TPA: Fast, Scalable, and Accurate Method for Approximate Random Walk with Restart on Billion Scale Graphs

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

Jinhong Jung
Seoul National University
jinhongjung@snu.ac.kr

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

***Abstract*—Given a large graph, how can we determine similarity between nodes in a fast and accurate way? Random walk with restart (RWR) is a popular measure for this purpose and has been exploited in numerous data mining applications including ranking, anomaly detection, link prediction, and community detection. However, previous methods for computing exact RWR require prohibitive storage sizes and computational costs, and alternative methods which avoid such costs by computing approximate RWR have limited accuracy.**

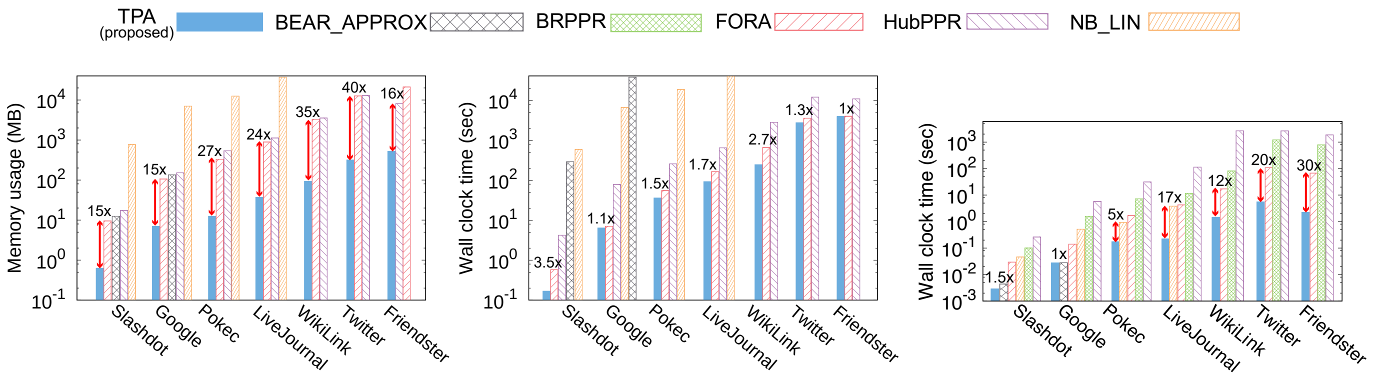
In this paper, we propose TPA, a fast, scalable, and highly accurate method for computing approximate RWR on large graphs. TPA exploits two important properties in RWR: 1) nodes close to a seed node are likely to be revisited in following steps due to block-wise structure of many real-world graphs, and 2) RWR scores of nodes which reside far from the seed node are proportional to their PageRank scores. Based on these two properties, TPA divides approximate RWR problem into two subproblems called neighbor approximation and stranger approximation. In the neighbor approximation, TPA estimates RWR scores of nodes close to the seed based on scores of few early steps from the seed. In the stranger approximation, TPA estimates RWR scores for nodes far from the seed using their PageRank. The stranger and neighbor approximations are conducted in the preprocessing phase and the online phase, respectively. Through extensive experiments, we show that TPA requires up to $3.5\times$ less time with up to $40\times$ less memory space than other state-of-the-art methods for the preprocessing phase. In the online phase, TPA computes approximate RWR up to $30\times$ faster than existing methods while maintaining high accuracy.

I. Introduction

Measuring similarity score between two nodes in a graph is widely recognized as a fundamental tool to analyze the graph and has been used in various data mining tasks to gain insights about the given graph [2], [4], [5]. Among many methods [9], [14], [18] to identify similarities within graphs, random walk with restart (RWR) [22], [23] has attracted considerable attention due to its ability to account for the global network structure from a particular user's point of view [8] and multi-faceted relationship between nodes in a graph [26]. RWR has been widely used in various applications across different domains including ranking [11], [27], community detection [31], [30], link prediction [3], graph similarity [16], and anomaly detection [25]. While RWR greatly expands its utility, it also brings a significant challenge on its computation - RWR scores are different across different seed nodes, and thus RWR needs to be recomputed for each new seed node.

To avoid enormous costs incurred by RWR computation, the majority of existing works focus on approximate RWR computation. BRPPR [6] improves RWR computation speed by limiting the amount of a Web graph data they need to access. NB-LIN [27] computes RWR approximately by exploiting low-rank matrix approximation. BEAR-APPROX [24] uses a block elimination approach and precomputes several matrices including the Schur complement to exploit them in online phase. FORA [29] combines two methods Forward Push and Monte Carlo Random Walk with an indexing scheme. Other methods such as FAST-PPR [20] and HubPPR [28] narrow down the scope of RWR problem (computing RWR scores from source to all nodes) by specifying a target node (computing a single RWR score between a source and the target node). However, those methods are not computation-efficient enough in terms of time and memory considering the amount of their sacrificed accuracy.

In this paper, we propose TPA (Two Phase Approximation for random walk with restart), a fast, scalable, and highly accurate method for computing approximate RWR scores on billion-scale graphs. TPA exploits two important properties in RWR: 1) nodes close to a seed node are likely to be revisited in following steps due to block-wise structure of many real world graphs, and 2) RWR scores of nodes which reside far from the seed node are proportional to their PageRank scores. Based on these two properties, TPA divides approximate RWR problem into two subproblems, the neighbor approximation and the stranger approximation. In the neighbor approximation, TPA estimates RWR scores of nodes close to the seed based on computation for few early steps from the seed. In the stranger approximation, TPA computes approximate RWR scores for nodes far from the seed using their PageRank scores. To divide an RWR problem into two subproblems, we use an iterative method, cumulative power iteration (CPI) which interprets an RWR problem as propagation of scores from a seed node across a graph. In CPI, ith iteration computes the distribution of propagated scores among nodes after i steps from the seed node. Based on CPI, the neighbor approximation handles iterations computed in early phase, while the stranger approximation estimates iterations computed in later phase. The stranger and neighbor approximation phases are conducted in



(a) Size of preprocessed data (b) Preprocessing time (c) Online time

Fig. 1: Performance of TPA: (a) compares the size of preprocessed data computed from preprocessing methods; (b) and (c) compare the preprocessing time and the online time, respectively, among approximate methods; Bars are omitted if the corresponding experiments run out of memory ($>$ 200GB). (a) TPA uses the least amount of space for preprocessed data among preprocessing methods. (b) In the preprocessing phase, TPA provides the fastest preprocessing speed among all preprocessing methods. (c) In the online phase, TPA computes approximate RWR scores faster than other competitors over all datasets. Details on these experiments are presented in Section IV.

the preprocessing phase and the online phase, respectively.

Through extensive experiments with various real-world graphs, we demonstrate the superiority of TPA over existing methods as shown in Figure 1. The main contributions of this paper are the followings:

- **Algorithm.** We propose TPA, a fast, scalable, and highly accurate algorithm for computing approximate RWR on billion-scale graphs (Algorithms 2 and 3). TPA efficiently approximates RWR scores in two phases: the stranger and the neighbor approximation phases by exploiting PageRank and block-wise structure of real graphs, respectively.
- **Analysis.** We present an analysis of TPA in terms of time complexity and memory requirement. We provide the theoretical approximation bounds for TPA and analyze reasons for the better approximation performance in practice than the theoretical bound suggested (Section III).
- **Experiment.** We present extensive empirical evidences for the performance of TPA using various large real-world graphs. We compare TPA with the state-of-the-art approximate RWR methods. Compared with other preprocessing methods, TPA needs $3.5\times$ less time and $40\times$ less memory for the preprocessing phase. In the online phase, TPA computes approximate RWR up to $30\times$ faster than other online methods, without sacrificing accuracy.

The code of our method and datasets used in the paper are available at http://datalab.snu.ac.kr/tpa. The rest of the paper is organized as follows. In Section II, we present preliminaries on RWR and CPI. In Section III, we describe the proposed algorithm TPA in detail along with its theoretical analysis. After presenting our experimental results in Section IV, we provide a review on related works in Section V and conclude in Section VI. The symbols frequently used in this paper are summarized in Table I and the real-world graph data used in our experiments are summarized in Table II. A brief description of each dataset is in Section IV.

TABLE I: Table of symbols.

Symbol	Definition
G	input graph
n	number of nodes in G
m	number of edges in G
s	seed node (= query node, source node)
c	restart probability
ϵ	convergence tolerance
$\mathbf{q}$	$(n \times 1)$ seed vector
$\mathbf{A}$	$(n \times n)$ adjacency matrix of G
$\tilde{\mathbf{A}}$	$(n \times n)$ row-normalized adjacency matrix of G
$\mathbf{r}_{\text{CPI}}$	$(n \times 1)$ RWR vector from CPI
$\mathbf{p}_{\text{CPI}}$	$(n \times 1)$ PageRank vector from CPI
$\mathbf{r}_{\text{TPA}}$	$(n \times 1)$ approximate RWR vector using neighbor and stranger approximation
S	$(n \times 1)$ starting iteration of neighbor part in CPI
T	$(n \times 1)$ starting iteration of stranger part in CPI
$\mathbf{x}^{(i)}$	$(n \times 1)$ interim score vector at ith iteration in CPI
$\mathbf{r}_{\text{family}}$	$(n \times 1)$ sum of $\mathbf{x}^{(i)}$ from 0 to $S-1$ iterations
$\mathbf{r}_{\text{neighbor}}$	$(n \times 1)$ sum of $\mathbf{x}^{(i)}$ from S to $T-1$ iterations
$\mathbf{r}_{\text{stranger}}$	$(n \times 1)$ sum of $\mathbf{x}^{(i)}$ from T to ∞ iterations

II. Preliminaries

In this section, we briefly review PageRank [21] algorithm which is used in our method for approximate value computation. Then, we describe our target problem RWR [22], and Cumulative Power Iteration (CPI) which computes RWR in an iterative matrix-vector multiplication form.

A. PageRank

PageRank [21] is a widely used algorithm to measure importance of vertices in a graph. The intuition behind PageRank is that a vertex is important if it is linked to by many important vertices. In other words, a vertex with large number of in-edges is estimated as an important vertex with high PageRank and a vertex with few in-edges is regarded as an unimportant

TABLE II: Dataset statistics: S denotes the starting iteration for the neighbor approximation and T denotes the starting iteration for the stranger approximation.

Dataset	Nodes	Edges	S	T
Friendster[1]	68,349,466	2,586,147,869	4	20
Twitter[1]	41,652,230	1,468,365,182	4	6
WikiLink[1]	12,150,976	378,142,420	5	6
LiveJournal[1]	4,847,571	68,475,391	5	10
Pokec[1]	1,632,803	30,622,564	5	10
Google[1]	875,713	5,105,039	5	20
Slashdot[1]	82,144	549,202	5	15

[1] http://konect.uni-koblenz.de/

vertex charged with low PageRank. PageRank scores for all nodes are represented as a PageRank score vector $\mathbf{p}$ which is defined by the following iterative equation:

$$\mathbf{p} = (1-c)\tilde{\mathbf{A}}^\top \mathbf{p} + \frac{c}{n}\mathbf{1}$$

where $\tilde{\mathbf{A}}$ is the row-normalized adjacency matrix, c is a restart probability, and $\mathbf{1}$ is an all-ones column vector of length n, the number of nodes. If $0 < c < 1$ and $\tilde{\mathbf{A}}$ is irreducible and aperiodic, $\mathbf{p}$ is guaranteed to converge to a unique solution [17].

B. Random Walk with Restart

Global view of vertex importance provided by PageRank does not reflect individual preferences. On the other hand, RWR measures each node's relevance w.r.t. a given seed node s in a graph. It assumes a random surfer who traverses the edges in the graph and occasionally restarts at node s. In each step, the surfer walks along edges with probability $1-c$ or jumps to the seed node with probability c. The iterative equation for an RWR score vector $\mathbf{r}$ is defined as follows:

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

where $\mathbf{q}$ is the seed vector with the index of the seed node s set to 1 and others to 0. In PageRank, $\mathbf{1}$ serves the role as a seed vector. The only difference between a random walk in PageRank and RWR is the seed vector: with the seed vector $\mathbf{1}$, a random walk in PageRank could restart from any node in the graph with uniform probability, while, with the seed vector $\mathbf{q}$, a random walk in RWR could restart only from the assigned seed node.

C. CPI: *Cumulative Power Iteration*

Cumulative Power Iteration (CPI) interprets an RWR problem as propagation of scores across a graph in an iterative matrix-vector multiplication form: score c is generated from the seed node in the beginning; at each step, scores are divided and propagated evenly into out-edges of their current nodes with decaying coefficient $1-c$; score x_v in a node v is propagated into n_v out-edged neighbors of v with value $\frac{1}{n_v}(1-c)x_v$. In a matrix-vector multiplication form, $\mathbf{x}^{(i)}$ 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 result

Algorithm 1: CPI Algorithm

```
Input: row-normalized adjacency matrix $\tilde{\mathbf{A}}$, seed nodes $\mathbf{S}$,
    restart probability $c$, convergence tolerance $\epsilon$, start iteration $s_{\text{iter}}$,
    and terminal iteration $t_{\text{iter}}$
Output: relevance score vector $\mathbf{r}$
 1: create a seed vector $\mathbf{q}$ from $\mathbf{S}$, i.e., $\mathbf{q}_s = 1/|\mathbf{S}|$ for $s$ in $\mathbf{S}$
    and the others are 0
 2: set $\mathbf{r} = \mathbf{0}$ and $\mathbf{x}^{(0)} = c\mathbf{q}$
 3: for iteration $i = 1$; $i \le t_{\text{iter}}$; $i$++ do
 4:     compute $\mathbf{x}^{(i)} \leftarrow (1-c)(\tilde{\mathbf{A}}^\top \mathbf{x}^{(i-1)})$
 5:     if $i \ge s_{\text{iter}}$ then
 6:         compute $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{x}^{(i)}$
 7:     end if
 8:     if $\|\mathbf{x}^{(i)}\|_1 < \epsilon$ then
 9:         break
10:     end if
11: end for
12: return $\mathbf{r}$
```

$\mathbf{r}_{\text{CPI}}$ as follows:

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

We show the correctness of CPI for RWR computation in the following Theorem 1.

Theorem 1: $\mathbf{r}_{\text{CPI}}$ is the true solution of the iterative equation $\mathbf{r} = (1-c)\tilde{\mathbf{A}}^\top \mathbf{r} + c\mathbf{q}$.

Proof: The spectral radius of $(1-c)\tilde{\mathbf{A}}^\top$ is less than one since $\tilde{\mathbf{A}}^\top$ is a column stochastic matrix, which implies that $\lim_{i\to\infty} c((1-c)\tilde{\mathbf{A}}^\top)^i \mathbf{q} = \mathbf{0}$. Then $\mathbf{r}_{\text{CPI}}$ is convergent, and the following computation shows that $\mathbf{r}_{\text{CPI}}$ obeys the steady state equation.

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

■

Note that there have been similar approaches [1], [10] as CPI to compute RWR, but they do not provide any algorithm in matrix-vector multiplication form. Thus, in this paper, we reinterpret the approaches as propagation of scores in an iterative matrix-vector multiplication form and name it CPI. In Algorithm 1, CPI accumulates only parts of the whole iterations using two input parameters, start iteration s_{iter} and terminal iteration t_{iter}. With s_{iter} and t_{iter}, CPI outputs the sum of $\mathbf{x}^{(i)}$ where $s_{\text{iter}} \le i \le t_{\text{iter}}$. To get the exact RWR from CPI, s_{iter} and t_{iter} are set to 0 and ∞, respectively. s_{iter} and t_{iter} are exploited in TPA (Algorithms 2 and 3 in Section III). At first, CPI creates a seed vector $\mathbf{q}$ (line 1). For PageRank, $\mathbf{q}$ is set to $\frac{1}{n}\mathbf{1}$, and for RWR, the index of the seed node s is set to 1 and others to 0 in $\mathbf{q}$. In ith iteration, scores in $\mathbf{x}^{(i-1)}$

from the previous iteration $(i-1)$ are propagated through $\tilde{\mathbf{A}}^\top$ with decaying coefficient $1-c$ (line 4). Then, interim score vector $\mathbf{x}^{(i)}$ is accumulated in RWR score vector $\mathbf{r}$ (line 6). In Algorithm 1, CPI returns the sum of iterations from s_{iter} to t_{iter} (line 3 and 5). Before the terminal iteration t_{iter}, CPI could stop iterations when the score vector $\mathbf{r}$ is converged with a convergence tolerance ϵ, and output $\mathbf{r}$ as a final score vector. $\|\mathbf{x}^{(i)}\|_1 < \epsilon$ is a condition for the final score vector $\mathbf{r}$ to converge (lines 8 $\sim$ 10). CPI could be used for PageRank and personalized PageRank which have several seed nodes.

III. Proposed Method

CPI performs iterations until convergence (i.e., $\|\mathbf{x}^{(i)}\|_1 < \epsilon$) to compute the RWR score vector $\mathbf{r}$. However, considerable amount of iterations are needed for convergence and computing all the iterations is not suitable for applications which require fast RWR computation speed. In this section, we propose TPA which approximates RWR scores with fast speed and high accuracy. We first divide the whole iterations in CPI into three parts as follows:

$$\begin{aligned}&\mathbf{r}_{\text{CPI}}\\&= \mathbf{r}_{\text{family}} + \mathbf{r}_{\text{neighbor}} + \mathbf{r}_{\text{stranger}}\\&= \underbrace{\mathbf{x}^{(0)} + \cdots + \mathbf{x}^{(S-1)}}_{\text{family part}} + \underbrace{\mathbf{x}^{(S)} + \cdots + \mathbf{x}^{(T-1)}}_{\text{neighbor part}} + \underbrace{\mathbf{x}^{(T)} + \cdots}_{\text{stranger part}}\end{aligned}$$

S denotes the starting iteration in $\mathbf{r}_{\text{neighbor}}$, and T denotes the starting iteration in $\mathbf{r}_{\text{stranger}}$. The family part $\mathbf{r}_{\text{family}} = \mathbf{x}^{(0)} + \cdots + \mathbf{x}^{(S-1)}$ denotes the propagation of scores into nearby nodes from the seed and comprises the iterations from 0th to $(S-1)$th in CPI. The neighbor part $\mathbf{r}_{\text{neighbor}} = \mathbf{x}^{(S)} + \cdots + \mathbf{x}^{(T-1)}$ denotes the propagation following the family part and comprises the iterations from Sth to $(T-1)$th. Finally, the rest propagation part, the iterations from Tth to the end, is denoted as $\mathbf{r}_{\text{stranger}} = \mathbf{x}^{(T)} + \cdots$. S and T are tuned to give a trade-off between accuracy and computation time (more details in Section III-C). Based on this partition, TPA approximates the exact RWR scores $\mathbf{r}_{\text{CPI}}$ by computing only $\mathbf{r}_{\text{family}}$ and estimating $\mathbf{r}_{\text{neighbor}}$ and $\mathbf{r}_{\text{stranger}}$.

$$\mathbf{r}_{\text{TPA}} = \mathbf{r}_{\text{family}} + \tilde{\mathbf{r}}_{\text{neighbor}} + \tilde{\mathbf{r}}_{\text{stranger}}$$

TPA approximates $\mathbf{r}_{\text{neighbor}}$ and $\mathbf{r}_{\text{stranger}}$ by the neighbor approximation phase and the stranger approximation phase, respectively. In the stranger approximation phase, TPA estimates $\mathbf{r}_{\text{stranger}}$ using PageRank. In the neighbor approximation phase, TPA approximates $\mathbf{r}_{\text{neighbor}}$ using $\mathbf{r}_{\text{family}}$ which is the only part computed exactly. Then the neighbor approximation and the stranger approximation are merged in the finalizing phase. The main ideas of our proposed method are summarized as follows:

- **TPA: stranger approximation** approximates the stranger part $\mathbf{r}_{\text{stranger}}$ in RWR with the stranger part $\mathbf{p}_{\text{neighbor}}$ in PageRank based on the observation that the distribution of scores in the stranger part is more affected by the distribution of edges than location of a seed node (Section III-A).
- **TPA: neighbor approximation** approximates the neighbor part $\mathbf{r}_{\text{neighbor}}$ using the family part $\mathbf{r}_{\text{family}}$ taking the advantage of block-wise structure of many real-world graphs (Section III-B).

Algorithm 2: Preprocessing phase of TPA

```
Input: row-normalized adjacency matrix $\tilde{\mathbf{A}}$, restart probability $c$,
    convergence tolerance $\epsilon$, and starting iteration $T$ of stranger part
Output: approximate stranger score vector $\tilde{\mathbf{r}}_{\text{stranger}}$
1: set seeds nodes $\mathbf{S} = \{1, \cdots, n\}$ for PageRank where $n$ is the
    number of nodes
2: $\tilde{\mathbf{r}}_{\text{stranger}} \leftarrow$ CPI ($\tilde{\mathbf{A}}$, $\mathbf{S}$, $c$, $\epsilon$, $T$, $\infty$) # Algorithm 1
3: return $\tilde{\mathbf{r}}_{\text{stranger}}$
```

Algorithm 3: Online phase of TPA

```
Input: row-normalized adjacency matrix $\tilde{\mathbf{A}}$, restart probability $c$,
    seed node $s$, convergence tolerance $\epsilon$, starting iteration $S$ of
    neighbor part, approximate stranger score vector $\tilde{\mathbf{r}}_{\text{stranger}}$
Output: TPA score vector $\mathbf{r}_{\text{TPA}}$
1: set a seed node $\mathbf{S} = \{s\}$ for RWR
2: $\mathbf{r}_{\text{family}} \leftarrow$ CPI ($\tilde{\mathbf{A}}$, $\mathbf{S}$, $c$, $\epsilon$, 0, $S-1$) # Algorithm 1
3: $\tilde{\mathbf{r}}_{\text{neighbor}} \leftarrow \frac{\|\mathbf{r}_{\text{neighbor}}\|_1}{\|\mathbf{r}_{\text{family}}\|_1}\mathbf{r}_{\text{family}}$
4: $\mathbf{r}_{\text{TPA}} \leftarrow \mathbf{r}_{\text{family}} + \tilde{\mathbf{r}}_{\text{neighbor}} + \tilde{\mathbf{r}}_{\text{stranger}}$
5: return $\mathbf{r}_{\text{TPA}}$
```

We describe each approximation phase with its accuracy analysis (Section III-A and III-B), and analyze time and space complexities of TPA (Section III-D).

A. Stranger Approximation

In the stranger approximation phase, TPA approximates the stranger part $\mathbf{r}_{\text{stranger}}$ using PageRank. PageRank score vector $\mathbf{p}_{\text{CPI}}$ is represented by CPI as follows:

$$\begin{aligned}&\mathbf{p}_{\text{CPI}}\\&= \mathbf{p}_{\text{family}} + \mathbf{p}_{\text{neighbor}} + \mathbf{p}_{\text{stranger}}\\&= \underbrace{\mathbf{x}'^{(0)} + \cdots + \mathbf{x}'^{(S-1)}}_{\text{family part}} + \underbrace{\mathbf{x}'^{(S)} + \cdots + \mathbf{x}'^{(T-1)}}_{\text{neighbor part}} + \underbrace{\mathbf{x}'^{(T)} + \cdots}_{\text{stranger part}}\end{aligned}$$

where $\mathbf{x}'^{(i)} = (1-c)\tilde{\mathbf{A}}^\top\mathbf{x}'^{(i-1)}$ and $\mathbf{x}'^{(0)} = \frac{c}{n}\mathbf{1}$. Note that the only difference between $\mathbf{r}_{\text{CPI}}$ and $\mathbf{p}_{\text{CPI}}$ is the seed vectors, $\mathbf{x}^{(0)}$ and $\mathbf{x}'^{(0)}$. Then, the stranger part $\mathbf{r}_{\text{stranger}}$ in RWR is approximated by the stranger part $\mathbf{p}_{\text{stranger}}$ in PageRank.

$$\tilde{\mathbf{r}}_{\text{stranger}} = \mathbf{p}_{\text{stranger}}$$

Intuition. The amount of scores propagated into each node are determined not only by the number of in-edges of each node, but also by the distance from the seed node. Nodes with many in-edges have many sources to receive scores, while nodes close to the seed node take in high scores since scores are decayed by factor $(1-c)$ as iteration progresses. However, scores $(\mathbf{x}^{(T)}, \mathbf{x}^{(T+1)}, \cdots)$ propagated in the stranger iterations are mainly determined by the number of in-edges since nodes receiving scores in the stranger iterations are already far from the seed, and thus the relative difference between their distances from the seed is too small to be considered. Note that PageRank score vector $\mathbf{p}_{\text{CPI}}$ presents the distribution of scores determined solely by the distribution of edges. This is the main motivation for the stranger approximation: approximate the stranger part $\mathbf{r}_{\text{stranger}}$ in RWR with $\mathbf{p}_{\text{stranger}}$ in PageRank. Since $\mathbf{p}_{\text{stranger}}$, the stranger part in PageRank is invariant regardless of which node is selected as a seed node, TPA precomputes $\tilde{\mathbf{r}}_{\text{stranger}}$ in the preprocessing phase (Algorithm 2).

Theoretical analysis. We show the accuracy bound of the stranger approximation in Lemma 1.

Lemma 1 (Accuracy bound for $\tilde{\mathbf{r}}_{stranger}$): Let $\mathbf{r}_{\text{stranger}}$ be the exact stranger part in CPI, $\tilde{\mathbf{r}}_{\text{stranger}}$ be the approximate stranger part via the stranger approximation, and T be the starting iteration of the stranger part. Then $\|\mathbf{r}_{\text{stranger}}-\tilde{\mathbf{r}}_{\text{stranger}}\|_1 \leq 2(1-c)^T$.

Proof: $\mathbf{r}_{\text{stranger}}$ and $\tilde{\mathbf{r}}_{\text{stranger}}$ are represented as follows:

$$\mathbf{r}_{\text{stranger}} = \mathbf{x}^{(T)} + \mathbf{x}^{(T+1)} + \cdots$$
$$\tilde{\mathbf{r}}_{\text{stranger}} = \mathbf{x}'^{(T)} + \mathbf{x}'^{(T+1)} + \cdots$$

Then, $\|\mathbf{r}_{\text{stranger}} - \tilde{\mathbf{r}}_{\text{stranger}}\|_1$ is bounded as follows:

$$\|\mathbf{r}_{\text{stranger}} - \tilde{\mathbf{r}}_{\text{stranger}}\|_1 = \|(\mathbf{x}^{(T)} + \cdots) - (\mathbf{x}'^{(T)} + \cdots)\|_1 \leq \sum_{i=T}^{\infty} \|\mathbf{x}^{(i)} - \mathbf{x}'^{(i)}\|_1$$

where the interim score vectors $\mathbf{x}^{(i)}$ and $\mathbf{x}'^{(i)}$ at i-th iteration in CPI are represented as follows:

$$\mathbf{x}^{(i)} = (1-c)\tilde{\mathbf{A}}^\top \mathbf{x}^{(i-1)} = c(1-c)^i (\tilde{\mathbf{A}}^\top)^i \mathbf{q}$$
$$\mathbf{x}'^{(i)} = (1-c)\tilde{\mathbf{A}}^\top \mathbf{x}'^{(i-1)} = c(1-c)^i (\tilde{\mathbf{A}}^\top)^i \mathbf{b}$$

where $\mathbf{q}$ is s-th unit vector, and $\mathbf{b} = \frac{1}{n}\mathbf{1}$. Suppose $(\tilde{\mathbf{A}}^\top)^i$ is represented by $\left[\mathbf{c}_1^{(i)}, \cdots, \mathbf{c}_n^{(i)}\right]$ where $\mathbf{c}_j^{(i)}$ is j-th column of the matrix $(\tilde{\mathbf{A}}^\top)^i$, and n is the number of nodes. Then, $\mathbf{x}^{(i)} - \mathbf{x}'^{(i)}$ is represented as follows:

$$\begin{aligned}\mathbf{x}^{(i)} - \mathbf{x}'^{(i)} &= c(1-c)^i (\tilde{\mathbf{A}}^\top)^i (\mathbf{q} - \mathbf{b}) \\ &= c(1-c)^i (\tilde{\mathbf{A}}^\top)^i \Big(-\frac{1}{n}, -\frac{1}{n}, \cdots, \underbrace{\frac{n-1}{n}}_{s\text{-th entry}}, -\frac{1}{n}, \cdots\Big)^\top \\ &= c(1-c)^i \Big(-\frac{1}{n}\mathbf{c}_1^{(i)} \cdots + \frac{n-1}{n}\mathbf{c}_s^{(i)} \cdots - \frac{1}{n}\mathbf{c}_n^{(i)}\Big) \\ &= \frac{c(1-c)^i}{n} \sum_{j \neq s} (\mathbf{c}_s^{(i)} - \mathbf{c}_j^{(i)})\end{aligned}$$

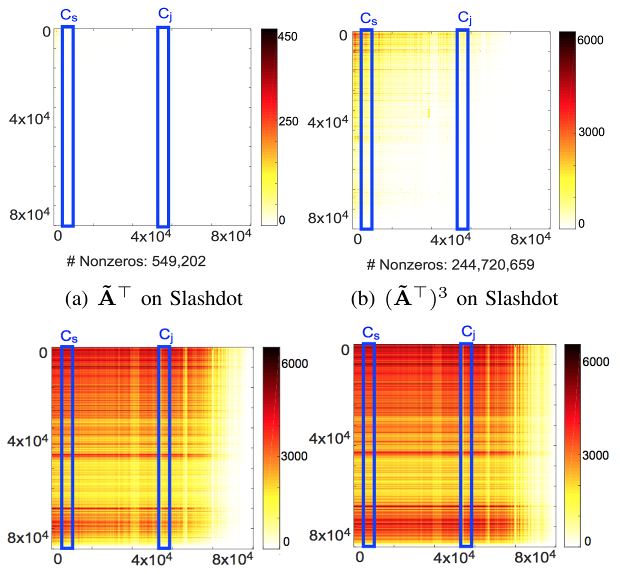
Then, $\|\mathbf{x}^{(i)} - \mathbf{x}'^{(i)}\|_1$ is bounded by the following inequality:

$$\begin{aligned}\|\mathbf{x}^{(i)} - \mathbf{x}'^{(i)}\|_1 &= \frac{c(1-c)^i}{n} \Big\|\sum_{j \neq s} (\mathbf{c}_s^{(i)} - \mathbf{c}_j^{(i)})\Big\|_1 \\ &\leq \frac{c(1-c)^i}{n} \sum_{j \neq s} \|\mathbf{c}_s^{(i)} - \mathbf{c}_j^{(i)}\|_1 \\ &\leq \frac{c(1-c)^i}{n} \times 2(n-1) \leq 2c(1-c)^i\end{aligned}$$

where in the second inequality we use the fact that $\|\mathbf{c}_j^{(i)}\|_1 = 1$ and $\|\mathbf{c}_s^{(i)} - \mathbf{c}_j^{(i)}\|_1 \leq \|\mathbf{c}_s^{(i)}\|_1 + \|\mathbf{c}_j^{(i)}\|_1 = 2$, since $\tilde{\mathbf{A}}^\top$ as well as $(\tilde{\mathbf{A}}^\top)^i$ are column stochastic. Then, $\|\mathbf{r}_{\text{stranger}} - \tilde{\mathbf{r}}_{\text{stranger}}\|_1$ is bounded as follows:

$$\begin{aligned}\|\mathbf{r}_{\text{stranger}} - \tilde{\mathbf{r}}_{\text{stranger}}\|_1 &\leq \sum_{i=T}^{\infty} \|\mathbf{x}^{(i)} - \mathbf{x}'^{(i)}\|_1 \\ &\leq \sum_{i=T}^{\infty} 2c(1-c)^i = 2(1-c)^T\end{aligned}$$

■



(a) $\tilde{\mathbf{A}}^\top$ on Slashdot (b) $(\tilde{\mathbf{A}}^\top)^3$ on Slashdot

(c) $(\tilde{\mathbf{A}}^\top)^5$ on Slashdot (d) $(\tilde{\mathbf{A}}^\top)^7$ on Slashdot

Fig. 2: Distribution of nonzeros in $(\tilde{\mathbf{A}}^\top)^i$ on Slashdot dataset: as i increases, $(\tilde{\mathbf{A}}^\top)^i$ has more nonzeros with denser columns $\mathbf{c}_s$ and $\mathbf{c}_j$. Colorbars present the number of nonzeros.

Real-world graphs. From the proof of Lemma 1, given a seed node s, the $L1$ difference $\|\mathbf{c}_s^{(i)} - \mathbf{c}_j^{(i)}\|_1$ between column $\mathbf{c}_s^{(i)}$ and other columns $\mathbf{c}_j^{(i)}$ of a matrix $(\tilde{\mathbf{A}}^\top)^i$ ($i = T, T+1, \cdots$) is a determining factor for accuracy of the stranger approximation. Considering that $(\tilde{\mathbf{A}}^\top)^i$ is a column stochastic matrix, and thus, its columns $\mathbf{c}_s^{(i)}$ and $\mathbf{c}_j^{(i)}$ are unit vectors with all non-negative entries, $\|\mathbf{c}_s^{(i)} - \mathbf{c}_j^{(i)}\|_1$ becomes large (close to its maximum 2) when $\mathbf{c}_s^{(i)}$ and $\mathbf{c}_j^{(i)}$ have their nonzero values in the different indices from each other. Note that $\tilde{\mathbf{A}}$ is a row-normalized adjacency matrix of a real-world graph with low density [15]. As raising the matrix $\tilde{\mathbf{A}}^\top$ to the ith power, $(\tilde{\mathbf{A}}^\top)^i$ tends to be a denser matrix with denser column vectors. We present this tendency in the Slashdot dataset in Figure 2. Then, the dense unit vectors $\mathbf{c}_s^{(i)}$ and $\mathbf{c}_j^{(i)}$ are likely to have nonzero values in the same indices resulting a small value of $\|\mathbf{c}_s^{(i)} - \mathbf{c}_j^{(i)}\|_1$. To show this tendency in real-world graphs, we estimate the number of nonzeros in $(\tilde{\mathbf{A}}^\top)^i$ and the average value for $C_i = \frac{1}{n}\sum_{j \neq s}\|\mathbf{c}_s^{(i)} - \mathbf{c}_j^{(i)}\|_1$ with 30 random seeds s on the Slashdot and Google datasets. In Figure 3, as i increases, the number of nonzeros increases while C_i decreases. This shows that the stranger approximation which approximates the stranger iterations $\mathbf{x}^{(i)}$ with high i values ($i \geq T$) would lead to smaller errors in practice than the bound suggested in Lemma 1. However, setting T, the starting iteration of the stranger approximation, with too high values leads to high errors in TPA since high values for T lead to high errors in the neighbor approximation. The reasons will be discussed concretely in Section III-C. Through extensive experiments (Section IV-C), we present the high accuracy of the stranger approximation in real-world graphs.

B. Neighbor Approximation

In the online phase, the remaining parts $\mathbf{r}_{\text{family}}$ and $\mathbf{r}_{\text{neighbor}}$ need to be computed based on an assigned seed node. Even though we need to compute only T iterations $(\mathbf{x}^{(0)}, \cdots, \mathbf{x}^{(T-1)})$ in the online phase with the help of the stranger approximation, calculating T iterations are still demanding in terms of running time. To handle this issue, TPA introduces the second approximation phase, the neighbor approximation. The neighbor approximation reduces running time further by limiting computation to the family part $\mathbf{r}_{\text{family}}$, and estimates the neighbor part $\mathbf{r}_{\text{neighbor}}$ by scaling $\mathbf{r}_{\text{family}}$ as follows:

$$\tilde{\mathbf{r}}_{\text{neighbor}} = \frac{\|\mathbf{r}_{\text{neighbor}}\|_1}{\|\mathbf{r}_{\text{family}}\|_1}\mathbf{r}_{\text{family}} = \frac{(1-c)^S - (1-c)^T}{1-(1-c)^S}\mathbf{r}_{\text{family}}$$

With restart probability c, the L1 norms of $\mathbf{r}_{\text{family}}$ and $\mathbf{r}_{\text{neighbor}}$ only depend on S and T, the starting numbers of the neighbor iterations and the stranger iterations, respectively (see Lemma 2).

Lemma 2 (L1 norms of $\mathbf{r}_{family}$ and $\mathbf{r}_{neighbor}$): $\|\mathbf{r}_{\text{family}}\|_1$ and $\|\mathbf{r}_{\text{neighbor}}\|_1$ are $1-(1-c)^S$ and $(1-c)^S-(1-c)^T$, respectively.

Proof: The family part $\mathbf{r}_{\text{family}}$ and the neighbor part $\mathbf{r}_{\text{neighbor}}$ are represented as follows:

$$\mathbf{r}_{\text{family}} = \mathbf{x}^{(0)} + \mathbf{x}^{(1)} + \cdots + \mathbf{x}^{(S-1)}$$
$$\mathbf{r}_{\text{neighbor}} = \mathbf{x}^{(S)} + \mathbf{x}^{(S+1)} + \cdots + \mathbf{x}^{(T-1)}$$

where $\mathbf{x}^{(i)} = c(1-c)^i(\tilde{\mathbf{A}}^\top)^i\mathbf{q}$. Then $\|\mathbf{r}_{\text{family}}\|_1$ and $\|\mathbf{r}_{\text{neighbor}}\|_1$ are represented as follows:

$$\|\mathbf{r}_{\text{family}}\|_1 = \|\mathbf{x}^{(0)} + \mathbf{x}^{(1)} + \cdots + \mathbf{x}^{(S-1)}\|_1 = \sum_{i=0}^{S-1}\|\mathbf{x}^{(i)}\|_1$$

$$\|\mathbf{r}_{\text{neighbor}}\|_1 = \|\mathbf{x}^{(S)} + \mathbf{x}^{(S+1)} + \cdots + \mathbf{x}^{(T-1)}\|_1 = \sum_{i=S}^{T-1}\|\mathbf{x}^{(i)}\|_1$$

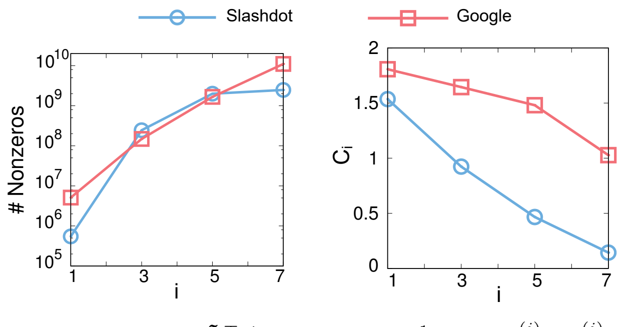
Note that all entries of $\mathbf{x}^{(i)}$ are non-negative. Since $\tilde{\mathbf{A}}^\top$ is a column stochastic matrix, $(\tilde{\mathbf{A}}^\top)^i$ is also a column stochastic matrix. Hence, $\|(\tilde{\mathbf{A}}^\top)^i\mathbf{q}\|_1 = \|\mathbf{q}\|_1 = 1$ and $\|\mathbf{x}^{(i)}\|_1 = \|c(1-c)^i(\tilde{\mathbf{A}}^\top)^i\mathbf{q}\|_1 = c(1-c)^i$. Then $\|\mathbf{r}_{\text{family}}\|_1$ and $\|\mathbf{r}_{\text{neighbor}}\|_1$ are written as follows:

$$\|\mathbf{r}_{\text{family}}\|_1 = \sum_{i=0}^{S-1} c(1-c)^i = 1-(1-c)^S$$
$$\|\mathbf{r}_{\text{neighbor}}\|_1 = \sum_{i=S}^{T-1} c(1-c)^i = (1-c)^S-(1-c)^T$$

■

In the online phase, TPA computes $\mathbf{r}_{\text{family}}$ at first, and estimates $\mathbf{r}_{\text{neighbor}}$ based on the neighbor approximation. Finally, TPA merges $\mathbf{r}_{\text{family}}$, $\tilde{\mathbf{r}}_{\text{neighbor}}$ and $\tilde{\mathbf{r}}_{\text{stranger}}$, and computes the approximate RWR score vector $\mathbf{r}_{\text{TPA}}$ (Algorithm 3).

Intuition. In many real-world graphs, nodes inside a community are densely inter-connected to each other than to nodes in other communities. This is an important property of real-world graphs called block-wise, community-like structure and widely exploited in graph mining [25], [27]. Our intuition for the neighbor approximation comes from this property. Based on block-wise structure, scores started from one community are likely to be propagated into nodes in the same community



(a) Nonzeros in $(\tilde{\mathbf{A}}^\top)^i$ (b) $C_i = \frac{1}{n}\sum_{j\neq s}\|\mathbf{c}_s^{(i)} - \mathbf{c}_j^{(i)}\|_1$

Fig. 3: Number of nonzeros and C_i of $(\tilde{\mathbf{A}}^\top)^i$: as i increases, the number of nonzeros increases while C_i decreases.

repeatedly for a while. Then we could assume that the nodes which receive scores in the early iterations (the family part) would receive scores again in the following iterations (the neighbor part). Furthermore, the nodes which have more in-edges thus receive more scores in the early iterations would receive more scores than other nodes in the following iterations. Note that scores propagated in the following iterations would be smaller than scores in the early iterations since scores are decayed by the decaying coefficient $(1-c)$ as iterations progress. Based on this assumption, we maintain ratios of scores among nodes in $\mathbf{r}_{\text{family}}$ and scale the scores with $\frac{\|\mathbf{r}_{\text{neighbor}}\|_1}{\|\mathbf{r}_{\text{family}}\|_1}$ to reflect the smaller amount of scores in $\mathbf{r}_{\text{neighbor}}$. This is the main motivation for the neighbor approximation based on block-wise structure of real-world graphs.

Theoretical analysis. We show the accuracy bound for the neighbor approximation in Lemma 3, and the total accuracy bound for our proposed method TPA in Theorem 2.

Lemma 3 (Accuracy bound for $\tilde{\mathbf{r}}_{neighbor}$): Let $\mathbf{r}_{\text{neighbor}}$ be the exact neighbor part in CPI, and $\tilde{\mathbf{r}}_{\text{neighbor}}$ be the approximate neighbor part via the neighbor approximation. Then $\|\mathbf{r}_{\text{neighbor}} - \tilde{\mathbf{r}}_{\text{neighbor}}\|_1 \leq 2(1-c)^S - 2(1-c)^T$.

Proof: For brevity, let $\tilde{\mathbf{A}}^\top \rightarrow \bar{\mathbf{A}}$, $c\mathbf{q} \rightarrow \bar{\mathbf{q}}$ and $\mathbf{r}_{\text{family}} = \bar{\mathbf{q}} + (1-c)\bar{\mathbf{A}}\bar{\mathbf{q}} + \cdots + ((1-c)\bar{\mathbf{A}})^{S-1}\bar{\mathbf{q}} \rightarrow \mathbf{f}$. We set $T = kS$ for simplicity of proof. Then $\mathbf{r}_{\text{neighbor}}$ and $\tilde{\mathbf{r}}_{\text{neighbor}}$ are represented as follows:

$$\begin{aligned}\mathbf{r}_{\text{neighbor}} =& ((1-c)\bar{\mathbf{A}})^S\bar{\mathbf{q}} + \cdots + ((1-c)\bar{\mathbf{A}})^{2S-1}\bar{\mathbf{q}} \\ &+ ((1-c)\bar{\mathbf{A}})^{2S}\bar{\mathbf{q}} + \cdots + ((1-c)\bar{\mathbf{A}})^{3S-1}\bar{\mathbf{q}} \\ &+ \cdots \\ &+ ((1-c)\bar{\mathbf{A}})^{(k-1)S}\bar{\mathbf{q}} + \cdots + ((1-c)\bar{\mathbf{A}})^{kS-1}\bar{\mathbf{q}} \\ =& ((1-c)\bar{\mathbf{A}})^S\mathbf{f} + \cdots + ((1-c)\bar{\mathbf{A}})^{(k-1)S}\mathbf{f} \\ \tilde{\mathbf{r}}_{\text{neighbor}} =& \frac{\|\mathbf{r}_{\text{neighbor}}\|_1}{\|\mathbf{r}_{\text{family}}\|_1}\mathbf{r}_{\text{family}} \\ =& \frac{(1-c)^S-(1-c)^T}{1-(1-c)^S}\mathbf{f}\end{aligned}$$

Note that $\frac{(1-c)^S-(1-c)^T}{1-(1-c)^S}$ could be expressed as follows:

$$\begin{aligned}\frac{(1-c)^S-(1-c)^T}{1-(1-c)^S} &= \frac{(1-c)^S(1-(1-c)^{(k-1)S})}{1-(1-c)^S} \\ &= (1-c)^S + \cdots + (1-c)^{(k-1)S}\end{aligned}$$

Then $\tilde{\mathbf{r}}_{\text{neighbor}}$ is presented as follows:

$$\tilde{\mathbf{r}}_{\text{neighbor}} = (1-c)^S\mathbf{f} + \cdots + (1-c)^{(k-1)S}\mathbf{f}$$

Then $\mathbf{r}_{\text{neighbor}} - \tilde{\mathbf{r}}_{\text{neighbor}}$ is written as follows:

$$\begin{aligned}
&\mathbf{r}_{\text{neighbor}} - \tilde{\mathbf{r}}_{\text{neighbor}} \\
&= (1-c)^S(\bar{\mathbf{A}}^S\mathbf{f} - \mathbf{f}) + (1-c)^{2S}(\bar{\mathbf{A}}^{2S}\mathbf{f} - \mathbf{f}) \\
&+ \cdots \\
&+ (1-c)^{(k-1)S}(\bar{\mathbf{A}}^{(k-1)S}\mathbf{f} - \mathbf{f}) \\
&= \sum_{i=1}^{k-1}(1-c)^{iS}(\bar{\mathbf{A}}^{iS}\mathbf{f} - \mathbf{f})
\end{aligned}$$

Hence, $\|\mathbf{r}_{\text{neighbor}} - \tilde{\mathbf{r}}_{\text{neighbor}}\|_1$ is bounded as follows:

$$\begin{aligned}
\|\mathbf{r}_{\text{neighbor}} - \tilde{\mathbf{r}}_{\text{neighbor}}\|_1 &\leq \sum_{i=1}^{k-1}(1-c)^{iS}\|(\bar{\mathbf{A}}^{iS}\mathbf{f} - \mathbf{f})\|_1 \\
&\leq \sum_{i=1}^{k-1}(1-c)^{iS}(\|\bar{\mathbf{A}}^{iS}\mathbf{f}\|_1 + \|\mathbf{f}\|_1) \\
&= 2\|\mathbf{f}\|_1\sum_{i=1}^{k-1}(1-c)^{iS} \\
&= 2\frac{(1-c)^S(1-(1-c)^{(k-1)S})}{1-(1-c)^S}\|\mathbf{f}\|_1 \\
&= 2\frac{(1-c)^S - (1-c)^{kS}}{1-(1-c)^S}(1-(1-c)^S) \\
&= 2(1-c)^S - 2(1-c)^T
\end{aligned}$$

Note that $\|\bar{\mathbf{A}}^{iS}\mathbf{f}\|_1 = \|\mathbf{f}\|_1$ since $\bar{\mathbf{A}}$ is a column stochastic matrix; thus, $\bar{\mathbf{A}}^{iS}$ is also a column stochastic matrix . ■

Real-world graphs. From the proof of Lemma 3, $\|(\bar{\mathbf{A}}^{iS}\mathbf{f} - \mathbf{f})\|_1$ $(i = 1, \cdots, k-1)$ is a decisive factor for the accuracy of the neighbor approximation. $\mathbf{f}$ has the distribution of scores among nodes after the family iterations $(\mathbf{x}^{(0)}, \cdots, \mathbf{x}^{(S-1)})$. Multiplying $\mathbf{f}$ with $\bar{\mathbf{A}}^S$ means that scores in $\mathbf{f}$ are propagated S steps further across a given graph. As shown in Figure 4, if the graph has an ideal block-wise, community-wise structure, scores in $\mathbf{f}$ would be mainly located in nodes around a seed node, which belong to the same community as the seed. During the next S steps, scores in $\mathbf{f}$ would be propagated into the nodes belonging to the same community again, without leaking into other communities. Then, the distribution of scores in $\bar{\mathbf{A}}^S\mathbf{f}$ would be similar to that in $\mathbf{f}$. To show that block-wise structure of real-world graphs also brings the similar effects, we compare $\|(\bar{\mathbf{A}}^S\mathbf{f} - \mathbf{f})\|_1$ of real-world graphs (WikiLink, LiveJournal, Pokec, Google, and Slashdot) with that of random graphs. Random graphs have the same numbers of nodes and edges as the corresponding real-world graphs, while having the random distribution of edges rather than block-wise structure. Restart probability c is set to 0.15 and S is set to 5 for all the datasets as described in Table II. As shown in Figure 5, real-world graphs have lower $\|(\bar{\mathbf{A}}^S\mathbf{f} - \mathbf{f})\|_1$ values than random graphs across all datasets. This means that the distribution of scores $(\bar{\mathbf{A}}^S\mathbf{f})$ after S steps is similar to the previous distribution $(\mathbf{f})$ in real-world graphs with the help of block-wise structure. By this process, the neighbor approximation succeeds in achieving high accuracy for real-world graphs.

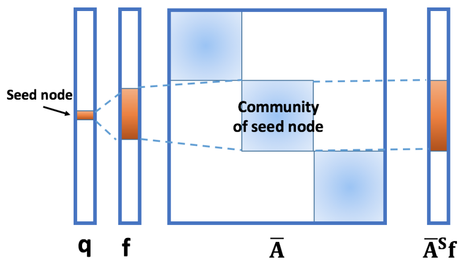


Fig. 4: Neighbor approximation with an ideal block-wise structure: with an ideal block-wise structure, $\mathbf{f}$ and $\bar{\mathbf{A}}^S\mathbf{f}$ have similar distributions.

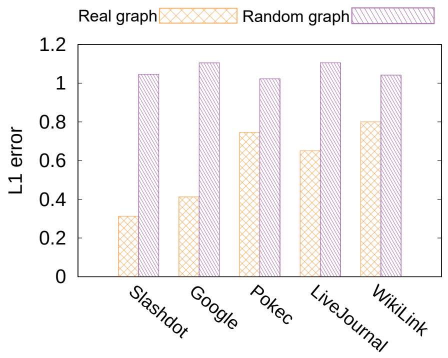


Fig. 5: Comparing $\|(\bar{\mathbf{A}}^S\mathbf{f} - \mathbf{f})\|_1$ between real-world and random graphs: $\bar{\mathbf{A}}^S\mathbf{f}$ and $\mathbf{f}$ have closer values in real-world graphs with block-wise structures than those in random graphs across all datasets.

Based on Lemmas 1 and 3, we present the total accuracy bound for TPA in Theorem 2. Note that TPA achieves higher accuracy in practice than the bound suggested in Theorem 2 as discussed in this section. We show high accuracy of TPA in real-world graphs experimentally in Section IV-C.

Theorem 2 (Accuracy bound for TPA*):* Let $\mathbf{r}_{\text{CPI}}$ be the exact RWR score vector from CPI, and $\mathbf{r}_{\text{TPA}}$ be the approximate RWR score vector from TPA. Then $\|\mathbf{r}_{\text{CPI}} - \mathbf{r}_{\text{TPA}}\|_1$ is bounded as follows:

$$\|\mathbf{r}_{\text{CPI}} - \mathbf{r}_{\text{TPA}}\|_1 \leq 2(1-c)^S$$

Proof: Note that $\mathbf{r}_{\text{CPI}}$ and $\mathbf{r}_{\text{TPA}}$ are represented as follows:

$$\mathbf{r}_{\text{CPI}} = \mathbf{r}_{\text{family}} + \mathbf{r}_{\text{neighbor}} + \mathbf{r}_{\text{stranger}}$$
$$\mathbf{r}_{\text{TPA}} = \mathbf{r}_{\text{family}} + \tilde{\mathbf{r}}_{\text{neighbor}} + \tilde{\mathbf{r}}_{\text{stranger}}$$

Then $\|\mathbf{r}_{\text{CPI}} - \mathbf{r}_{\text{TPA}}\|_1$ is bounded as the following inequality:

$$\begin{aligned}
\|\mathbf{r}_{\text{CPI}} - \mathbf{r}_{\text{TPA}}\|_1 &= \|\mathbf{r}_{\text{neighbor}} - \tilde{\mathbf{r}}_{\text{neighbor}} + \mathbf{r}_{\text{stranger}} - \tilde{\mathbf{r}}_{\text{stranger}}\|_1 \\
&\leq \|\mathbf{r}_{\text{neighbor}} - \tilde{\mathbf{r}}_{\text{neighbor}}\|_1 + \|\mathbf{r}_{\text{stranger}} - \tilde{\mathbf{r}}_{\text{stranger}}\|_1 \\
&\leq 2(1-c)^T + 2(1-c)^S - 2(1-c)^T \\
&= 2(1-c)^S
\end{aligned}$$

Note that $\|\mathbf{r}_{\text{neighbor}} - \tilde{\mathbf{r}}_{\text{neighbor}}\|_1 \leq 2(1-c)^S - 2(1-c)^T$ by Lemma 3 and $\|\mathbf{r}_{\text{stranger}} - \tilde{\mathbf{r}}_{\text{stranger}}\|_1 \leq 2(1-c)^T$ by Lemma 1. ■

According to Theorem 2, the accuracy of TPA is bounded by S, the starting iteration of the neighbor approximation. Note that S also determines the scope of $\mathbf{r}_{\text{family}}$, thus the amount of computation needed in the online phase. TPA trades off the accuracy and the online computation cost using S.

C. Selecting S and T

We set the starting iteration S of the neighbor approximation considering accuracy and speed since S gives a tradeoff between them. If we set S to a large value, computation time for $\mathbf{r}_{\text{family}}$ in the online phase escalates sharply. Otherwise, when we set S to a small value, error increases since a portion of exact computation decreases.

When we set the starting iteration T of the stranger approximation to a small value, error increases sharply. Intuitively, with small T, the effect of PageRank becomes higher than that of a seed node. Theoretically, we discussed the reason with the error bound of the stranger approximation in Section III-A. Otherwise, when we choose a large T, scores of nodes far from the seed, the latter part of $\mathbf{r}_{\text{neighbor}}$ such as $\mathbf{x}^{(T-1)}$ and $\mathbf{x}^{(T-2)}$, are estimated by the family part $\mathbf{r}_{\text{family}}$ in the neighbor approximation. Nodes far from the seed are likely to belong to different communities from that of the seed node. Considering that the neighbor approximation assumes that nodes visited in the family part and in the neighbor part belong to the same community by block-wise structure of real-world graphs, errors for the neighbor approximation increase significantly. Thus we need to choose T with a value which minimizes the total errors for TPA. We show the effects of S and T on the speed and accuracy of TPA in Section IV-D.

D. Complexity analysis for TPA

We analyze the time and space complexities of TPA. First, we evaluate the time complexity of CPI since TPA is based on CPI.

Lemma 4 (Time Complexity of CPI*):* At each iteration, CPI takes $O(m)$ where m is the number of edges in a given graph. In total, CPI takes $O(m \log_{(1-c)}(\frac{\epsilon}{c}))$ time where $\log_{(1-c)}(\frac{\epsilon}{c})$ indicates the number of iterations needed for convergence.

Proof: CPI computes $\mathbf{x}^{(i)} = (1-c)(\tilde{\mathbf{A}}^\top \mathbf{x}^{(i-1)})$ for each iteration, and takes $O(m)$ time where m is the number of nonzeros in $\tilde{\mathbf{A}}$. CPI stops the iteration with convergence when $\|\mathbf{x}^{(i)}\|_1 = c(1-c)^i < \epsilon$. Then the number of iterations to be converged is $\log_{(1-c)}(\frac{\epsilon}{c})$ and the total computation time is $O(m \log_{(1-c)}(\frac{\epsilon}{c}))$. ■

Theorem 3 (Time Complexity of TPA*):* TPA takes $O(m \log_{(1-c)}(\frac{\epsilon}{c}))$ in the preprocessing phase and $O(mS)$ in the online phase where S is the starting iteration of the neighbor approximation.

Proof: In the preprocessing phase, TPA computes PageRank using CPI which takes $O(m \log_{(1-c)}(\frac{\epsilon}{c}))$ time. In the online phase, TPA computes $\mathbf{r}_{\text{family}}$ which runs S iterations in CPI; thus, it requires $O(mS)$ time. ■

According to Theorem 3, the preprocessing cost and the online cost of TPA mainly depend on the number of iterations conducted in CPI. Since only the family part is computed in the online phase, TPA demands much smaller costs compared to other state-of-the-art methods as shown in Figure 1.

Theorem 4 (Space complexity of TPA*):* TPA requires $O(n+m)$ memory space where n and m are the numbers of vertices and edges, respectively.

Proof: TPA requires $O(n)$ memory space for an approximate stranger score vector $\tilde{\mathbf{r}}_{\text{stranger}}$ and $O(m)$ memory space for a row-normalized adjacency matrix $\tilde{\mathbf{A}}$. ■

Theorem 4 indicates that the space cost of TPA mainly depends on $n+m$, node and edge sizes of the given graph. As shown in Figure 1(a), TPA requires reasonable memory space, and thus, succeeds in processing billion-scale graphs.

IV. Experiments

In this section, we experimentally evaluate the performance of TPA compared to other approximate RWR methods. We aim to answer the following questions:

- **Q1 Performance of TPA.** How much does TPA enhance the computational efficiency compared with its competitors? (Section IV-B)
- **Q2 Accuracy of TPA.** How much does TPA reduce the approximation error in real-world graphs from the theoretical error bound? (Section IV-C)
- **Q3 Effects of parameters.** How does the starting iteration S of the neighbor approximation affect the accuracy and speed of TPA? How does the starting iteration T of the stranger approximation affect the accuracy of TPA? (Section IV-D)

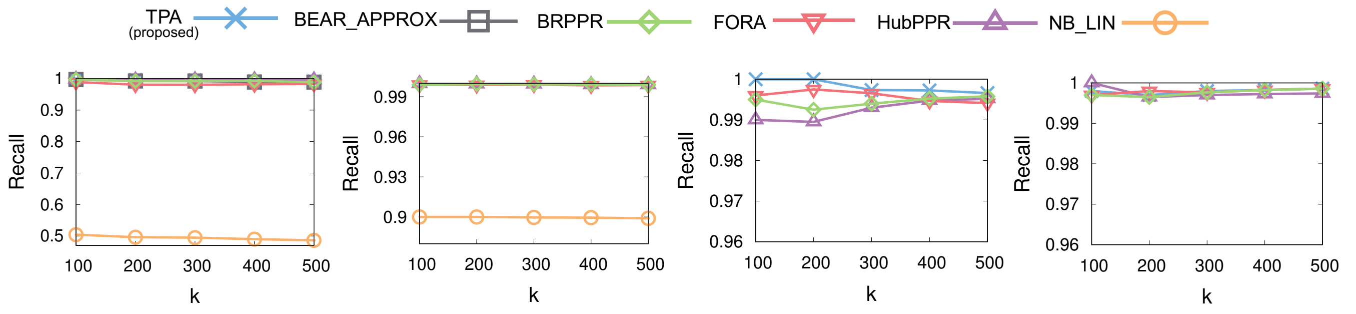
A. Setup

1) Datasets

We use 7 real-world graphs to evaluate the effectiveness and efficiency of our method. The datasets and their statistics are summarized in Table II. Among them, Friendster, Twitter, LiveJournal, Pokec, and Slashdot are social networks, whereas WikiLink and Google are hyperlink networks.

2) Environment

All experiments are conducted on a workstation with a single core Intel(R) Xeon(R) CPU E5-2630 @ 2.2GHz and 200GB memory. We compare TPA with five state-of-the-art approximate RWR methods, BRPPR [6], NB-LIN [27], BEAR-APPROX [24], HubPPR [28], and FORA [29], all of which are described in Section V. All these methods including TPA choose an implementation showing a better performance between MATLAB and C++. We set the restart probability c to 0.15. The starting iteration S of the neighbor approximation and the starting iteration T of the stranger approximation is set differently for each graph as noted in Table II to gain the best performance of TPA. The convergence tolerance ϵ for CPI is set to 10^{-9}. For each dataset, we measure the average value for 30 random seed nodes. To show the best performance, parameters of each competitor are set as follows: the drop tolerance of BEAR-APPROX and NB-LIN is set to $n^{-1/2}$ and 0, respectively; the threshold to expand nodes in RPPR and BRPPR is set to 10^{-4}; parameters for the result quality guarantee of HubPPR and FORA are set to values $(1/n, 1/n, 0.5)$ as suggested in their papers [28], [29]. We obtained the source codes of HubPPR from the authors, which are optimized to compute an approximate RWR score vector. By querying all nodes in a graph as the target nodes, HubPPR computes an approximate RWR score vector.



(a) Slashdot (b) Pokec (c) WikiLink (d) Twitter

Fig. 6: Recall of top-k RWR vertices: except NB-LIN, all methods show similarly high recall on the four datasets. Note that TPA requires less computation time and memory space than other methods, while maintaining the similar recall. Lines are omitted if corresponding experiments run out of memory ($>$ 200GB).

BePI [12], the state-of-the-art exact RWR method, is used to provide the ground truth RWR values in all experiments. We compare the computational efficiency between TPA and BePI in Appendix A.

B. Competitors

Under the environmental settings described above, in the preprocessing phase, we estimate the preprocessing time and the size of preprocessed data of each method. In the online phase, we estimate the computation time and the accuracy of approximate RWR scores computed from each method. From Figures 1 to 6, TPA runs faster than other methods, while requiring less memory space and maintaining high accuracy.

1) Speed

We examine the running time of TPA in the preprocessing phase and the online phase, respectively. Running time is measured in wall-clock time. In the preprocessing phase, TPA computes PageRank using CPI to get $\tilde{\mathbf{r}}_{\text{stranger}}$; BEAR-APPROX precomputes several matrices required in the online phase; NB-LIN computes low-rank approximation and inversion of some small size matrices; HubPPR precomputes and indexes auxiliary information for selected hub nodes that are often involved in RWR processing; FORA precomputes a number of random walks from nodes, and stores the destination of each walk. As shown in Figure 1(b), TPA performs preprocessing faster than other preprocessing methods by up to $3.5\times$. Even though FORA shows relatively fast computation speed in the preprocessing phase, it requires up to $40\times$ larger memory space and up to $30\times$ more online computation time than TPA. Note that the preprocessing phase is executed only once for a graph and the online phase is executed every time for a new seed node. Then the superior performance of TPA for online computation has more significant effects in terms of total computation efficiency. Under 200GB memory capacity, BEAR-APPROX and NB-LIN fail to preprocess the datasets from Pokec and WikiLink, respectively, due to out of memory error. In the online phase, TPA computes an approximate RWR vector up to $30\times$ faster than other methods. Although BEAR-APPROX takes similar online time as TPA in the Google dataset, BEAR-APPROX takes $5923\times$ more preprocessing time than TPA does for the same dataset. On the contrary, TPA maintains superior speed compared to all other methods in both phases.

2) Memory Usage

To compare memory efficiency, we measure how much memory each method requires for the preprocessed data. As shown in Figure 1(a), compared with other preprocessing methods, TPA requires up to $40\times$ less memory space across all the datasets. This result shows the superior scalability of TPA. Under 200GB memory capacity, BEAR-APPROX and NB-LIN consume a significant memory space, thus, are feasible only on the small datasets (LiveJournal, Pokec, Google, and Slashdot). Although HubPPR and FORA succeed in preprocessing billion-scale graphs, they require a significant memory space for the preprocessed data. Note that HubPPR and FORA trade off the online computation time against the size of preprocessed data [28], [29]. Thus, when they manipulate the size of preprocessed data smaller than the memory presented in Figure 1(a), they would require more online computation time than the one presented in Figure 1(c) which is already up to $30\times$ more than TPA.

3) Accuracy

In most applications of RWR, the typical approach is to return the top-k ranked vertices of RWR vector. For instance, in Twitter's "Who to Follow" recommendation service [7], the top-500 ranked users in RWR will be recommended. Therefore, it is important to measure the accuracy of the top-k results to examine the accuracy of an approximate RWR vector. We first calculate the exact top-k vertices using BePI, then evaluate the top-k results of each method by measuring their recall with respect to the exact top-k. For brevity, we show the result on Twitter, WikiLink, Pokec, and Slashdot; results on other graphs are similar. As shown in Figure 6, all methods except NB-LIN provide high recall around 0.99 across all datasets. Note that as shown in Figure 1, TPA requires less computation time and smaller memory space than other methods, while maintaining the similar accuracy.

TABLE III: Error statistics: we measure $L1$ norm errors of the neighbor approximation $\tilde{\mathbf{r}}_{\text{neighbor}}$, the stranger approximation $\tilde{\mathbf{r}}_{\text{stranger}}$, and the final approximate RWR score vector $\mathbf{r}_{\text{TPA}}$ with regard to the exact score vectors $\mathbf{r}_{\text{neighbor}}$, $\mathbf{r}_{\text{stranger}}$, and $\mathbf{r}_{\text{CPI}}$, respectively. Then we compare the $L1$ norm errors with their theoretical error bounds, respectively. The theoretical error bounds for $\tilde{\mathbf{r}}_{\text{neighbor}}$, $\tilde{\mathbf{r}}_{\text{stranger}}$ and $\mathbf{r}_{\text{TPA}}$ are $2(1-c)^S - 2(1-c)^T$, $2(1-c)^T$, and $2(1-c)^S$, respectively. Percentage denotes the ratio of $L1$ norm error in real-world graphs with regard to the theoretical error bound. S denotes the starting iteration of the neighbor approximation and T denotes the starting iteration of the stranger approximation. Both neighbor approximation and stranger approximation lower errors significantly from their theoretical error bounds by exploiting characteristics of real-world graphs.

Dataset	Neighbor Approximation			Stranger Approximation			TPA		
	Theoretical bound: (A)	**Actual error: (B)**	**Percentage: (B/A)**	**Theoretical bound: (A)**	**Actual error: (B)**	**Percentage: (B/A)**	**Theoretical bound: (A)**	**Actual error: (B)**	**Percentage: (B/A)**
Slashdot	0.7127	0.3367	47.24%	0.1747	0.0861	49.27%	0.8874	0.0505	5.69%
Google	0.8099	0.3377	41.70%	0.0775	0.0451	58.19%	0.8874	0.1805	20.33%
Pokec	0.4937	0.3041	61.59%	0.3937	0.1011	25.68%	0.8874	0.1946	21.93%
LiveJournal	0.4937	0.2711	54.91%	0.3937	0.1456	36.98%	0.8874	0.2555	28.79%
WikiLink	0.1331	0.0739	55.51%	0.7543	0.2097	27.80%	0.8874	0.2370	26.71%
Twitter	0.2897	0.1953	67.43%	0.7543	0.0349	4.63%	1.0440	0.1015	9.73%
Friendster	0.9665	0.4479	46.34%	0.0775	0.0419	54.06%	1.0440	0.0675	6.46%

C. TPA *in Real-world Graphs*

In Section III, we analyze the error bounds of the neighbor approximation and the stranger approximation theoretically and elaborate how the approximations achieve lower errors than the theoretical bounds in real-world graphs. The stranger approximation uses the increased density of adjacency matrices of real-world graphs as the matrices are raised to the ith power. With the help of block-wise structure of real-world graphs, the neighbor approximation results in low error in practice. In Table III, we compare the errors of the neighbor approximation and the stranger approximation in real-world graphs with their theoretical bounds, respectively. S and T used in each dataset are described in Table II. The neighbor approximation lowers the error up to 42% and the stranger approximation lowers the error up to 5% from their theoretical error bounds, respectively. This results show that both approximations exploit the characteristics of real-world graphs effectively. One interesting point is that the total error of TPA is significantly lower than the sum of errors of the neighbor approximation and the stranger approximation. E.g., in the Slashdot dataset, the neighbor and stranger approximations lower errors to the half of the suggested theoretical bounds, but the total experimental error of TPA decreases to 6% of its theoretical upper bound. This presents the stranger approximation and the neighbor approximation complement each other effectively. The neighbor approximation could not consider nodes which are not visited in the family iterations since it approximates the neighbor iterations only based on the family iterations. On the other hand, the stranger approximation could not consider the effect of seed node since it precomputes PageRank in the preprocessing phase without any information about which node would be a seed node. Merged with the stranger approximation, the neighbor approximation acquires information about nodes across whole graphs, stored in PageRank. On the other hand, merged with the neighbor approximation, the stranger approximation has a chance to put more priority on the seed node. TPA compensates the weak points of each approximations successfully.

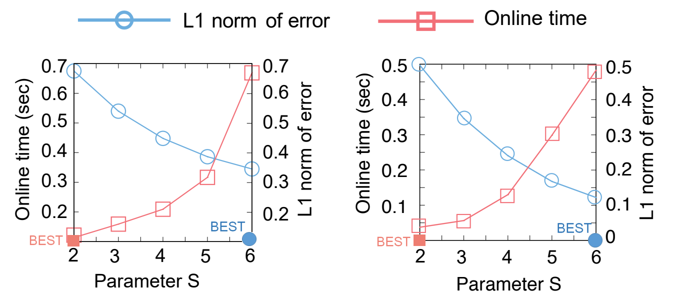
D. *Effects of Parameters*

We discuss the effects of two parameters S and T in this subsection. We first investigate the effects of S, the starting iteration of the neighbor approximation, on the performance of TPA. We measure online computation time and $L1$ norm error of TPA varying S. During this experiment, T is fixed to 10. As shown in Figure 7, as S increases, online time increases sharply while $L1$ norm error decreases since a portion of the exact computation $\mathbf{r}_{\text{family}}$ increases. Thus, S is selected to a proper number considering the tradeoff between accuracy and running time of TPA.

Next, we examine the effects of T, the starting iteration of the stranger approximation, on the accuracy of TPA. We measure $L1$ norm errors of the neighbor approximation, the stranger approximation, and TPA, respectively, varying T. Note that S is fixed to 5 during this experiment. In Figure 8, as T increases, $L1$ error of the neighbor approximation increases, that of the stranger approximation decreases, and that of TPA decreases at first and then rebounds from $T = 10$. With small T, the stranger approximation applies to nodes close to the seed, then, the nodes are estimated by their PageRank scores and the effects of their close distances from the seed are ignored. This leads to high $L1$ norm error of the stranger approximation. On the other hand, with large T, the neighbor approximation applies to nodes far from the seed. The nodes which reside far from the seed are likely to belong to different communities from that of the seed. However, by the neighbor approximation, such nodes are estimated as the same community members as the seed. Then, $L1$ norm error of the neighbor approximation becomes high. Thus T is set to a value which minimizes the total $L1$ norm error of TPA.

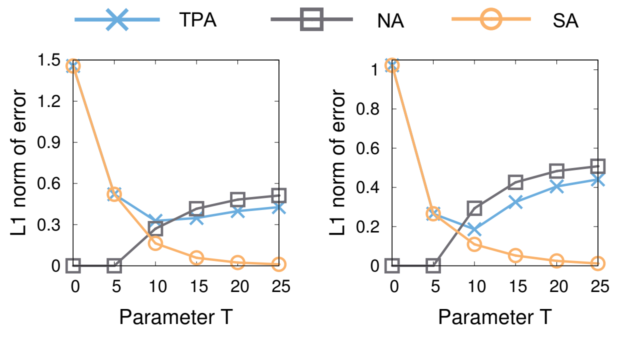
V. Related Works

In this section, we review previous approximate methods for RWR. To avoid enormous costs incurred by RWR computation, many efforts have been devoted to estimating RWR in a cost-efficient way while sacrificing little accuracy. Gleich et al. [6] introduced boundary restricted personalized PageRank



(a) Online time vs. L1 norm of error on the LiveJournal dataset (b) Online time vs. L1 norm of error on the Pokec dataset

Fig. 7: Effects of S: with small S, TPA runs fast with high $L1$ norm error. On the other hand, with large S, TPA takes long computation time with low $L1$ error.



(a) L1 norm of error on the LiveJournal dataset (b) L1 norm of error on the Pokec dataset

Fig. 8: Effects of T: as T increases, $L1$ norm error of the neighbor approximation (NA) increases, that of the stranger approximation (SA) decreases, and that of TPA decreases at first and then rebounds from $T = 10$.

(BRPPR) which improves speed by limiting the amount of graph data that need to be accessed. BRPPR iteratively divides the vertices of a graph into an active and an inactive set. At each iteration, the set of active vertices is expanded to include more vertices that are likely to have a high RWR score. BRPPR expands nodes until the total rank on the frontier set of nodes is less than κ. Proposed by Tong et al. [27], NB-LIN exploits linear correlations across rows and columns of the adjacency matrix in many real-world graphs. NB-LIN computes low-rank approximation of the adjacency matrix and uses it to estimate RWR score vector based on the Sherman-Morrison lemma. NB-LIN divides whole computation into the preprocessing phase and online phase, and yields faster response time in the online phase. Shin et al. extended their exact RWR method BEAR [13], [24] to an approximate RWR method BEAR-APPROX which drops non-zero entries whose absolute value is smaller than the drop tolerance in its preprocessed matrix. Forward Push [1] computes RWR by propagating residuals across a graph until all the residuals become smaller than a given threshold. Proposed by Wang et al. [29], FORA first performs Forward Push with early termination, and subsequently runs random walks. FORA utilizes Forward Push to significantly cut down the number of required random walks while satisfying the same result quality guarantees of random walks. FORA precomputes a number of random walks in the preprocessing phase to further improve computation efficiency. Other methods such as FAST-PPR [20], BiPPR [19] and HubPPR [28] narrow down the scope of RWR problem by specifying a target node. BiPPR processes an RWR query through a bi-directional search on the input graph. HubPPR precomputes indexes in the preprocessing phase and approximates RWR with the help of precomputed indexes in the bi-directional way. We compare our method with HubPPR since HubPPR is the most recent study with the best performance among bi-directional methods [28]. Our proposed TPA outperforms all methods described above by providing a better cost-efficiency.

VI. Conclusion

In this paper, we propose TPA, a fast and accurate method for computing approximate RWR. TPA is based on cumulative power iteration (CPI) which interprets RWR problem as propagation of scores from a seed node across a graph. To avoid long computation time, TPA divides the whole iterations of CPI into three parts (family, neighbor, and stranger parts), and estimates the neighbor part and the stranger part using our proposed approximation methods called neighbor approximation and stranger approximation, respectively. With the help of two approximation phases, TPA quickly computes only the family part in the online phase, and then approximates RWR with high accuracy. Our evaluation shows that TPA outperforms other state-of-the-art methods in terms of speed and memory-efficiency, without sacrificing accuracy. Future works include extending TPA into a disk-based RWR method to handle huge, disk-resident graphs.

Acknowledgment

This work was supported by ICT R&D program of MSIP/IITP. [2013-0-00179, Development of Core Technology for Context-aware Deep-Symbolic Hybrid Learning and Construction of Language Resources]. The ICT at Seoul National University provides research facilities for this study. The Institute of Engineering Research at Seoul National University provided research facilities for this work. U Kang is the corresponding author.

References

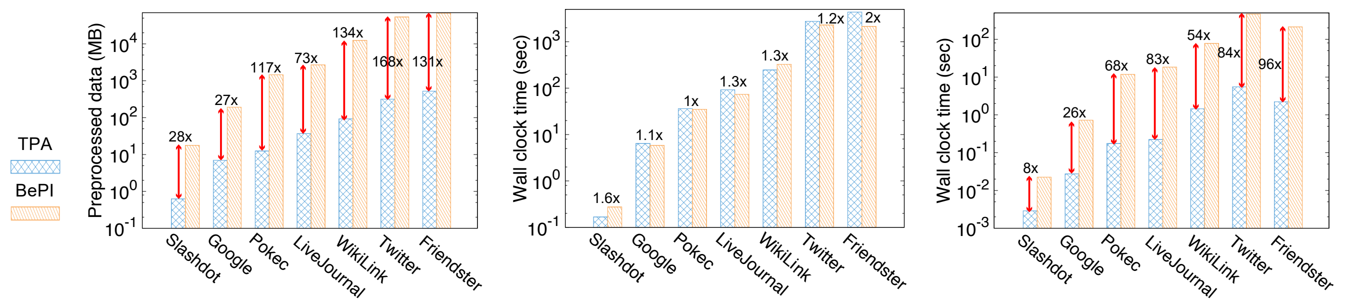
[1] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 475–486. IEEE, 2006.

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

[3] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644. ACM, 2011.

[4] S. Chakrabarti, A. Pathak, and M. Gupta. Index design and query processing for graph conductance search. *The VLDB Journal*, 20(3):445–470, 2011.

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



(a) Size of preprocessed data (b) Preprocessing time (c) Online time

Fig. 9: Comparison with BePI: (a) TPA uses less amount of space for preprocessed data than BePI does across all datasets. (b) In the preprocessing phase, TPA and BePI take similar computation time. (c) In the online phase, TPA computes RWR scores faster than BePI does over all datasets. Note that TPA computes the approximate RWR scores while BePI results in the exact RWR scores.

[6] D. Gleich and M. Polito. Approximating personalized pagerank with minimal use of web graph data. *Internet Mathematics*, 3(3):257–294, 2006.

[7] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*, pages 505–514. ACM, 2013.

[8] J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 9–16. ACM, 2004.

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

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

[11] J. Jung, W. Jin, L. Sael, and U. Kang. 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*, pages 973–978, 2016.

[12] J. Jung, N. Park, L. Sael, and U. Kang. Bepi: Fast and memory-efficient method for billion-scale random walk with restart. In *SIGMOD*, 2017.

[13] J. Jung, K. Shin, L. Sael, and U. Kang. Random walk with restart on large graphs using block elimination. *ACM Trans. Database Syst.*, 41(2):12, 2016.

[14] U. Kang, M. Bilenko, D. Zhou, and C. Faloutsos. Axiomatic analysis of co-occurrence similarity functions. *CMU-CS-12-102*, 2012.

[15] U. Kang and C. Faloutsos. Beyond 'caveman communities': Hubs and spokes for graph compression and mining. In *ICDM*, 2011.

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

[17] A. N. Langville and C. D. Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2011.

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

[19] P. Lofgren, S. Banerjee, and A. Goel. Personalized pagerank estimation and search: A bidirectional approach. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 163–172. ACM, 2016.

[20] P. A. Lofgren, S. Banerjee, A. Goel, and C. Seshadhri. Fast-ppr: Scaling personalized pagerank estimation for large graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1436–1445. ACM, 2014.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

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

[23] H. Park, J. Jung, and U. Kang. A comparative study of matrix factorization and random walk with restart in recommender systems. In *BigData*, 2017.

[24] K. Shin, J. Jung, S. Lee, and U. Kang. 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*, pages 1571–1585. ACM, 2015.

[25] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

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

[27] H. Tong, C. Faloutsos, and J.-Y. Pan. Random walk with restart: fast solutions and applications. *Knowledge and Information Systems*, 14(3):327–346, 2008.

[28] S. Wang, Y. Tang, X. Xiao, Y. Yang, and Z. Li. Hubppr: effective indexing for approximate personalized pagerank. *Proceedings of the VLDB Endowment*, 10(3):205–216, 2016.

[29] S. Wang, R. Yang, X. Xiao, Z. Wei, and Y. Yang. Fora: Simple and effective approximate single-source personalized pagerank. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 505–514. ACM, 2017.

[30] J. J. Whang, D. F. Gleich, and I. S. Dhillon. Overlapping community detection using seed set expansion. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2099–2108. ACM, 2013.

[31] Z. A. Zhu, S. Lattanzi, and V. S. Mirrokni. A local algorithm for finding well-connected clusters. In *ICML (3)*, pages 396–404, 2013.

Appendix

A. *Comparison with BePI*

BePI [12] is the state-of-the-art exact RWR method which precomputes several matrices required by the online phase in the preprocessing phase and computes RWR scores by exploiting the precomputed matrices in the online phase. As shown in Figure 9, TPA and BePI take the similar preprocessing time, while TPA is up to 96$\times$ faster than BePI in the online phase. Considering that the preprocessing phase is executed only once for a graph and the online phase is executed everytime for a new seed node, the superior performance of TPA for online computation brings significant advantages for users who put more priority on speed than accuracy. Moreover, TPA requires up to 168$\times$ less memory space for preprocessed data than BePI. Note that while TPA outperforms BePI in terms of computation time and memory usage, TPA computes the approximate RWR scores and BePI results in the exact RWR scores.