:

$$q_{\text{offset}} = (1 - c)(\bar{B}^T \bar{A})r_{\text{old}} = (1 - c)(\Delta A)^T r_{\text{old}}$$

$$x_{\text{offset}}(i) = (1 - c)(\bar{B}^T \bar{A})q_{\text{offset}}$$

$$r_{\text{offset}} = \sum_{i=0}^{\infty} x_{\text{offset}}(i) = \sum_{i=0}^{\infty} (1 - c)(\bar{B}^T \bar{A})^i q_{\text{offset}}$$

$$r_{\text{new}} = r_{\text{old}} + r_{\text{offset}}$$

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

Lemma 3.1 (Convergence of $r_{\text{offset}}$). $r_{\text{offset}}$ converges to a constant value.

Proof. $r_{\text{offset}}$ is represented in OSP as follows:

$$r_{\text{offset}} = \sum_{i=0}^{\infty} x_{\text{offset}}(i) = \sum_{i=0}^{\infty} (1 - c)(\bar{B}^T \bar{A})^i q_{\text{offset}}$$

$$= \sum_{i=0}^{\infty} (1 - c)^i(\bar{B}^T \bar{A})^i(1 - c)(\bar{B}^T \bar{A})r_{\text{old}}$$

$$= \sum_{i=0}^{\infty} (1 - c)^i(\bar{B}^T \bar{A})^i r_{\text{old}} - \sum_{i=0}^{\infty} (1 - c)^{i+1}(\bar{B}^T \bar{A})^i \bar{A}^T r_{\text{old}}$$

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

In Theorem 3.2, we show that the sum of $r_{\text{old}}$ and $r_{\text{offset}}$ 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). $r_{\text{new}}$ computed by OSP is the exact RWR score vector of the updated graph $G + \Delta G$.

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

$$r_{\text{offset}} = \sum_{i=0}^{\infty} (\tilde{B}^T)^i q_{\text{offset}}$$

$$= \sum_{i=0}^{\infty} (\tilde{B}^T)^i(\tilde{B}^T - \bar{A})^i r_{\text{old}}$$

$$= \sum_{i=0}^{\infty} (\tilde{B}^T)^i(\tilde{B}^T - \bar{A})^i \sum_{k=0}^{\infty} (\bar{A}^k)^q$$

$$= \sum_{i=0}^{\infty} (\tilde{B}^T)^i(\tilde{B}^T)^i \sum_{k=0}^{\infty} (\bar{A}^k)^q$$

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

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

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

Using this equation, $r_{\text{offset}}$ is:

$$r_{\text{offset}} = \sum_{i=0}^{\infty} (\tilde{B}^T)^i(\tilde{B}^T)^i \sum_{k=0}^{\infty} (\bar{A}^k)^q - \sum_{i=0}^{\infty} (\tilde{B}^T)^i(\tilde{B}^T)^i q + \sum_{i=0}^{\infty} (\tilde{B}^T)^i q$$

$$= \sum_{i=0}^{\infty} (\tilde{B}^T)^i(\tilde{B}^T)^i q + \sum_{i=0}^{\infty} (\tilde{B}^T)^i q$$

The first term of the last equation, $\sum_{i=0}^{\infty} (\tilde{B}^T)^i(\tilde{B}^T)^i q + \sum_{i=0}^{\infty} (\tilde{B}^T)^i q$ is expressed as follows:

$$\sum_{i=0}^{\infty} (\tilde{B}^T)^i(\tilde{B}^T)^i q = \sum_{i=0}^{\infty} (\tilde{B}^T)^i q$$

Then $r_{\text{offset}}$ is expressed as follows:

$$r_{\text{offset}} = -\sum_{k=0}^{\infty} (\bar{A}^k)^q q + \sum_{i=0}^{\infty} (\tilde{B}^T)^i q$$

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

$$r_{\text{new}} = r_{\text{old}} + r_{\text{offset}}$$

$$= \sum_{i=0}^{\infty} (\bar{A}^k)^q q + \sum_{i=0}^{\infty} (\tilde{B}^T)^i q$$

$$= \sum_{i=0}^{\infty} (\bar{A}^k)^q q$$

Note that RWR score vector of the updated graph $G + \Delta G$ is expressed as $\sum_{i=0}^{\infty} (\tilde{B}^T)^i q$ in CPI. □
Algorithm 1: OSP and OSP-T Algorithm

Require: previous RWR score vector: $r_{\text{old}}$, row-normalized adjacency matrix: $\tilde{A}$, update in $\tilde{A} + \Delta A$, restart probability: $c$, error tolerance: $\epsilon$

Ensure: updated RWR score vector: $r_{\text{new}}$

1. set seed offset vector $q_{\text{offset}} = (1 - c)(\Delta A)^T r_{\text{old}}$
2. set $r_{\text{offset}} = 0$ and $x_{\text{offset}} = q_{\text{offset}}$
3. for iteration $i = 1; \|x_{\text{offset}}(i)\| > \epsilon; i++$
4. compute $x_{\text{offset}}(i) \leftarrow (1 - c)(\tilde{A} + \Delta A)^T x_{\text{offset}}(i)$
5. compute $r_{\text{offset}} \leftarrow r_{\text{offset}} + x_{\text{offset}}(i)$
6. end for
7. $r_{\text{new}} \leftarrow r_{\text{old}} + r_{\text{offset}}$
8. return $r_{\text{new}}$

Algorithm 1 describes how OSP works. OSP first calculates a seed offset vector $q_{\text{offset}}$ (line 1). Then OSP initializes RWR score vector $r_{\text{offset}}$ and $x_{\text{offset}}(0)$ using the offset vector $q_{\text{offset}}$ (line 2). In ith iteration, scores in $x_{\text{offset}}(i-1)$ from the previous offset (i - 1) are propagated through $\tilde{A} + \Delta A$ with decay coefficient $1 - c$ (line 4). Then, interim score vector $x_{\text{offset}}(i)$ is accumulated in $r_{\text{offset}}$ (line 5). OSP stops when $\|x_{\text{offset}}(i-1)\| \leq \epsilon$ which is a condition for the final score vector $r_{\text{offset}}$ to converge (line 3). Finally, OSP sums up $r_{\text{old}}$ and $r_{\text{offset}}$ (line 7). To retrieve exact RWR scores, OSP sets error tolerance $\epsilon$ to a very small value such as $10^{-9}$. Using higher values for $\epsilon$, 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 $\epsilon$. In the following, we analyze how much OSP-T sacrifices its accuracy and increases its speed when an error tolerance $\epsilon$ is given.

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

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

$$\|q_{\text{offset}}\| = \|(1 - c)(\tilde{B}^T - \tilde{A})\| r_{\text{old}}\|$$

$$\| = (1 - c)\|((\tilde{B}^T - \tilde{A})\|)\|1 \leq (1 - c)\|\tilde{B}^T\|_1 + \|\tilde{A}\|_1 = 2\|1 - c\|$$

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

$$O(\text{iteration}) = \log (1 - c) (\frac{\epsilon}{2(1 - c)}) < \log (1 - c) (\frac{\epsilon}{2})$$

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

Note that the upper bound of $\|q_{\text{offset}}\|$ determines the upper bound of time complexity: $\|q_{\text{offset}}\|$ is a denominator in $\log (1-c) (\frac{\epsilon}{\|q_{\text{offset}}\|})$ 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 $q_{\text{offset}}$ of OSP and OSP-T has a small $L1$ length: $\|q_{\text{offset}}\| = (1 - c)(\Delta A)^T r_{\text{old}}$, and thus $\|q_{\text{offset}}\| \leq (1 - c)\|\Delta A\|_1$ with a unit RWR score vector $r_{\text{old}}$: then, with small update $\Delta G$, $(\Delta A)^T$ is a sparse matrix with small $L1$ length and leads to a small value of $\|q_{\text{offset}}\|$. When $\tilde{A} + \Delta A$ is multiplied with $q_{\text{offset}}$, only small number of edges in $G + \Delta G$ would be visited. Table 3 shows the $L1$ length of the starting vector $\|q_{\text{offset}}\|$, 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 $\epsilon$ is set to $10^{-1}, 10^{-3}$, and $5 \times 10^{-3}$ for CPI, OSP, and OSP-T, respectively. OSP and OSP-T have smaller size of the starting vector $q_{\text{offset}}$ 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 $\Delta G$ 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 $\epsilon$ determines the computation cost of OSP-T. With error tolerance $\epsilon$ 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 $\epsilon$, error bound of RWR score vector $r_{\text{new}}$ 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:

$$O(\text{error}) = \sum_{i=1}^{\infty} q_{\text{offset}}(i) q_{\text{offset}}(i) \|x_{\text{offset}}(i)\|_1$$

$$= \sum_{i=1}^{\infty} ((1 - c)\tilde{B})^i ((1 - c)(\tilde{B}^T - \tilde{A}) r_{\text{old}})$$

$$\leq \sum_{i=1}^{\infty} (1 - c)^i \|\tilde{B}^T\|_1 \|\tilde{B}^T - \tilde{A}\|_1 r_{\text{old}}\|_1$$

$$\leq \sum_{i=1}^{\infty} 2(1 - c)^i = \frac{2}{c} (1 - c)^{i+1}$$

From the proof of Theorem 3.3, $k = \log (1-c) (\frac{\epsilon}{\|q_{\text{offset}}\|})$. Then error is bounded as:

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

□
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 $\epsilon$. 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 ($\epsilon = 10^{-3}$) with the help of higher error tolerance. Note that the total number of edges of LiveJournal is 34, 681, 389.

<table>
<thead>
<tr>
<th>#modified edges</th>
<th>CPI</th>
<th># visited edges($10^3$)</th>
<th>OSP</th>
<th># visited edges($10^3$)</th>
<th>L1 norm error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>116</td>
<td>2.60 x $10^{-9}$</td>
<td>2</td>
<td>12.15</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>116</td>
<td>1.51 x $10^{-7}$</td>
<td>14</td>
<td>405,717</td>
</tr>
<tr>
<td>$10^2$</td>
<td>1</td>
<td>116</td>
<td>2.19 x $10^{-6}$</td>
<td>26</td>
<td>839,137</td>
</tr>
<tr>
<td>$10^3$</td>
<td>1</td>
<td>116</td>
<td>2.31 x $10^{-5}$</td>
<td>35</td>
<td>1,169,546</td>
</tr>
<tr>
<td>$10^4$</td>
<td>1</td>
<td>116</td>
<td>2.30 x $10^{-4}$</td>
<td>47</td>
<td>1,604,965</td>
</tr>
<tr>
<td>$10^5$</td>
<td>1</td>
<td>116</td>
<td>2.05 x $10^{-3}$</td>
<td>61</td>
<td>2,104,446</td>
</tr>
</tbody>
</table>

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

3.3 Effects of $\Delta G$

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 $q_{offset}$. When $D$ is a diagonal matrix where $D_{ii} = \sum_{j=1}^{n} |A_{ij}|$, $\Delta A = D^{-1}A$ is a row-normalized matrix and $\Delta A^{T}$ is a column stochastic matrix. Then $q_{offset}$ and its L1 length are presented as follows:

$$q_{offset} = (1 - \epsilon) (\Delta A)^T r_{old}$$
$$= (1 - \epsilon) (\Delta D^{-1} A)^T r_{old}$$
$$= (1 - \epsilon) (\Delta A)^T (D_{old})$$
$$\|q_{offset}\|_1 \leq (1 - \epsilon) \|D_{old}\|_1$$

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

3.3.1 Size of $\Delta G$. When there are many nodes affected by $\Delta G$, many nonzero values are located in $D$’s diagonal and multiplied with $r_{old}$. This leads to a high value of $\|D_{old}\|_1$. In other words, $\|D_{old}\|_1$ is determined by the size of $\Delta G$. 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 $\Delta G$ increases, $\|q_{offset}\|_1$ 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 $\|q_{offset}\|_1$ as shown in the proof of Theorem 3.4. In Section 4.5.1, we show the effects of size of $\Delta G$ on performance of OSP-T in real-world graphs.

3.3.2 Location of $\Delta G$. When the number of nonzeros is fixed, the location of nonzeros in $D$ is a crucial factor for determining the value of $\|D_{old}\|_1$. When nonzeros in $D$ are multiplied with high scores in $r_{old}$, the product becomes large. Otherwise, when nonzeros in $D$ are multiplied with low scores in $r_{old}$, the product becomes small. In other words, location of $\Delta G$ determines the running time of OSP-T. When $\Delta G$ appears around high RWR score nodes, running time skyrockets. On the other hand, when $\Delta G$ 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 $\Delta G$ 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 $\Delta G$ 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 $\tilde{A}$ with all zero values. In $(n + 1)$th row of the updated matrix $\tilde{B}$, edge distribution of the new node is stored. Likewise, OSP adds one row to the $(n + 1)$ vector $r_{old}$ with a zero value and stores an RWR score of the new node in $(n + 1)$th row of $r_{new}$. On the other hand, when a node is deleted from a given graph, the corresponding row in $\tilde{B}$ 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 \( r_{\text{temp}} \) denotes a result score vector in OSP without any handling for dead-ends, scaling \( r_{\text{temp}} \) to \( r_{\text{final}} = \frac{r_{\text{temp}}}{\|r_{\text{temp}}\|} \) results in an exact RWR score vector \( r_{\text{final}} \) which resolves a score leakage problem caused by dead-ends.

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

\[
\|r_{\text{temp}}\| = c \sum_{i=0}^{\infty} (1 - c)B^T)^i q \|_1
\]

\[
\|\sum_{i=0}^{\infty} ((1 - c)B^T)^i q\|_1 = \frac{\|r_{\text{temp}}\|}{c}
\]

Then the final RWR vector \( r_{\text{final}} \) with dead-end handling is presented as follows:

\[
r_{\text{final}} = c \sum_{i=0}^{\infty} ((1 - c)B^T)^i q + d_{\text{total}} \sum_{i=0}^{\infty} ((1 - c)B^T)^i q + \ldots
\]

\[
= (c + d_{\text{total}} + d_{\text{total}} + d_{\text{total}} + \ldots) \sum_{i=0}^{\infty} ((1 - c)B^T)^i q
\]

\[
\|r_{\text{final}}\|_1 = (c + d_{\text{total}} + d_{\text{total}} + d_{\text{total}} + \ldots) \sum_{i=0}^{\infty} ((1 - c)B^T)^i q = 1
\]

Then \( c + d_{\text{total}} + d_{\text{total}} + d_{\text{total}} + \ldots \) is expressed as follows:

\[
c + d_{\text{total}} + d_{\text{total}} + d_{\text{total}} + \ldots = 1/\|\sum_{i=0}^{\infty} ((1 - c)B^T)^i q\|_1
\]

Thus we get \( r_{\text{final}} \) by scaling \( r_{\text{temp}} \) with \( 1/\|r_{\text{temp}}\|_1 \) as follows:

\[
r_{\text{final}} = (c + d_{\text{total}} + d_{\text{total}} + \ldots) \sum_{i=0}^{\infty} ((1 - c)B^T)^i q
\]

\[
= \frac{c}{\|r_{\text{temp}}\|_1} \sum_{i=0}^{\infty} ((1 - c)B^T)^i q
\]

\[
= \frac{1}{\|r_{\text{temp}}\|_1} r_{\text{temp}} \]

\[\square\]

### 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 \( \epsilon \), error tolerance.** How does the error tolerance \( \epsilon \) affect the accuracy and the speed of OSP-T? (Section 4.4)

- **Q4 Effects of \( \Delta G \), a graph modification.** How does the size of \( \Delta G \) affect the performance of OSP-T? (Section 4.5.1) How does the location of \( \Delta G \) 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 \( \epsilon \) 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.

#### 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 \( \Delta G \). We set the initial graph \( G \) as a graph with all its edges and modify \( G \) by deleting edges. The size of \( \Delta G \) varies from one edge to \( 10^8 \) 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 $q_{offset}$. Note that CPI is initialized with $q_{offset} = c q$. OSP-T trades off accuracy against speed using higher error tolerance $\epsilon$, 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 $AG$ becomes larger, computation time of OSP-T increases while CPI maintains similar computation time. The effects of size of $AG$ 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 $10^{-6}$ 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 $\epsilon$ 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 15x, 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 $\Delta G$ of a fixed size. For brevity, we show the result on Orkut; results on other graphs are similar. $|\Delta G|$ is fixed to $5 \times 10^6$ 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 $O(m \log(1-c) \|q_{offset}\| L1)$. 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 $\log(1-c) \|q_{offset}\| L1$ indicates the number of iterations needed to converge, and is positively correlated with $\|q_{offset}\| L1$. From Section 3.3, $\|q_{offset}\| L1$ is decided by RWR scores of updated nodes: as RWR scores of updated nodes increase, $\|q_{offset}\| L1$ increases. Assume that $\Delta G$ 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,
Since TrackingPPR \cite{oh2019efficient}, the original time complexity is $O\left(\frac{|E|}{\epsilon} + \frac{1}{\epsilon} + \frac{1}{d}\right)$ where $\delta$ is a degree-normalized error tolerance per node and $d$ is the average degree 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 \cite{oh2019efficient} does not provide a time complexity model\footnote{In \cite{oh2019efficient}, the original time complexity is $O\left(\frac{|E|}{\epsilon} + \frac{1}{\epsilon} + \frac{1}{d}\right)$ where $\delta$ is a degree-normalized error tolerance per node such that $|\|\epsilon(t)\|/d(t) < \delta$ for residuals $r$ with any node $t$ at $t$th iteration. To consider the effect of changes in degree, we present $\delta$ as $\bar{d}\delta$ with the average degree $\bar{d}$.}, lazy PPR succeeds in analyzing its scalability based on its time complexity model.

### 4.4 Effects of $\epsilon$, error tolerance

To examine the effects of error tolerance $\epsilon$ on the performance of OSP-T, we check the $L1$ error and running time varying $\epsilon$. We report results on DBLP and Berkstan for brevity; results on other graphs are similar. As shown in Figure 6, as $\epsilon$ 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 $O$(1/1 error) is proportional to $\epsilon$ 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 $\epsilon$ work in real-world datasets.

### 4.5 Effects of $\Delta G$, a graph modification

#### 4.5.1 Size of $\Delta G$

In Figure 7, as the size of $\Delta G$ increases, the running time and $L1$ error of OSP-T also increase. Larger size of $\Delta G$ leads to longer $L1$ length of the starting vector $q_{offset}$ with longer computation time. Likewise, longer $L1$ length of $q_{offset}$ 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 $\epsilon$ is the error tolerance and $c$ is the restart probability.
Figure 6: Effects of error tolerance $\epsilon$ on OSP-T; as $\epsilon$ increases, the L1 error increases while the running time decreases.

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

Figure 8: Effects of location of $\Delta G$: when $\Delta G$ 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.

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 $q_{\text{offset}}$ 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.