# Efficient Estimation of Heat Kernel PageRank for Local Clustering

Renchi Yang
Nanyang Technological University
yang0461@e.ntu.edu.sg

Xiaokui Xiao
National University of Singapore
xkxiao@nus.edu.sg

Zhewei Wei[*]
Renmin University of China
zhewei@ruc.edu.cn

Sourav S Bhowmick
Nanyang Technological University
assourav@ntu.edu.sg

Jun Zhao
Nanyang Technological University
junzhao@ntu.edu.sg

Rong-Hua Li
Beijing Institute of Technology
lironghuascut@gmail.com

## ABSTRACT

Given an undirected graph $G$ and a *seed* node $s$, the *local clustering* problem aims to identify a high-quality cluster containing $s$ in time roughly proportional to the size of the cluster, regardless of the size of $G$. This problem finds numerous applications on large-scale graphs. Recently, *heat kernel PageRank* (HKPR), which is a measure of the proximity of nodes in graphs, is applied to this problem and found to be more efficient compared with prior methods. However, existing solutions for computing HKPR either are prohibitively expensive or provide unsatisfactory error approximation on HKPR values, rendering them impractical especially on billion-edge graphs.

In this paper, we present TEA and TEA+, two novel local graph clustering algorithms based on HKPR, to address the aforementioned limitations. Specifically, these algorithms provide non-trivial theoretical guarantees in *relative error* of HKPR values and the time complexity. The basic idea is to utilize deterministic graph traversal to produce a rough estimation of exact HKPR vector, and then exploit Monte-Carlo random walks to refine the results in an optimized and non-trivial way. In particular, TEA+ offers practical efficiency and effectiveness due to non-trivial optimizations. Extensive experiments on real-world datasets demonstrate that TEA+ outperforms the state-of-the-art algorithm by more than four times on most benchmark datasets in terms of computational time when achieving the same clustering quality, and in particular, is an order of magnitude faster on large graphs including the widely studied *Twitter* and *Friendster* datasets.

## CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**.

## KEYWORDS

heat kernel PageRank; local clustering

**ACM Reference Format:**
Renchi Yang, Xiaokui Xiao, Zhewei Wei, Sourav S Bhowmick, Jun Zhao, and Rong-Hua Li. 2019. Efficient Estimation of Heat Kernel PageRank for Local Clustering. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3299869.3319886

## 1 INTRODUCTION

Graph clustering is a fundamental problem that finds numerous applications, *e.g.,* community detection [24, 30, 35], image segmentation [23, 29], and protein grouping in biological networks [18, 40]. The problem has been studied extensively in the literature, and yet, clustering massive graphs remains a challenge in terms of computation efficiency. This motivates a series of algorithms [3–5, 11, 17, 21, 26, 31, 36, 38, 39, 43] for *local clustering*, which takes as input an undirected graph $G$ and a *seed* node $s$, and identifies a cluster (*i.e.,* a set of nodes) containing $s$ in time depending on the size of the cluster, regardless of the size of $G$.

Local clustering algorithms have the potential to underpin interactive exploration of massive graphs. Specifically, they can facilitate exploration of a relatively small region of a large graph that is of interest to a user. For example, consider Bob, a budding entrepreneur, who is interested in exploring the local clusters of visionary entrepreneurs in *Twitter*. Particularly, he wishes to begin his exploration with the cluster associated with Elon Musk (*i.e.,* seed). Since Bob thinks that Elon Musk is an inspirational entrepreneur, he would like

[*] Work partially done at Beijing Key Laboratory of Big Data Management and Analysis Methods.

*SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands*
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3319886

to explore if there are any other notable entrepreneurs (e.g., Kevin Rose) in Elon's local cluster (*e.g.,* followers, followees) and wishes to further explore the local neighborhoods of these entrepreneurs. In order to ensure a palatable and non-disruptive interactive experience, Bob needs an efficient local clustering framework that can return high quality clusters within few seconds. *Which existing local clustering framework can Bob utilize for his exploration?*

Spielman and Teng [21, 38] are the first to study the local clustering problem, and they propose a random-walk-based algorithm, Nibble, that optimizes the *conductance* [7] of the cluster returned. Specifically, the conductance of cluster $S$ is defined as $\Phi(S) = \frac{|\mathrm{cut}(S)|}{\min\{\mathrm{vol}(S), 2m - \mathrm{vol}(S)\}}$, where vol$(S)$ is the sum of the degrees of all nodes in $S$, $m$ is the number of edges in the graph $G$, and $|\mathrm{cut}(C)|$ is number of edges with one endpoint in $S$ and another not in $S$. Intuitively, if a cluster $C$ has a small conductance, then the nodes in $S$ are better connected to each other than to the nodes apart from $S$, in which case $S$ should be considered a good cluster. This algorithm is subsequently improved in a series of work [3, 4, 11, 26, 31, 36, 38, 39, 43] that aims to either improve the efficiency of local clustering or reduce the conductance of the cluster returned.

The state-of-the-art solutions [11, 17] for local clustering are based on *heat kernel PageRank* (HKPR) [8], which is a measure of the proximity of nodes in graphs. Given a seed node $s$, these solutions first compute a vector $\widehat{\boldsymbol{\rho}}_s$ where each element $\widehat{\boldsymbol{\rho}}_s[v]$ approximates the HKPR value of a node $v$ with respect to $s$ (i.e., $\widehat{\boldsymbol{\rho}}_s[v]$ approximately measures the proximity of $s$ to $v$). Then, they utilize $\widehat{\boldsymbol{\rho}}_s$ to derive a local cluster $C$ containing $s$. It is shown that the quality of $S$ depends on the accuracy of $\widehat{\boldsymbol{\rho}}_s$ [12, 17], in the sense that the conductance of $S$ tends to decrease with the approximation error in $\widehat{\boldsymbol{\rho}}_s$. Therefore, existing HKPR-based solutions [11, 17] all focus on striking a good trade-off between time efficiency and the accuracy of $\widehat{\boldsymbol{\rho}}_s$. In particular, the current best solution HK-Relax [17] ensures that (i) $\frac{1}{d(v)}\left|\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v]\right| < \epsilon_a$ for any node $v$, where $\epsilon_a$ is a given threshold, $d(v)$ is the degree of node $v$ and $\boldsymbol{\rho}_s[v]$ is the exact HKPR value of node $v$ with respect to $s$, and (ii) $\widehat{\boldsymbol{\rho}}_s$ is computed in $O\left(\frac{te^t \log(1/\epsilon_a)}{\epsilon_a}\right)$ time, where $t$ is constant (referred to as the *heat constant*) used in the definition of HKPR.

**Motivation.** The time complexity of HK-Relax has a large factor $e^t$, where $t$ (i.e., the heat constant) could be as large as a few dozens [11, 17, 22]. Consequently, it can be inefficient for several applications. For instance, reconsider Bob's endeavor to explore the local clusters of Elon Musk and Kevin Rose. HK-Relax consumes around 15s and 48s, respectively, to compute their local clusters. Such performance is disruptive for any interactive graph exploration. In addition, HK-Relax provides an accuracy guarantee on each $\frac{\widehat{\rho}_s[v]}{d(v)}$ in terms of its *absolute error*, but as we discuss in Section 3, this guarantee is less than ideal for accurate local clustering. Specifically, HKPR-based local clustering requires ranking each node $v$ in descending order of $\frac{\widehat{\rho}_s[v]}{d(v)}$, which we refer to as $v$'s *normalized HKPR.* To optimize this accuracy of this ranking, we observe that it is more effective to minimize the *relative errors* of normalized HKPR values than their absolute errors. To explain, we note that the normalized HKPR varies significantly from nodes to nodes. For the aforementioned ranking, nodes with large normalized HKPR could tolerate more absolute errors than nodes with small normalized HKPR, and hence, imposing the same absolute error guarantees on all nodes tend to produce sub-optimal results.

**Our contributions.** Motivated by the deficiency of existing solutions, we present an in-depth study on HKPR-based local clustering, and make the following contributions. First, we formalize the problem of approximate HKPR computation with probabilistic relative error guarantees, and pinpoint why none of the existing techniques could provide an efficient solution to this problem.

Second, based on our problem formulation, we propose two new algorithms, TEA and TEA+, both of which (i) take as input a seed node $s$, two thresholds $\epsilon_r, \delta$, and a failure probability $p_f$, and (ii) return an approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$ where each element $\widehat{\boldsymbol{\rho}}_s[v]$ with $\frac{\rho_s[v]}{d(v)} > \delta$ has at most $\epsilon_r$ relative error with at least $1 - p_f$ probability (i.e., all *significant* HKPR values are accurately approximated with high probability). The core of TEA is an adaptive method that combines deterministic graph traversal with random walks to estimate normalized HKPR in a cost-effective manner, while TEA+ significantly improves over TEA in terms of practical efficiency by incorporating a number of non-trivial optimization techniques. Both algorithms have a time complexity of $O\left(\frac{t\log(n/p_f)}{\epsilon_r^2 \cdot \delta}\right)$, which eliminates the exponential term $e^t$ in HK-Relax's efficiency bound (see Table 1).

Third, we experimentally evaluate them against the state of the art, using large datasets with up to 65 million nodes and 1.8 billion edges. Our results show that TEA+ is up to an order of magnitude faster than competing methods when achieving the same clustering quality. In particular, it can compute the local clusters of Elon Musk and Kevin Rose within 1.3s and 6.1s, respectively, thereby facilitating interactive exploration.

**Paper Organization.** The rest of the paper is organized as follows. In Section 2, we introduce background on heat kernel-based local clustering. An overview of our solution framework is presented in Section 3. We present TEA and TEA+ in Sections 4 and 5, respectively. Related work is reviewed in Section 6. We evaluate the practical efficiency of our algorithms against the competitors in Section 7. Finally,

**Table 1: Theoretical guarantee of our solution against that of the state-of-the-art solutions.**

| Algorithm | Accuracy Guarantee | Time Complexity |
|---|---|---|
| ClusterHKPR [11] | with probability at least $1-\epsilon$, $\begin{cases} \lvert\widehat{\boldsymbol{\rho}}_s[v]-\boldsymbol{\rho}_s[v]\rvert\rvert \le \epsilon \cdot \boldsymbol{\rho}_s[v], & \text{if } \boldsymbol{\rho}_s[v] > \epsilon \\ \lvert\widehat{\boldsymbol{\rho}}_s[v]-\boldsymbol{\rho}_s[v]\rvert\rvert \le \epsilon, & \text{otherwise,} \end{cases}$ | $O\left(\frac{t\log(n)}{\epsilon^3}\right)$ |
| HK-Relax [17] | $\frac{1}{d(v)}\left\lvert\widehat{\boldsymbol{\rho}}_s[v]-\boldsymbol{\rho}_s[v]\right\rvert < \epsilon_a$ | $O\left(\frac{te^t\log(1/\epsilon_a)}{\epsilon_a}\right)$ |
| Our solutions | with probability at least $1-p_f$, $\begin{cases} \frac{1}{d(v)}\left\lvert\widehat{\boldsymbol{\rho}}_s[v]-\boldsymbol{\rho}_s[v]\right\rvert \le \epsilon_r \cdot \frac{\boldsymbol{\rho}_s[v]}{d(v)}, & \text{if } \frac{\boldsymbol{\rho}_s[v]}{d(v)} > \delta \\ \frac{1}{d(v)}\left\lvert\widehat{\boldsymbol{\rho}}_s[v]-\boldsymbol{\rho}_s[v]\right\rvert \le \epsilon_r \cdot \delta, & \text{otherwise,} \end{cases}$ | $O\left(\frac{t\log(n/p_f)}{\epsilon_r^2\cdot\delta}\right)$ |

**Table 2: Frequently used notations.**

| Notation | Description |
|---|---|
| $G=(V,E)$ | An undirected graph with node set $V$ and edge set $E$ |
| $n, m$ | The numbers of nodes and edges in $G$, respectively |
| $N(v)$ | The set of neighbors of node $v$ |
| $d(v)$ | The degree of node $v$ |
| $\bar{d}$ | The average degree of the graph, i.e., $\frac{2m}{n}$ |
| $\mathbf{A}, \mathbf{D}, \mathbf{P}$ | The adjacency, diagonal degree, and transition matrices of $G$ |
| $t$ | The heat constant of HKPR |
| $\eta(k), \psi(k)$ | See Equation (1) and Equation (3), respectively |
| $\boldsymbol{\rho}_s[v]$ | HKPR of $v$ w.r.t. $s$, defined by Equation (2) |
| $\epsilon_r, \delta, p_f$ | Parameters of an approximate HKPR, as in Section 3 |
| $\mathbf{r}_s^{(k)}[v]$ | The $k$-hop residue of $v$ during performing push operations from $s$ |
| $\mathbf{q}_s[v]$ | The reserve of $v$ during performing push operations from $s$ |
| $K$ | The maximum number of hops during performing push operations from the seed node |

Section 8 concludes the paper. Proofs of theorems and lemmas are given in Appendix A and [1], respectively. Table 2 lists the notations that are frequently used in our paper.

# 2 PRELIMINARIES

## 2.1 Basic Terminology

Let $G = (V, E)$ be an undirected and unweighted graph, where $V$ and $E$ denote the node and edge sets, respectively. We use $d(v)$ to denote the degree of node $v$, and $\mathbf{A}$ to denote the adjacency matrix of $G$; i.e., $\mathbf{A}[i,j] = \mathbf{A}[j,i] = 1$ if and only if $(v_i, v_j) \in E$. Let $\mathbf{D}$ be the *diagonal degree matrix* of $G$, where $\mathbf{D}[i,i] = d(v_i)$. Then, the *probability transition matrix* (a.k.a. *random walk transition matrix*) for $G$ is defined as $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$. Accordingly, $\mathbf{P}^k[s,v]$ denotes the probability that a $k$-hop ($k \ge 1$) random walk from node $s$ would end at $v$.

A *cluster* in $G$ is a node set $S \subseteq V$. Intuitively, a good cluster should be both internally cohesive and well separated from the remainder of $G$. We say that $S$ is a high-quality cluster if it has a small *conductance* [7] $\Phi(S)$, defined as:

$$\Phi(S) = \frac{|\mathrm{cut}(S)|}{\min\{(\mathrm{vol}(S), \mathrm{vol}(V \setminus S))\}},$$

where $\mathrm{vol}(S)$ is the *volume* of $S$, namely, the sum of the degrees of all nodes in $S$, and $\mathrm{cut}(S)$ is the *cut* of $S$, i.e., the set of edges with one endpoint in $S$ and another not in $S$.

## 2.2 Heat Kernel-based Local Clustering

Given a *heat constant* $t$ and two nodes $u$ and $v$, the HKPR value from $u$ to $v$ is defined as the probability that a random walk of length $k$ starting from $u$ would end at $v$, where $k$ is sampled from the following Poisson distribution:

$$\eta(k) = \frac{e^{-t}t^k}{k!}. \tag{1}$$

Let $s$ be the seed node for local clustering. We define the HKPR vector $\boldsymbol{\rho}_s$ of $s$ as an $n$-size vector, such that the $i$-th element of $\boldsymbol{\rho}_s$ equals the HKPR value from $s$ to the $i$-th node in $G$. In addition, we use $\boldsymbol{\rho}_s[v]$ to denote the HKPR value from $s$ to $v$, which is defined by

$$\boldsymbol{\rho}_s[v] = \sum_{k=0}^{\infty} \eta(k) \cdot \mathbf{P}^k[s,v]. \tag{2}$$

Existing heat-kernel-based algorithms [11, 13, 17, 22] all adopt a two-phase approach. In particular, they first compute an *approximate* HKPR vector $\widehat{\boldsymbol{\rho}}_s$ for $s$, and then perform a *sweep* as follows:

(1) Take the set $S^*$ of nodes with non-zero values in $\widehat{\boldsymbol{\rho}}_s$.

(2) Sort the nodes $v \in S^*$ in descending order of $\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)}$. Let $S_i^*$ be a set containing the first $i$ nodes in the sorted sequence.

(3) Inspect $S_i^*$ in ascending order of $i$. Return the set $S_i^*$ with the smallest conductance among the ones that have been inspected.

It is shown in [22, 43] that the above sweep can be conducted in $O(|S^*| \cdot \log |S^*|)$ time, assuming that $\widehat{\boldsymbol{\rho}}_s$ is given in a sparse representation with $O(|S^*|)$ entries. In contrast, the computation of $\widehat{\boldsymbol{\rho}}_s$ is much more costly, and hence, has been the main subject of research in existing work [11, 13, 17, 22].

# 3 SOLUTION OVERVIEW

Our solution for local clustering is based on heat kernel PageRank, and it follows the same two-phase framework in the existing work [8, 9, 11, 13, 17, 22]. That is, we also compute an approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$ for $s$, and then conduct a sweep on $\widehat{\boldsymbol{\rho}}_s$. However, we require that $\widehat{\boldsymbol{\rho}}_s$ should be a $(d, \epsilon_r, \delta)$*-approximate HKPR vector*, which is a criterion not considered in any existing work [8, 9, 11, 13, 17, 22].

Definition 1. *($(d, \epsilon_r, \delta)$-approximate HKPR) Let $\boldsymbol{\rho}_s$ be the HKPR vector for a seed node $s$, and $\widehat{\boldsymbol{\rho}}_s$ be an approximated version of $\boldsymbol{\rho}_s$. $\widehat{\boldsymbol{\rho}}_s$ is $(d, \epsilon_r, \delta)$-approximate if it satisfies the following conditions:*

- *For every $v \in V$ with $\frac{\boldsymbol{\rho}_s[v]}{d(v)} > \delta$,*

$$\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \leq \epsilon_r \cdot \frac{\boldsymbol{\rho}_s[v]}{d(v)};$$

- *For every $v \in V$ with $\frac{\boldsymbol{\rho}_s[v]}{d(v)} \leq \delta$,*

$$\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \leq \epsilon_r \cdot \delta. \qquad \square$$

In other words, we require $\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)}$ to provide a relative error guarantee when $\frac{\boldsymbol{\rho}_s[v]}{d(v)} > \delta$, and an absolute error guarantee when $\frac{\boldsymbol{\rho}_s[v]}{d(v)} \leq \delta$. This is to ensure that when we sort the nodes in descending order or $\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)}$ (which is a crucial step in the sweep for local clustering), the sorted sequence would be close to the one generated based on $\frac{\boldsymbol{\rho}_s[v]}{d(v)}$. We do not consider relative error guarantees when $\frac{\boldsymbol{\rho}_s[v]}{d(v)} \leq \delta$, because (i) ensuring a small relative error for such a node $v$ requires an extremely accurate estimation of its normalized HKPR, which would incur significant computation overheads, and (ii) providing such high accuracy for $v$ is unnecessary, since $v$'s tiny normalized HKPR value indicates that it is not relevant to the result of local clustering.

By the definition of HKPR (in Equation (2)), the HKPR value of $v$ w.r.t. $s$ is a weighted sum of $k$-hop random walk transition probabilities from $s$ to $v$, where $k$ is a Poisson distributed length. Thus, a straightforward method to compute $(d, \epsilon_r, \delta)$-approximate HKPR for seed node $s$ is to conduct Monte-Carlo simulations using a large number of random walks. Specifically, each random walk should start from $s$, and should have a length that is sampled from the Poisson distribution in Equation (1). Let $n_r$ be the total number of random walks, and $\widehat{\boldsymbol{\rho}}_s[v]$ be the fraction of walks that end at a node $v$. Then, we can use $\widehat{\boldsymbol{\rho}}_s[v]$ as an estimation of $\boldsymbol{\rho}_s[v]$. By the Chernoff bound (see Lemma 5) and union bound, it can be shown that $\widehat{\boldsymbol{\rho}}_s$ is $(d, \epsilon_r, \delta)$-approximate with probability at least

$$1 - n \cdot \exp\left(-\frac{n_r \cdot \epsilon_r^2 \cdot \delta}{2(1 + \epsilon_r/3)}\right).$$

Therefore, if we are to ensure that the above probability is at least $1 - p_f$, then we can set $n_r = \frac{2(1+\epsilon_r/3)\log(n/p_f)}{\epsilon_r^2 \cdot \delta}$. In that case, the time required to generate the random walks is $O\left(\frac{t \log(n/p_f)}{\epsilon_r^2 \cdot \delta}\right)$. The main issue of this Monte-Carlo approach, however, is that it incurs considerable overheads in practice (see our experimental results in Section 7.4). To explain, consider a node $v$ with a small $\boldsymbol{\rho}_s[v]$ but a relatively large $\frac{\boldsymbol{\rho}_s[v]}{d(v)} > \delta$. By the requirements of $(d, \epsilon_r, \delta)$-approximation, we need to ensure that $|\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v]| \leq \epsilon_r \cdot \boldsymbol{\rho}_s[v]$. In turn, this requires that the number $n_r$ of random walks should be large; otherwise, the number of walks that end at $v$ would be rather small, in which case the estimation of $\boldsymbol{\rho}_s[v]$ would be inaccurate.

We also note that none of the existing methods [11, 17] can be adopted to compute $(d, \epsilon_r, \delta)$-approximate HKPR efficiently. In particular, HK-Relax [17] only ensures that for any node $v \in V$, $\frac{1}{d(v)}\left|\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v]\right| < \epsilon_a$. If we are to use HK-Relax for $(d, \epsilon_r, \delta)$-approximation, then we need to set $\epsilon_a = \epsilon_r \cdot \delta$, in which case its complexity would be $O\left(\frac{te^t \log\left(\frac{1}{\epsilon_r \cdot \delta}\right)}{\epsilon_r \cdot \delta}\right)$, which is considerably worse than the time complexity of the Monte-Carlo approach, due to the exponential term $e^t$. The ClusterHKPR [11] algorithm suffers from a similar issue, as we point out in Section 6.

To mitigate the deficiencies of the aforementioned methods, we present (in Section 4 and Section 5) two more efficient HKPR algorithms that satisfy the following requirements:

(1) Return a $(d, \epsilon_r, \delta)$-approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$ with at least $1 - p_f$ probability, where $p_f$ is a user-specified parameter;
(2) Run in $O\left(\frac{t \log(n/p_f)}{\epsilon_r^2 \cdot \delta}\right)$ expected time.

## 4 THE TEA ALGORITHM

This section presents TEA[1], our first-cut solution for $(d, \epsilon_r, \delta)$-approximate HKPR. TEA is motivated by the inefficiency of the Monte-Carlo approach explained in Section 3, i.e., it requires a large number of random walks to accurately estimate HKPR values. To address this issue, we propose to combine Monte-Carlo with a secondary algorithm that could help reduce the number of random walks needed. In particular, we first utilize the secondary algorithm to efficiently compute a rough estimation $\mathbf{q}_s[v]$ of $\boldsymbol{\rho}_s[v]$, and then perform random walks to refine each $\mathbf{q}_s[v]$, so as to transform $\mathbf{q}_s$ into a $(d, \epsilon_r, \delta)$-approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$. Towards this end, there are several issues that we need to address:

(1) How to design a secondary algorithm that could generate a rough approximation of the HKPR vector at a small computation cost?
(2) How to enable Monte-Carlo to leverage the output of the secondary algorithm for improved efficiency?
(3) How to ensure that the combination of Monte-Carlo and the secondary algorithm still provides strong theoretical assurance in terms of time complexity and accuracy?

To answer the above questions, we first present our choice of the secondary algorithm, referred to as HK-Push, in Section 4.1; after that, we elaborate the integration of HK-Push and Monte-Carlo in Section 4.2, and then provide a theoretical analysis of the combined algorithm in Section 4.3.

[1] Two-Phase Heat Kernel Approximation

**Algorithm 1:** HK-Push

```
Input: Graph $G$, seed node $s$, threshold $r_{max}$
Output: An approximate HKPR vector $\mathbf{q}_s$ and $K+1$
        residue vectors $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$
1 $\mathbf{q}_s \leftarrow \mathbf{0}$, $\mathbf{r}_s^{(k)} \leftarrow \mathbf{0}$ for $k = 0, \ldots$;
2 $\mathbf{r}_s^{(0)}[s] \leftarrow 1$;
3 while $\exists v \in V, k$ such that $\mathbf{r}_s^{(k)}[v] > r_{max} \cdot d(v)$ do
4     $\mathbf{q}_s[v] \leftarrow \mathbf{q}_s[v] + \frac{\eta(k)}{\psi(k)} \cdot \mathbf{r}_s^{(k)}[v]$;
5     for $u \in N(v)$ do
6         $\mathbf{r}_s^{(k+1)}[u] \leftarrow \mathbf{r}_s^{(k+1)}[u] + \left(1 - \frac{\eta(k)}{\psi(k)}\right) \cdot \frac{\mathbf{r}_s^{(k)}[v]}{d(v)}$;
7     $\mathbf{r}_s^{(k)}[v] \leftarrow 0$;
8 $K \leftarrow \max\left\{k \,\middle|\, \exists v \in V, \mathbf{r}_s^{(k)}[v] > 0\right\}$;
9 return $\mathbf{q}_s$ and $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$;
```

## 4.1 HK-Push

Algorithm 1 shows the pseudo-code of our secondary algorithm, HK-Push, for deriving a rough approximation $\widehat{\boldsymbol{\rho}}_s$ of the HKPR vector. Its basic idea is to begin with a vector $\widehat{\boldsymbol{\rho}}_s$ where $\widehat{\boldsymbol{\rho}}_s[s] = 1$ and $\widehat{\boldsymbol{\rho}}_s[v] = 0$ for all nodes $v$ except $s$, and then perform a traversal of $G$ starting from $s$, and keep refining $\widehat{\boldsymbol{\rho}}_s$ during the course of the traversal. In addition, to facilitate its combination with random walks, it not only returns an approximate HKPR vector $\mathbf{q}_s$, but also outputs $K+1$ auxiliary vectors $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)} \in \mathbb{R}^n$ that could be used to guide the random walks conducted by Monte-Carlo. We refer to $\mathbf{q}_s$ as the *reserve vector* and $\mathbf{r}_s^{(k)}$ as the *$k$-hop residue vector*. Accordingly, for any node $v$, $\mathbf{q}_s[v]$ and $\mathbf{r}_s^{(k)}[v]$ are referred to as the *reserve* and *$k$-hop residue* of $v$, respectively.

More specifically, HK-Push takes as input $G$, $s$, and a residue threshold $r_{max}$. It begins by setting all entries in $\mathbf{q}_s$ and $\mathbf{r}_s^{(k)}$ to zero, except that $\mathbf{r}_s^{(0)}[s] = 1$ (Lines 1-2). After that, it starts an iterative process to traverse $G$ from $s$ (Lines 3-7). In particular, in each iteration, it inspects the $k$-hop residue vectors to identify a node $v$ whose $k$-hop residue $\mathbf{r}_s^{(k)}[v]$ is above $r_{max} \cdot d(v)$. If such a node $v$ exists, then the algorithm updates the reserve and $k$-hop residue of $v$, as well as the $(k+1)$-hop residues of $v$'s neighbors. In particular, it first adds $\frac{\eta(k)}{\psi(k)}$ fraction of $v$'s $k$-hop residue $\mathbf{r}_s^{(k)}[v]$ to its reserve $\mathbf{q}_s[v]$, where $\eta(k)$ is as defined in Equation (1) and

$$\psi(k) = \sum_{\ell=k}^{\infty} \eta(\ell), \tag{3}$$

and then evenly distribute the other $1 - \frac{\eta(k)}{\psi(k)}$ fraction to the $(k+1)$-hop residues of $v$'s neighbors (Lines 4-6). For convenience, we refer to the operation of distributing a fraction of $v$'s $k$-hop residue to one of its neighbors as a *push* operation.

**Algorithm 2:** $k$-RandomWalk

```
Input: Graph $G$, node $u$, constant $k$,
Output: An end node $v$
$\ell \leftarrow k$;
$v_0 \leftarrow u$;
while True do
    if rand(0, 1) $\leq \frac{\eta(k+\ell)}{\psi(k+\ell)}$ then
        break;
    else
        Pick a neighbor $v_{\ell+1} \in N(v_\ell)$ uniformly at
          random;
        $\ell \leftarrow \ell + 1$;
return $v_\ell$;
```

The rationale of the aforementioned push operations is that, if a random walk from $s$ arrives at $v$ at the $k$-th hop, then it has $\frac{\eta(k)}{\psi(k)}$ probability to terminate at $v$, and has $1 - \frac{\eta(k)}{\psi(k)}$ probability to traverse to a randomly selected neighbor of $v$ at the next hop. After that, the algorithm sets $\mathbf{r}_s^{(k)}[v] = 0$ (Line 7), and proceeds to the next iteration. After the iterative process terminates, HK-Push identifies the largest $K$ such that $\mathbf{r}_s^{(K)}$ has at least one non-zero entry, and returns $\mathbf{q}_s$ and $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$. The following lemma shows a crucial property of these reserve and residue vectors:

Lemma 1. *Consider any iteration in Algorithm 1. Let $\mathbf{q}_s$ and $\mathbf{r}_s^{(0)}, \ldots \mathbf{r}_s^{(K)}$ be the reserve and residue vectors constructed by the end of the iteration. We have*

$$\boldsymbol{\rho}_s[v] = \mathbf{q}_s[v] + \textstyle\sum_{u \in V} \sum_{k=0}^{K} \mathbf{r}_s^{(k)}[u] \cdot \mathbf{h}_u^{(k)}[v], \tag{4}$$

*where*

$$\mathbf{h}_u^{(k)}[v] = \textstyle\sum_{\ell=0}^{\infty} \frac{\eta(k+\ell)}{\psi(k)} \cdot \mathbf{P}^{\ell}[u, v], \tag{5}$$

*i.e., $\mathbf{h}_u^{(k)}[v]$ is the probability that a random walk stops at $v$, conditioned on the $k$-hop of the walk is at $u$.* □

Intuitively, Lemma 1 indicates that for any node $v$, $\mathbf{q}_s[v]$ is a lower bound of $\boldsymbol{\rho}_s[v]$ in any iteration in Algorithm 1. Since each iteration of Algorithm 1 only increases the reserve of a selected node and never decreases any others, it guarantees that the difference between $\mathbf{q}_s$ and $\boldsymbol{\rho}_s$ monotonically decreases, i.e., $\mathbf{q}_s$ becomes a better approximation of $\boldsymbol{\rho}_s$ as the algorithm progresses. Although HK-Push may produce results that are far from satisfying the requirements of $(d, \epsilon_r, \delta)$-approximate HKPR, it is sufficient for the integration with Monte-Carlo, as we show in Section 4.2.

## 4.2 Algorithm

**Basic Idea.** To incorporate HK-Push into Monte-Carlo, we utilize Equation (4), which shows that the exact HKPR vector $\boldsymbol{\rho}_s$ can be expressed as a function of $\mathbf{q}_s$, $\mathbf{r}_s^{(k)}$, and $\mathbf{h}_u^{(k)}[v]$ for

**Algorithm 3:** TEA

```
Input: Graph $G$, seed node $s$, thresholds $\epsilon_r$ and $\delta$, threshold
       $r_{max}$, and failure probability $p_f$
Output: A $(d, \epsilon_r, \delta)$-approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$
1  if $\sum_{v \in V} p_f^{d(v)-1} \le 1$ then
2      $p'_f \leftarrow p_f$;
3  else
4      $p'_f \leftarrow \frac{p_f}{\sum_{v \in V} p_f^{d(v)-1}}$;
5  $\omega \leftarrow \frac{2(1+\epsilon_r/3)\log(1/p'_f)}{\epsilon_r^2 \delta}$;
6  $\left(\widehat{\boldsymbol{\rho}}_s, \mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}\right) \leftarrow$ HK-Push $(s, r_{max})$;
7  $\alpha \leftarrow \sum_{k=0}^{K} \sum_{u \in V} \mathbf{r}_s^{(k)}[u]$;
8  $n_r \leftarrow \alpha \cdot \omega$;
9  for $i = 1$ to $n_r$ do
10     Sample an entry $(u, k)$ from $V \times \{0, 1, \ldots, K\}$ with
         probability $\frac{\mathbf{r}_s^{(k)}[u]}{\alpha}$;
11     $v \leftarrow k$-RandomWalk $(G, u, k)$;
12     $\widehat{\boldsymbol{\rho}}_s[v] \leftarrow \widehat{\boldsymbol{\rho}}_s[v] + \frac{\alpha}{n_r}$;
13 return $\widehat{\boldsymbol{\rho}}_s$;
```

any $u, v \in V$, and $k \in [0, K]$. Recall that $\mathbf{q}_s$ and $\mathbf{r}_s^{(k)}$ are outputs of HK-Push, while $\mathbf{h}_u^{(k)}[v]$ is the conditional probability that a random walk terminates at node $v$ given that its $k$-th hop is at node $u$. If we can accurately estimate $\mathbf{h}_u^{(k)}[v]$ for any given $u$, $v$, and $k$, then we can combine the estimated values with $\mathbf{q}_s$ and $\mathbf{r}_s^{(k)}$ to obtain an approximate version of $\boldsymbol{\rho}_s$. Towards this end, we conduct random walks starting from $u$, and estimate $\mathbf{h}_u^{(k)}[v]$ based on the frequency that $v$ appears at the $k$-th hop of the random walks. Algorithm 2 shows the pseudo-code of our random walk generation method, referred to as $k$-RandomWalk.

The following lemma proves that $k$-RandomWalk samples each node $v$ with probability $\mathbf{h}_u^{(k)}[v]$.

Lemma 2. *Given $G$, $u$, and $k$, for any node $v$, Algorithm 2 returns $v$ with probability $\mathbf{h}_u^{(k)}[v]$.* □

**Details.** Algorithm 3 illustrates the pseudo-code of TEA, our first-cut solution that (i) incorporates HK-Push and $k$-RandomWalk and (ii) computes a $(d, \epsilon_r, \delta)$-approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$ with at least $1 - p_f$ probability for any given seed node $s$. Given $G$, $\epsilon_r$, $\delta$, $r_{max}$, and failure probability $p_f$ as inputs, the algorithm starts by invoking HK-Push with three parameters: $G$, $s$, and $r_{max}$ (Line 6). It returns an approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$ and $K + 1$ residue vectors $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$ from HK-Push. Then, TEA proceeds to refine $\widehat{\boldsymbol{\rho}}_s$ using $k$-RandomWalk (Lines 7-13). In particular, TEA first computes the sum $\alpha$ of the residues in $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$ (Line 7), and computes

$$\omega = \frac{2(1+\epsilon_r/3)\log(1/p'_f)}{\epsilon_r^2 \cdot \delta},$$

where

$$p'_f = \begin{cases} p_f, & \text{if } \sum_{v \in V} p_f^{d(v)-1} \le 1 \\ \frac{p_f}{\sum_{v \in V} p_f^{d(v)-1}}, & \text{otherwise.} \end{cases} \tag{6}$$

Note that $p'_f$ can be pre-computed when the graph $G$ is loaded. Given $\omega$, TEA performs $n_r = \alpha \cdot \omega$ random walks using $k$-RandomWalk (Lines 9-12), such that the starting point $u$ of each walk is sampled with probability $\frac{\mathbf{r}_s^{(k)}[u]}{\alpha}$ (Line 10). Note that this sampling procedure can be conducted efficiently by conducting an *alias structure* [41] on the non-zero elements in $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$. For each random walk that ends at a node $v$, TEA increases $\widehat{\boldsymbol{\rho}}_s[v]$ by $\frac{\alpha}{n_r}$ (Line 12).

Observe that the parameter $r_{max}$ in TEA controls the balance between its two main components: HK-Push and $k$-RandomWalk. In particular, by Algorithm 1, the entries in $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$ are bounded by $r_{max}$. Therefore, when $r_{max}$ is small, $\alpha = \sum_{k=0}^{K} \sum_{u \in V} \mathbf{r}_s^{(k)}[u]$ would decrease, in which case the total number $\alpha \cdot \omega$ of random walks conducted by TEA would also be small. As a trade-off, the processing cost of HK-Push would increase, as shown in the following lemma:

Lemma 3. *Given residue threshold $r_{max}$, Algorithm 1 runs in $O\left(\frac{1}{r_{max}}\right)$ time and requires $O\left(\frac{1}{r_{max}}\right)$ space (excluding the space required by the input graph). In addition, in the residue vectors $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$ returned by Algorithm 1, there are $O\left(\frac{1}{r_{max}}\right)$ non-zero elements in total.*

To strike a balance between the costs incurred by HK-Push and $k$-RandomWalk, we set $r_{max} = O(\frac{1}{\omega \cdot t})$. In that case, the processing cost of HK-Push is $O(\omega \cdot t)$, while $k$-RandomWalk incurs $O(\alpha\omega t)$ expected cost, due to the following lemma:

Lemma 4. *The expected cost of each invocation of $k$-RandomWalk is $O(t)$.*

Hence, setting $r_{max} = O(\frac{1}{\omega \cdot t})$ ensures that the overheads of HK-Push and $k$-RandomWalk are roughly comparable.

## 4.3 Analysis

**Correctness.** Let $\mathbf{q}_s$ denote the approximate HKPR vector obtained from HK-Push in Line 6 of TEA, and $\widehat{\boldsymbol{\rho}}_s$ be the approximate HKPR vector eventually output by TEA. In the following, we show that $\widehat{\boldsymbol{\rho}}_s$ is a $(d, \epsilon_r, \delta)$-approximate HKPR vector.

First, by Lemma 1, we have the following equation for any node $v$:

$$\boldsymbol{\rho}_s[v] = \mathbf{q}_s[v] + \mathbf{a}_s[v], \tag{7}$$

where

$$\mathbf{a}_s[v] = \alpha \cdot \sum_{k=0}^{K} \sum_{u \in V} \frac{\mathbf{r}_s^{(k)}[u]}{\alpha} \cdot \mathbf{h}_u^{(k)}[v]. \tag{8}$$

Consider the $i$-th invocation of $k$-RandomWalk in TEA. Let $(u, k)$ be the entry sampled by TEA (in Line 10) before the invocation, and $v$ be the node returned by $k$-RandomWalk. Let $Y_i$ be a Bernoulli variable that equals 1 if $v$ is returned, and 0 otherwise. By Lemma 2,

$$\mathbb{E}[Y_i] = \sum_{u \in V} \sum_{k=0}^{K} \frac{\mathbf{r}_s^{(k)}[u]}{\alpha} \cdot \mathbf{h}_u^{(k)}[v]. \tag{9}$$

Combining Equations (8) and (9), we have

$$\mathbb{E}\left[\sum_{i=1}^{n_r} Y_i \cdot \frac{\alpha}{n_r}\right] = \mathbf{a}_s[v], \tag{10}$$

which indicates that $\sum_{i=1}^{n_r} Y_i \cdot \frac{\alpha}{n_r}$ is an unbiased estimator of $\mathbf{a}_s[v]$. By the Chernoff bound (in Lemma 5), we can prove that this estimator is highly accurate, based on which we obtain Theorem 1.

Lemma 5 (Chernoff Bound [10]). *Let $X_1, X_2, \cdots, X_{n_r} \in [0, 1]$ be i.i.d. random variables, and $X = \sum_{i=1}^{n_r} X_i$. Then,*

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq \lambda] \leq \exp\left(-\frac{\lambda^2}{2\mathbb{E}[X] + 2\lambda/3}\right). \qquad \square$$

Theorem 1. TEA *outputs a $(d, \epsilon_r, \delta)$-approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$ with probability at least $1 - p_f$.*

**Time and Space Complexities.** Given $r_{max} = O\left(\frac{1}{\omega \cdot t}\right)$, HK-Push runs in $O(\omega \cdot t)$ time and $O(\omega \cdot t)$ space. In addition, the computation of $\alpha$ as well as the construction of alias structure on $\mathbf{r}_s^{(k)}$ can be done in time and space linear to the total number of non-zero entries in $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$, which is $O(\omega \cdot t)$ (see Lemma 3). Furthermore, in terms of both space and time, the total expected cost incurred by the random walks in TEA is $O(\alpha \omega t)$, where $\alpha < 1$. Therefore, the time complexity of TEA is

$$O\left(\frac{1}{r_{max}} + \alpha \cdot \omega t\right) = O\left(\frac{t \log\left(1/p_f'\right)}{\epsilon_r^2 \cdot \delta}\right) = O\left(\frac{t \log\left(n/p_f\right)}{\epsilon_r^2 \cdot \delta}\right),$$

and its space complexity is $O\left(n + m + \frac{t \log(n/p_f)}{\epsilon_r^2 \cdot \delta}\right)$, where the $n + m$ term is due to storing of the input graph.

# 5 THE TEA+ ALGORITHM

Although TEA provides a strong accuracy guarantee, we observe in our experiments that it often performs a large number of random walks, which degrades its computation efficiency. One may attempt to reduce the cost of random walks by decreasing the residue threshold $r_{max}$ in TEA (see the discussion in the end of Section 4.2), but this cost reduction would be offset by the fact that HK-Push incurs a larger overhead when $r_{max}$ is small.

In this section, we present TEA+, an algorithm that significantly improves over TEA in terms of practical efficiency without degrading its theoretical guarantees. TEA+ is similar in spirit to TEA in that it combines random walks with a variant of HK-Push, but there is a crucial difference: after TEA+ obtains the residue vectors $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$, it may reduce some entries in the residue vectors before performing random walks. That is, for each node $u$ that has a non-zero entry $\mathbf{r}_s^{(k)}[u]$ for some $k$, TEA+ may choose to perform a smaller number of random walks from $u$ than TEA does, which effectively reduces the total cost of random walks. Establishing the correctness of this pruning approach, however, is nontrivial. In what follows, we first discuss in Section 5.1 the extreme case where we can derive $(d, \epsilon_r, \delta)$-approximate HKPR while *ignoring all elements in the residue vectors* $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$; after that, in Section 5.2, we generalize our discussions to the case where we reduce the non-zero entries in the residue vectors instead of completely omitting them.

**Algorithm 4:** HK-Push+

```
Input: Graph $G$, seed node $s$, thresholds $\epsilon_r$ and $\delta$,
       maximum hop number $K$, maximum number of
       pushes $n_p$
Output: An approximate HKPR vector $\mathbf{q}_s$ and $K + 1$
        residue vectors $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$
1 $\mathbf{q}_s \leftarrow \mathbf{0}$, $\mathbf{r}_s^{(k)} \leftarrow \mathbf{0}$ for $k = 0, \cdots, K$;
2 $\mathbf{r}_s^{(0)}[s] \leftarrow 1$;
3 $i \leftarrow 0$;
4 while $\exists v \in V, k < K$ such that $\mathbf{r}_s^{(k)}[v] > \frac{\epsilon_r \cdot \delta}{K} \cdot d(v)$ do
5     $i \leftarrow i + d(v)$;
6     if $i \geq n_p$ or $\sum_{\ell=0}^{K} \max_{u \in V} \left\{\frac{\mathbf{r}_s^{(\ell)}[u]}{d(u)}\right\} \leq \epsilon_r \cdot \delta$ then
7         break;
      Lines 8-11 are the same as Lines 4-7 in Algorithm 1;
  Line 12 is the same as Line 9 in Algorithm 1;
```

## 5.1 The Case without Random Walks

Suppose that we are to let TEA achieve $(d, \epsilon_r, \delta)$-approximation without performing any random walks. In that case, we would need to ensure that Line 6 of TEA obtains a $(d, \epsilon_r, \delta)$-approximate HKPR vector from HK-Push. Towards this end, we present the following theorem:

Theorem 2. *Let $\mathbf{q}_s$ and $\mathbf{r}_s^{(0)}, \ldots \mathbf{r}_s^{(K)}$ be the reserve and residue vectors returned by* HK-Push. *If*

$$\sum_{\ell=0}^{K} \max_{v \in V} \left\{\frac{\mathbf{r}_s^{(\ell)}[v]}{d(v)}\right\} \leq \epsilon_a, \tag{11}$$

*then, for any node $v$ in $G$, we have*

$$\left|\frac{\mathbf{q}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \leq \epsilon_a. \tag{12}$$

Theorem 2 provides a sufficient condition (i.e., Inequality (11)) for HK-Push to ensure $\epsilon_a$ absolute error in each $\frac{\mathbf{q}_s[v]}{d(v)}$. By Definition 1, such $\mathbf{q}_s$ is a $(d, \epsilon_r, \delta)$-approximate HKPR vector as long as $\epsilon_a \leq \epsilon_r \cdot \delta$. That said, it is rather inefficient to let HK-Push run until Inequality (11) is satisfied. Instead, we propose to let HK-Push run with a fixed budget of processing cost. If it is able to satisfy Inequality (11) with $\epsilon_a = \epsilon_r \cdot \delta$ before the budget is depleted, then we return $\mathbf{q}_s$ as the final result; otherwise, we proceed to refine $\mathbf{q}_s$ using random walks (see Section 5.2).

Based on the above discussion, we present HK-Push+ (in Algorithm 4), which is a revised version of HK-Push with three major changes. First, HK-Push+'s input parameters include three thresholds $\epsilon_r$, $\delta$, and $n_p$, and it has two new termination conditions (in Line 6): (i) Inequality (11) holds with $\epsilon_a = \epsilon_r \cdot \delta$; (ii) The number of push operations that it performs reaches $n_p$. Recall that a push operation refers to the operation of converting part of a node's $k$-hop residue to one of its neighbor's $(k+1)$-hop residue. In other words, HK-Push+ runs in $O(n_p)$ time and requires $O(n_p)$ space, and it returns a $(d, \epsilon_r, \delta)$-approximate HKPR vector whenever Inequality (11) is satisfied.

Second, HK-Push+ judiciously performs push operations only on nodes $v$ with residue $\mathbf{r}_s^{(k)}[v] > \frac{\epsilon_r \cdot \delta}{K} \cdot d(v)$ (Line 4), whereas HK-Push conducts push operations only when $\mathbf{r}_s^{(k)}[v]$ is larger than an input given threshold $r_{max} \cdot d(v)$. The rationale is that HK-Push+ strives to reduce the $k$-hop residue of each node below $\frac{\epsilon_r \cdot \delta}{K} \cdot d(v)$, so as to satisfy Inequality (11); in contrast, HK-Push is not guided by Inequality (11), and hence, uses an ad hoc threshold $r_{max}$ instead.

Third, HK-Push+ makes the maximum number $K$ of hops be specified as an input parameter, whereas HK-Push does not fix $K$ in advance. We use a fixed $K$ in HK-Push+ because (i) as mentioned, Line 4 of HK-Push+ requires testing whether there exists a node $v$ with $\mathbf{r}_s^{(k)}[v] > \frac{\epsilon_r \cdot \delta}{K} \cdot d(v)$, and (ii) such a test can be efficiently implemented when $K$ is fixed. Otherwise, whenever $K$ changes, we would need to recheck all nodes' residues to see if $\mathbf{r}_s^{(k)}[v] > \frac{\epsilon_r \cdot \delta}{K} \cdot d(v)$ holds, which would incur considerable overheads. Meanwhile, HK-Push can afford to let $K$ dynamically change, since it uses a fixed residue threshold $r_{max}$ given as input. In our implementation of HK-Push+, we set

$$K = c \cdot \frac{\log\left(\frac{1}{\epsilon_r \cdot \delta}\right)}{\log(\bar{d})},$$

where $\bar{d}$ is the average degree of the nodes in $G$, and $c$ is a constant that we decide based on our experiments in Section 7.2. We refer interested readers to [1] for the rationale of this setting of $K$.

## 5.2 The Case with Random Walks

Suppose that HK-Push+ depletes its computation budget $n_p$ before it satisfies Inequality (11) with $\epsilon_a = \epsilon_r \cdot \delta$. In that case, the HKPR vector $\mathbf{q}_s$ returned by HK-Push+ does not ensure $(d, \epsilon_r, \delta)$-approximation, and we need to refine $\mathbf{q}_s$ by conducting random walks according to the residue vectors $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$ returned by HK-Push+. To reduce the number of random walks required, we propose to reduce the residues in $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$ when conducting random walks, based on the following intuition.

**Table 3: An example for TEA+**

|  | $k \leq 2$ | $k = 3$ | $k = 4$ |
|---|---|---|---|
| $\sum_{v \in V} \mathbf{r}_s^{(k)}[v]$ | 0 | 0.1 | 0.3 |
| $\max_{v \in V} \frac{\mathbf{r}_s^{(k)}[v]}{d(v)}$ | 0 | $\frac{\mathbf{r}_s^{(k)}[v_1]}{d(v_1)} = 0.0025$ | $\frac{\mathbf{r}_s^{(k)}[v_2]}{d(v_2)} = 0.0076$ |
| $\max_{v \in V} \mathbf{r}_s^{(k)}[v]$ | 0 | $\mathbf{r}_s^{(k)}[v_1] = 0.0025$ | $\mathbf{r}_s^{(k)}[v_2] = 0.076$ |

First, the reduction of residues would incur some errors in the approximate HKPR vector, as demonstrated in Theorem 2 (where we ignore all residues in $\mathbf{r}_s^{(k)}$ and using $\mathbf{q}_s$ directly as the final approximate HKPR vector). Second, if we only reduce the residues in $\mathbf{r}_s^{(k)}$ by a small value, then the absolute errors incurred by the reduction could be so small that they would not jeopardize $(d, \epsilon_r, \delta)$-approximation. In particular, as we show in Section 5.4, if we reduce every residue $\mathbf{r}_s^{(k)}[v]$ by $\beta_k \cdot \epsilon_r \delta \cdot d(v)$ ($k = 0, 1, \ldots, K$ and $v \in V$), then the absolute error in $\frac{\widehat{\rho}_s[v]}{d(v)}$ incurred by the residue reduction is at most $\epsilon_r \delta \cdot \sum_{k=0}^{K} \beta_k$. In other words, if we choose $\beta_k$ such that $\sum_{k=0}^{K} \beta_k = 1$, then the resulting absolute error in $\frac{\widehat{\rho}_s[v]}{d(v)}$ is at most $\epsilon_r \delta$, which is permissible under $(d, \epsilon_r, \delta)$-approximation. The following example demonstrates the benefit of this residue reduction method.

Example 1. Suppose that given a graph $G$, seed node $s$, $K = 4$, and $\epsilon_r \cdot \delta = 0.01$, HK-Push+ returns residue vectors $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(4)}$ that have the characteristics in Table 3. Note that $v_1$ has the largest 3-hop residue and degree-normalized 3-hop residue, while $v_2$ has the largest 4-hop residue and degree-normalized 4-hop residue. In addition, all of the 3-hop residues except $v_1$'s are less than 0.0025, and all of the 4-hop residues except $v_2$'s are less than 0.075. We can observe that

$$\sum_{\ell=0}^{4} \max_{v \in V} \left\{ \frac{\mathbf{r}_s^{(\ell)}[v]}{d(v)} \right\} = 0.0025 + 0.0076 = 0.0101,$$

which is slightly larger than $\epsilon_r \cdot \delta$. In this case, according to Lines 7-8 in TEA, we need to perform $\alpha \cdot \omega$ random walks, where

$$\alpha = \sum_{k=0}^{K} \sum_{u \in V} \mathbf{r}_s^{(k)}[u] = 0.4.$$

Now suppose that we apply the residue reduction method, setting $\beta_3 = 1/4$ and $\beta_4 = 3/4$. In that case, we reduce every residue $\mathbf{r}_s^{(3)}[v]$ by $\beta_3 \cdot \epsilon_r \delta \cdot d(v) = 0.0025 \cdot d(v)$, and every residue $\mathbf{r}_s^{(4)}[v]$ by $\beta_4 \cdot \epsilon_r \delta \cdot d(v) = 0.0075 \cdot d(v)$. Then, all residues in $\mathbf{r}_s^{(3)}$ and $\mathbf{r}_s^{(4)}$ are reduced to 0, except that $\mathbf{r}_s^{(4)}[v_2]$ is decreased to $0.076 - \beta_4 \cdot \epsilon_r \delta \cdot d(v_2) = 0.001$. Accordingly, $\alpha = \sum_{\ell=0}^{4} \max_{v \in V} \mathbf{r}_s^{(\ell)}[v]$ is reduced from 0.4 to 0.001, which implies that the number of random walks required is reduced by 400 times. □

## 5.3 Details of TEA+

Based on the ideas described in Sections 5.1 and 5.2, we present TEA+, which utilizes HK-Push+ and random walks to compute a $(d, \epsilon_r, \delta)$-approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$ with at least $1-p_f$ probability for any given seed node $s$. Algorithm 5 illustrates the pseudo-code of TEA+. The input parameters of TEA+ are identical to those of TEA, except that TEA+ takes an additional parameter $c$, which, as mentioned in Section 5.1, is used to decide the maximum number $K$ of hops used in HK-Push+.

TEA+ starts by invoking HK-Push+ with the following parameters (Lines 5-6): $G, \epsilon_r, \delta, K = c\cdot\frac{\log(\frac{1}{\epsilon_r\cdot\delta})}{\log(\bar{d})}$, and $n_p = \frac{\omega\cdot t}{2}$, where

$$\omega = \frac{8(1+\epsilon_r/6)\log(1/p'_f)}{\epsilon_r^2\cdot\delta},$$

and $p'_f$ is as defined in Equation (6) and is pre-computed when $G$ is loaded. Then, HK-Push+ returns an approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$ and $K+1$ residue vectors $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$.

If $\sum_{k=0}^{K}\max_{u\in V}\left\{\frac{\mathbf{r}_s^{(k)}[u]}{d(u)}\right\} \le \epsilon_r\cdot\delta$, then by Theorem 2, $\widehat{\boldsymbol{\rho}}_s$ is a $(d, \epsilon_r, \delta)$-approximate HKPR vector. In that case, TEA+ immediately terminates and returns $\widehat{\boldsymbol{\rho}}_s$ (Line 7). Otherwise, TEA+ proceeds to refine $\widehat{\boldsymbol{\rho}}_s$ using $k$-RandomWalk (Lines 8-20). But before that, TEA+ first applies the residue reduction method discussed in Section 5.2. Specifically, it decreases each residue value $\mathbf{r}_s^{(k)}[u]$ by $\beta_k\cdot\epsilon_r\delta\cdot d(u)$ (Lines 8-11), where

$$\beta_k = \frac{\sum_{u\in V}\mathbf{r}_s^{(k)}[u]}{\sum_{\ell=0}^{K}\sum_{u\in V}\mathbf{r}_s^{(\ell)}[u]}.$$

The rationale of this choice of $\beta_k$ is as follows. First, $\sum_{k=0}^{K}\beta_k = 1$, which is crucial for $(d, \epsilon_r, \delta)$-approximation, as we mention in Section 5.2. Second, we set $\beta_k$ to be proportional to $\sum_{u\in V}\mathbf{r}_s^{(k)}[u]$ because, intuitively, when $\sum_{u\in V}\mathbf{r}_s^{(k)}[u]$ is large, the residue values in $\mathbf{r}_s^{(k)}$ also tend to be large, in which case we need a larger reduction of the residues in $\mathbf{r}_s^{(k)}$ to effectively reduce the number of random walks needed.

After reducing the residues in $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$, TEA+ performs random walks according to the reduced residues, in the same way as TEA does (Lines 12-17). This results in a refined approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$. Then, TEA+ gives $\widehat{\boldsymbol{\rho}}_s[v]$ a final touch by adding $\frac{\epsilon_r\cdot\delta}{2}\cdot d(v)$ to each $\widehat{\boldsymbol{\rho}}_s[v]$ (Lines 18-19). The intuition of adding this offset to each $\widehat{\boldsymbol{\rho}}_s[v]$ is as follows. The residue reduction method leads to an underestimation of each HKPR value, and amount of underestimation is in $[0, \epsilon_r\cdot\delta\cdot d(v)]$. By adding an offset $\frac{\epsilon_r\cdot\delta}{2}\cdot d(v)$ to $\widehat{\boldsymbol{\rho}}_s[v]$, the range of the error in $\widehat{\boldsymbol{\rho}}_s[v]$ becomes $[-\frac{\epsilon_r\cdot\delta}{2}\cdot d(v), \frac{\epsilon_r\cdot\delta}{2}\cdot d(v)]$, in which case the maximum absolute error in $\widehat{\boldsymbol{\rho}}_s[v]$ is reduced by half, which help tightening the accuracy bound of TEA+.

Note that Lines 18-19 in TEA+ can be performed in $O(1)$ time, as we can keep each $\widehat{\boldsymbol{\rho}}_s[v]$ unchanged but record the value of $\frac{\epsilon_r\cdot\delta}{2}$ along with $\widehat{\boldsymbol{\rho}}_s$. Then, whenever $\widehat{\boldsymbol{\rho}}_s[v]$ is accessed, we can add $\frac{\epsilon_r\cdot\delta}{2}\cdot d(v)$ to $\widehat{\boldsymbol{\rho}}_s[v]$ on the fly. In addition, for the purpose of local clustering, we can simply ignore this offset of $\frac{\epsilon_r\cdot\delta}{2}\cdot d(v)$ since it does not affect the ranking of nodes based on $\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)}$. An example to illustrate TEA+ is given in [1].

**Algorithm 5:** TEA+

```
Input: Graph $G$, seed node $s$, constant $c$, thresholds $\epsilon_r$ and
       $\delta$, and failure probability $p_f$
Output: A $(d, \epsilon_r, \delta)$-approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$
Lines 1-4 are the same as Lines 1-4 in Algorithm 3;
5  $\omega \leftarrow \frac{8(1+\epsilon_r/6)\log(1/p'_f)}{\epsilon_r^2\delta}, n_p \leftarrow \frac{\omega\cdot t}{2}, K \leftarrow c\cdot\frac{\log(\frac{1}{\epsilon_r\cdot\delta})}{\log(\bar{d})}$;
6  $\left(\widehat{\boldsymbol{\rho}}_s, \mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}\right) \leftarrow$ HK-Push+ $(s, \epsilon_r, \delta, K, n_p)$;
7  if $\sum_{k=0}^{K}\max_{u\in V}\left\{\frac{\mathbf{r}_s^{(k)}[u]}{d(u)}\right\} \le \epsilon_r\cdot\delta$ then return $\widehat{\boldsymbol{\rho}}_s$ ;
8  for $k = 0$ to $K$ do
9      $\beta_k \leftarrow \frac{\sum_{u\in V}\mathbf{r}_s^{(k)}[u]}{\sum_{\ell=0}^{K}\sum_{u\in V}\mathbf{r}_s^{(\ell)}[u]}$;
10     for any node $u$ with $\mathbf{r}_s^{(k)}[u] > 0$ do
11         $\mathbf{r}_s^{(k)}[u] = \max\left\{0, \mathbf{r}_s^{(k)}[u] - \beta_k\cdot\epsilon_r\delta\cdot d(u)\right\}$;
Lines 12-17 are the same as Lines 7-12 in Algorithm 3;
18 for $v \in V$ do
19     $\widehat{\boldsymbol{\rho}}_s[v] \leftarrow \widehat{\boldsymbol{\rho}}_s[v] + \frac{\epsilon_r\cdot\delta}{2}\cdot d(v)$;
20 return $\widehat{\boldsymbol{\rho}}_s$;
```

## 5.4 Analysis

**Correctness.** Let $\mathbf{q}_s$ denote the approximate HKPR vector returned by HK-Push+ in Line 6 of TEA+, $\mathbf{r}_s^{(0)}, \ldots, \mathbf{r}_s^{(K)}$ be the residue vectors output by HK-Push+ at the same step, and $\widehat{\boldsymbol{\rho}}_s$ be the final version of the HKPR vector output by TEA+. We define a residue vector $\mathbf{rb}_s^{(k)}$ as:

$$\mathbf{rb}_s^{(k)}[u] = \min\left\{\mathbf{r}_s^{(k)}[u],\ \beta_k\cdot\epsilon_r\delta\cdot d(u)\right\}. \tag{13}$$

Observe that $\mathbf{rb}_s^{(k)}[u]$ equals the amount of residue reduction on $\mathbf{r}_s^{(k)}[u]$ occurred in Lines 8-11 of TEA+.

Similar to the correctness analysis in Section 4.3, by Lemma 1, we have the following equation for any node $v$:

$$\boldsymbol{\rho}_s[v] = \mathbf{q}_s[v] + \mathbf{a}_s[v] + \mathbf{b}_s[v], \tag{14}$$

where

$$\mathbf{a}_s[v] = \alpha\cdot\sum_{k=0}^{K}\sum_{u\in V}\frac{\mathbf{r}_s^{(k)}[u]}{\alpha}\cdot\mathbf{h}_u^{(k)}[v], \text{ and} \tag{15}$$

$$\mathbf{b}_s[v] = \sum_{k=0}^{K}\sum_{u\in V}\mathbf{rb}_s^{(k)}[u]\cdot\mathbf{h}_u^{(k)}[v]. \tag{16}$$

Then, the approximation error in each $\widehat{\boldsymbol{\rho}}_s[v]$ can be regarded as the sum of two approximation errors for $\mathbf{a}_s[v]$ and $\mathbf{b}_s[v]$, respectively. The error in $\mathbf{a}_s[v]$ is caused by sampling errors in $k$-RandomWalk, and hence, it can be bounded using the Chernoff bound, in a way similar to the analysis in Section 4.3. Meanwhile, the error in $\mathbf{b}_s[v]$ is due to the residue reduction procedure in Lines 8-11 of TEA+. In what follows, we present an analysis of the error in $\mathbf{b}_s[v]$.

By Equation (13), for any node $u \in V$ and $k \in [0, K]$, the amount of residue reduction on $\mathbf{r}_s^{(k)}[u]$ satisfies $0 \le \mathbf{rb}_s^{(k)}[u] \le \beta_k \cdot \epsilon_r \delta \cdot d(u)$. Combining this with Equation (16), we have

$$0 \le \mathbf{b}_s[v] \le \textstyle\sum_{k=0}^{K} \left(\beta_k \cdot \epsilon_r \delta \sum_{u \in V} d(u) \cdot \mathbf{h}_u^{(k)}[v]\right). \quad (17)$$

Lemma 6 ([37]). *Let $u$ and $v$ be any two nodes in $G$, and $\mathbf{P}^k[u,v]$ (resp. $\mathbf{P}^k[v,u]$) be the probability that a length-$k$ random walk from $u$ ends at $v$ (resp. from $v$ ends at $u$). Then, $\frac{\mathbf{P}^k[u,v]}{d(v)} = \frac{\mathbf{P}^k[v,u]}{d(u)}$.* □

By Lemma 6 and the definition of $\mathbf{h}_u^{(k)}[v]$ in Equation 5, for any node $v \in V$ and $k \in [0, K]$, we have

$$\begin{aligned} \textstyle\sum_{u \in V} d(u) \cdot \mathbf{h}_u^{(k)}[v] &= d(v) \cdot \textstyle\sum_{u \in V} \mathbf{h}_v^{(k)}[u] \\ &= d(v) \cdot \textstyle\sum_{\ell=0}^{\infty} \left[\frac{\eta(k+\ell)}{\psi(k)} \cdot \sum_{u \in V} \mathbf{P}^{\ell}[v,u]\right] \\ &= d(v) \cdot \textstyle\sum_{\ell=0}^{\infty} \frac{\eta(k+\ell)}{\psi(k)} = d(v). \quad (18) \end{aligned}$$

Combining Equations (17) and (18), we have

$$0 \le \mathbf{b}_s[v] \le d(v) \cdot \epsilon_r \delta. \quad (19)$$

Therefore, estimating $\mathbf{b}_s[v]$ as $\frac{\epsilon_r \cdot \delta}{2} \cdot d(v)$ incurs an absolute error of at most $\frac{\epsilon_r \cdot \delta}{2} \cdot d(v)$.

Based on the above analysis, we establish the accuracy guarantee of TEA+ as follows:

Theorem 3. *TEA+ outputs a $(d, \epsilon_r, \delta)$-approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$ with probability at least $1 - p_f$.*

**Time and Space Complexities.** The time and space complexities of TEA+ depend on its two main components: HK-Push+ and $k$-RandomWalk. As discussed in Section 5.1, both the computation and space overheads of HK-Push+ are $O(n_p)$. Since TEA+ sets $n_p = \frac{\omega \cdot t}{2}$, its invocation of HK-Push+ incurs $O\left(\frac{t \cdot \log(n/p_f)}{\epsilon_r^2 \cdot \delta}\right)$ time and space costs. Meanwhile, the total number of random walks conducted by TEA+ is no more than that by TEA, and hence, the computational and space costs of generating random walks in TEA+ are both $O\left(\frac{t \cdot \log(n/p_f)}{\epsilon_r^2 \cdot \delta}\right)$ in expectation. Thus, the expected time and space complexities of TEA+ are $O\left(\frac{t \cdot \log(n/p_f)}{\epsilon_r^2 \cdot \delta}\right)$ and $O\left(m + n + \frac{t \cdot \log(n/p_f)}{\epsilon_r^2 \cdot \delta}\right)$, respectively, where the $m + n$ term is due to the space required by the input graph.

## 6 RELATED WORK

In this section, we first review two HKPR algorithms, ClusterHKPR and HK-Relax, that are most related to our solutions; after that, we review other work related to local clustering and HKPR computation.

**ClusterHKPR**. ClusterHKPR [11] is a random-walk-based method for computing approximate HKPR. Given a seed node $s$, it performs $\frac{16 \log n}{\epsilon^3}$ random walks from $s$, with the constraint that each walk has a length at most $K$; after that, for each node $v$, it uses the fraction of walks that end at $v$ as an estimation $\widehat{\boldsymbol{\rho}}_s[v]$ of $v$'s HKPR. It is shown that with probability at least $1 - \epsilon$, ClusterHKPR guarantees that

$$\begin{cases} |\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v]| \le (1 + \epsilon) \cdot \boldsymbol{\rho}_s[v], & \text{if } \boldsymbol{\rho}_s[v] > \epsilon \\ |\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v]| \le \epsilon, & \text{otherwise.} \end{cases}$$

Note that for the above guarantee to be meaningful, $\epsilon \ll 1$ should hold; otherwise, the successful probability $1 - \epsilon$ of ClusterHKPR would be too small, and there could be too many nodes having a large absolute error up to $\epsilon$. In particular, if we are to ensure that ClusterHKPR achieves $(d, \epsilon_r, \delta)$-approximation with probability at least $1 - p_f$, then we have to set $\epsilon \le \min\{\epsilon_r \cdot \delta, p_f\}$. However, when $\epsilon \ll 1$, ClusterHKPR incurs excessive computation cost, since its expected time complexity is inversely proportional to $\epsilon^3$.

**HK-Relax**. HK-Relax [17] is a deterministic algorithm that runs in $O\left(\frac{te^t \log(1/\epsilon_a)}{\epsilon_a}\right)$ time and returns an approximate HKPR vector $\widehat{\boldsymbol{\rho}}_s$ such that $\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \le \epsilon_a$ for any $v \in V$. HK-Relax is similar to our HK-Push algorithm in that they both (i) maintain an approximation HKPR vector $\widehat{\boldsymbol{\rho}}_s$ and a number of residue vectors, and (ii) incrementally refine $\widehat{\boldsymbol{\rho}}_s$ by applying push operations according to the residue of each node. However, there exist three major differences between HK-Relax and HK-Push. First, HK-Relax and HK-Push define the residue of each node in different manners, due to which HK-Relax requires more sophisticated approaches than HK-Push to update $\widehat{\boldsymbol{\rho}}_s$ and the residue vectors after each push operation. Second, HK-Relax ignores all $k$-hop residues with $k > 2t \log \frac{1}{\epsilon_a}$, whereas HK-Push retains all residues generated for the refinement of $\widehat{\boldsymbol{\rho}}_s$ via random walks. Third, HK-Relax and HK-Push have different termination conditions. Due to these differences, we are unable combine HK-Relax with random walks to achieve the same performance guarantee provided by our TEA and TEA+ algorithms.

**Other methods for local clustering.** The first local graph clustering algorithm, Nibble, is introduced in the seminal work [21, 38] by Spielman and Teng. After that, Anderson *et al.* [3] propose PR-Nibble, a local clustering algorithm based on *personalized PageRank* [15, 32], and show that it improves over Nibble in terms of the theoretical guarantees of both clustering quality and time complexity. In turn, Anderson *et al.*'s method is improved in subsequent work

[4, 31] based on the *volume-biased evolving set process* [14]. Subsequent work [26, 36, 39, 43] achieves further improved guarantees on the quality of local clustering, but the methods proposed are mostly of theoretical interests only, as they are difficult to implement and offer rather poor practical efficiency. As a consequence, HK-Relax remains the state-of-the-art for local clustering in terms of practical performance [5, 12, 17].

In recent work [22], Shun *et al.* study parallel implementations for Nibble, PR-Nibble, ClusterHKPR, and HK-Relax, respectively, and are able to achieve significant speedup on a machine with 40 cores. We believe that our algorithms may also exploit parallelism for higher efficiency, but a thorough treatment of this problem is beyond the scope of this paper.

**Methods for personalized PageRank.** We note that TEA and TEA+ are similar in spirit to several recent methods [19, 25, 27, 28] for computing personalized PageRank (PPR), since they all combine a push-operation-based algorithm with random walks. Hence, at first glance, it may seem that we can simply adopt and extend these techniques to address HKPR computation. Unfortunately, this is not the case as HKPR is inherently more sophisticated than PPR. In particular, even though both HKPR and PPR measure the proximity of a node $v$ with respect to another node $u$ by the probability that a random walk starting from $u$ would end at $v$, they differ significantly in the ways that they define random walks:

- PPR's random walks are *Markovian*: in each step of a walk, it terminates with a fixed probability $\alpha \in (0, 1)$, regardless of the previous steps.
- HKPR's random walks are *non-Markovian*: the termination probability of a walk at the $i$-th step is a function of $i$, i.e., the walk has to remember the number of steps that it has traversed, so as to decide whether it should terminate.

The Markovianness of PPR random walks is a key property exploited in the methods in [19, 25, 27, 28]. Specifically, when computing the PPR $p(u, v)$ from node $u$ to node $v$, the methods in [19, 25, 28] require performing a *backward search* which starts from $v$ and traverses the incoming edges of each node in a backward manner. For each node $w$ encountered and each of its incoming neighbor $x$, the backward search needs to calculate the probability that a random walk hitting $x$ at a certain step would arrive at $w$ at the next step. For PPR random walks, this probability is a constant decided only by $\alpha$ and the number of $x$'s outgoing neighbors. Unfortunately, for HKPR random walks the probability is not a constant, but a variable depending on the number of steps that the walk has taken before reaching $x$. In other words, this variable is not unique even when $w$ and $x$ are fixed, due to which the backward search no longer can be utilized. This issue makes it unpalatable to extend the methods in [19, 25, 28] to compute HKPR.

**Table 4: Statistics of graph datasets.**

| Dataset | $n$ | $m$ | $\bar{d}$ |
|---|---:|---:|---:|
| *DBLP* | 317,080 | 1,049,866 | 6.62 |
| *Youtube* | 1,134,890 | 2,987,624 | 5.27 |
| *PLC* | 2,000,000 | 9,999,961 | 9.99 |
| *Orkut* | 3,072,441 | 117,185,083 | 76.28 |
| *LiveJournal* | 3,997,962 | 34,681,189 | 17.35 |
| *3D-grid* | 9,938,375 | 29,676,450 | 5.97 |
| *Twitter* | 41,652,231 | 1,202,513,046 | 57.74 |
| *Friendster* | 65,608,366 | 1,806,067,135 | 55.06 |

Meanwhile, the FORA method in [27] does not require a backward search; instead, it combines random walks with a forward search from $u$ that is similar to the HK-Push algorithm used in TEA. However, TEA is more sophisticated than FORA as it needs to account for the non-Markovianness of HKPR, and there are three major differences between the two methods. First, TEA requires maintaining multiple residue vectors in its invocation of HK-Push, since it needs to differentiate the residues generated at different steps of the forward search; otherwise, it would not be able to combine the results of HK-Push with random walks because of HKPR's non-Markovianness. In contrast, FORA only needs to maintain one residue vector, as the Markovianness of PPR allows it to merge the resides produced at different steps of the forward search. Second, the theoretical analysis of TEA is more challenging than that of FORA, since it is more complicated to (i) maintain and update multiple residue vectors and (ii) combine random walks with the forward traversal in a way that takes into account the non-Markovianness of HKPR. Third, TEA provides an accuracy guarantee in terms of each node's *degree-normalized* HKPR, whereas FORA's accuracy guarantee is on each node's PPR without normalization.

Last but not the least, we note that our TEA+ algorithm, which significantly improves over TEA in terms of practical efficiency, is based on a new optimization that is specifically designed for HKPR and our notion of $(d, \epsilon_r, \delta)$-approximation. This optimization is inapplicable for PPR computation, which further differentiates TEA+ from FORA.

# 7 EXPERIMENTS

We now investigate the performance of our proposed algorithms and report the key results. Additional results are discussed in Appendix B and [1].

## 7.1 Experimental Setup

We conduct all experiments on a Linux server with a Intel Xeon(R) E5-2650 v2@2.60GHz CPU and 64GB RAM. For fair comparison, all algorithms are implemented in C++ and compiled by g++ 4.8 with -O3 optimization.

We use 6 undirected real-world graphs and 2 synthetic graphs which are used in recent work [12, 17, 22] as benchmark datasets (Table 4). We obtain *DBLP*, *Youtube*, *Orkut*, *LiveJournal*, and *Friendster* from [2]. *PLC* is a synthetic graph, and it is generated by Holme and Kim algorithm for generating graphs with powerlaw degree distribution and approximate average clustering. *3D-grid* is a synthetic grid graph in 3-dimensional space where every node has six edges, each connecting it to its 2 neighbors in each dimension. *Twitter* is a symmetrized version of a snapshot of the Twitter network [16]. For each dataset, we pick 50 seed nodes uniformly at random as our query sets.

Unless specified otherwise, following previous work [6, 17], we set heat constant $t = 5$. In addition, for all randomized algorithms, we set failure probability $p_f = 10^{-6}$. We report the average query time (measured in wall-clock time) of each algorithm on each dataset with various parameter settings. Note that the $y$-axis is in log-scale and the measurement unit for running time is millisecond (ms).

## 7.2 Tuning Parameter $c$ for TEA+

First, we experimentally study how to set the parameter $c$ so as to obtain the best performance for TEA+ in practice. We run TEA+ with parameters $\epsilon_r = 0.5$, $\delta = \frac{1}{n}$, and varying $c$ from 0.5 to 5 on all 8 graphs.

Figure 1 plots the running time of TEA+ on each dataset for different $c$. We omit the results for *Twitter* and *Friendster* when $c = 0.5$, because it takes several hours to finish one query. We can make the following observations. For each dataset, the running time decreases first as $c$ grows. The reason is that TEA+ degrades to Monte-Carlo when $c$ is very small, and if we keep increasing $c$, HK-Push+ will perform more push operations so as to reduce the number of random walks until $c$ balances the costs incurred for HK-Push+ and $k$-RandomWalk. On the other hand, when $c$ increases further, the overhead of HK-Push+ goes up gradually, disrupting the balance between HK-Push+ and $k$-RandomWalk. This leads to higher running time. More specifically, we can see that for graphs with small average degree including *DBLP*, *Youtube*, *PLC* and *3D-grid*, the costs are minimized when $c$ is around 2. On the other hand, for graphs with high average degree (*e.g., Orkut*, *LiveJournal*, *Twitter*, and *Friendster*), we note that $c = 2.5$ achieves the best performance. Based on the above observations, a good value choice for $c$ is 2.5, when the overheads on most of the graphs are minimized. In the sequel, we set $c = 2.5$.

## 7.3 Comparison between TEA and TEA+

In this set of experiments, we compare TEA+ with TEA based on identical theoretical accuracy guarantees. For both TEA and TEA+, we set the relative error threshold $\epsilon_r = 0.5$. Since the best values for $r_{max}$ vary largely for different parameter settings and various datasets, we are unable to find a universal optimal value for $r_{max}$. Instead, we tune $r_{max}$ for TEA with different error thresholds on each dataset separately. That is, we scale $\frac{1}{\omega \cdot t}$ up or down such that the costs for HK-Push and $k$-RandomWalk in TEA are roughly balanced and the total cost is minimized.

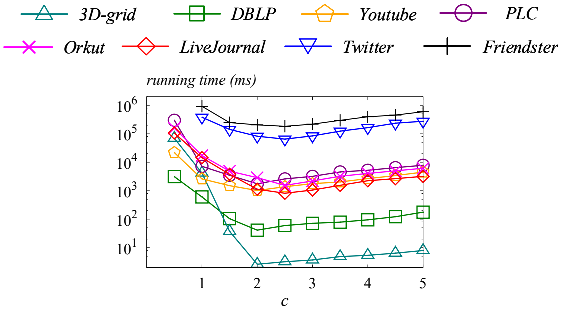
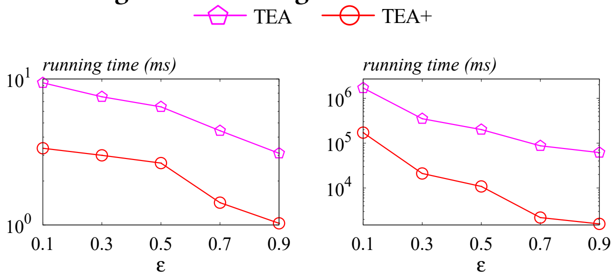


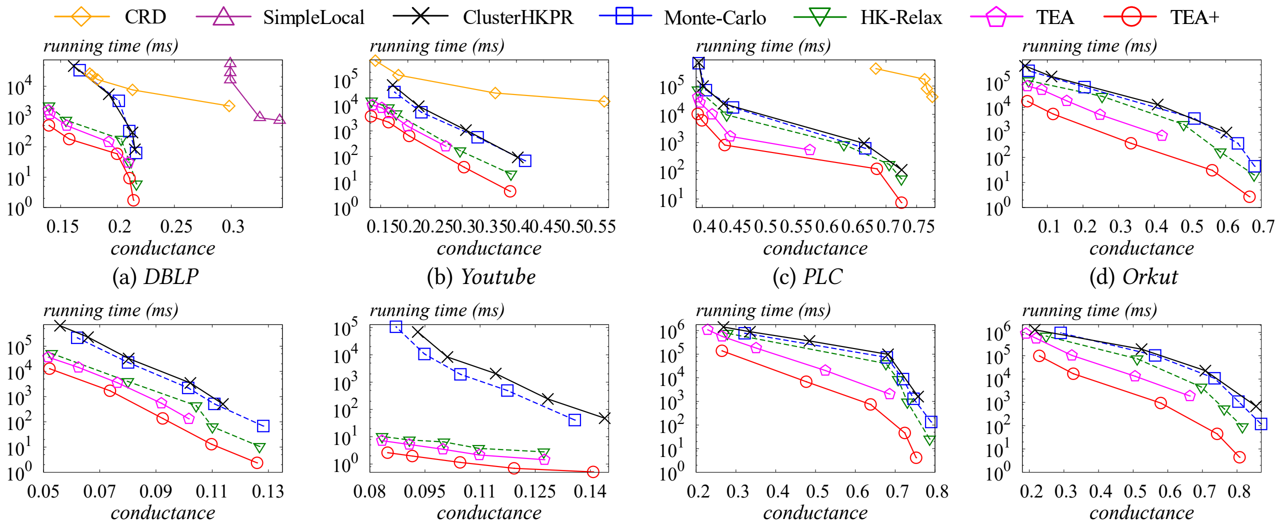
**Figure 1: Running time of TEA+ vs $c$.**



**(a) *3D-grid*** **(b) *Friendster***

**Figure 2: Running time vs $\epsilon_r$.**

Figure 2 reports the computational time of TEA and TEA+ on representative datasets when varying $\epsilon_r$ from 0.1 to 0.9 and fixing $\delta = 10^{-6}$. Results on other datasets are qualitatively similar and are reported in [1]. Observe that TEA+ always outperforms TEA markedly for all datasets. In particular, when the relative error threshold $\epsilon_r$ is large (e.g., $0.5 - 0.9$), TEA+ is one to two orders of magnitude faster than TEA on *Friendster*, but only around 5 times faster on *3D-grid*. This is caused by the fact that each node in *3D-grid* has six neighbors, thus residues will drop below the threshold quickly and both TEA and TEA+ require very few random walks. As we keep decreasing $\epsilon_r$, the gap between TEA and TEA+ is narrowed. Especially, when $\epsilon_r = 0.1$, TEA+ achieves around 5× to 10× speedup. The reasons are as follows. When the relative error thresholds are large, TEA+ only needs a small number of push operations and random walks due to the new termination conditions of HK-Push+ and residue reduction method compared to TEA. However, when the relative error thresholds are very small, the termination conditions of HK-Push+ are harder to satisfy and as a result incurs much higher costs to terminate. Furthermore, the residue reduction method is not able to reduce the number of random walks significantly since the residue sum is already small. Thus, both TEA and TEA+ perform many push

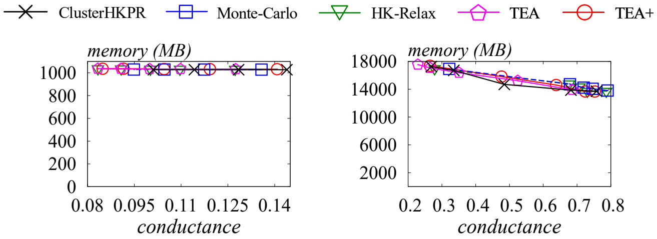


(a) *DBLP* (b) *Youtube* (c) *PLC* (d) *Orkut*

(e) *LiveJournal* (f) *3D-grid* (g) *Twitter* (h) *Friendster*

**Figure 3: Running time vs conductance for local clustering queries (best viewed in color).**

operations and random walks, and as a result, the speedup is modest. The results demonstrate the power of new termination conditions of HK-Push+ and residue reducation method in TEA+, especially when the error thresholds are not very small. *In summary,* TEA+ *outperforms* TEA *without sacrificing theoretical accuracies of HKPR values.*



**(a)** ***3D-grid*** **(b)** ***Twitter***

**Figure 4: Memory cost vs. conductance.**

## 7.4 Comparisons with Competitors

We compare TEA and TEA+ against ClusterHKPR, SimpleLocal [39], CRD [26], Monte-Carlo and HK-Relax in terms of clustering quality and efficiency (or memory overheads). Recall that Monte-Carlo is a random-walk-based approach as described in Section 3. Monte-Carlo accepts two thresholds $\epsilon_r$ and $\delta$, and a failure probability $p_f$ as inputs. It performs $\frac{2(1+\epsilon_r/3)\log(n/p_f)}{\epsilon_r^2 \cdot \delta}$ random walks from the seed node $s$ and returns a $(d, \epsilon_r, \delta)$-approximate HKPR vector for $s$ with probability at least $1-p_f$. HK-Relax ensures $\epsilon_a$ absolute error in each $\frac{\widehat{\rho}_s[v]}{d(v)}$, which is incomparable with the accuracy guarantees of other three methods, that is $(d, \epsilon_r, \delta)$-approximation guarantees. Hence, we do not compare all algorithms under the same theoretical accuracy guarantees. Instead, we evaluate each method by its empirical clustering quality (i.e., conductance) and empirical running time (or memory overheads) with various parameter settings and find the method that achieves the best trade-off in terms of clustering quality and running time. HK-Relax only has one internal parameter $\epsilon_a$. We vary it in $\{10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$ in our experiments. Since Monte-Carlo, TEA, and TEA+ have almost the same parameters, we set relative error threshold $\epsilon_r = 0.5$, and $\delta$ is varied in $\{2\times10^{-8}, 2\times10^{-7}, 2\times10^{-6}, 2\times10^{-5}, 2\times10^{-4}\}$ for all of them. ClusterHKPR has one internal parameter $\epsilon$, which is varied in $\{0.005, 0.01, 0.02, 0.05, 0.1\}$. We vary the locality parameter $\delta$ of SimpleLocal in $\{0.005, 0.01, 0.02, 0.05, 0.1\}$. In addition, we vary the number of iterations of CRD in $\{7, 10, 15, 20, 30\}$ and keep other parameters default. For fair comparison, we let the $x$-axis be the average conductance of the output clusters and the $y$-axis be the average running time (or memory overheads), depicting the empirical clustering quality and empirical efficiency (or memory overheads), respectively.

Figure 3 shows the average conductance of the output clusters and the the average running time when varying the aforementioned parameters for each algorithm. We can make the following observations. First, for each algorithm, when the error thresholds (i.e., $\epsilon_a$, $\epsilon$ and $\delta$) become smaller or the number of iterations increase, the conductance of the output clusters reduces (i.e., the quality of the output clusters improves), as well as the computational time goes up markedly, which accords with their theoretical time complexities. In particular, SimpleLocal incurs very high running time as well as poor cluster quality due to its high time complexity and the fact that it is mainly devised for recovering the cluster for a subset of nodes of the cluster rather than detecting a cluster for a single seed node. We also note that CRD shows a better performance than SimpleLocal. However, it is still much slower than HKPR-based methods. Hence, we omit the results of SimpleLocal on the remaining datasets and that of CRD on *Orkut, LiveJournal, 3D-grid, Twitter* and *Friendster*.

Second, we can see that Monte-Carlo and ClusterHKPR take several minutes to finish one local clustering query on all graph datasets in order to find clusters with small conductance. Hence, it is outperformed by HK-Relax, TEA and TEA+ by 1 to 3 orders of magnitude when they output clusters with almost the same conductance. This is due to the fact that Monte-Carlo and ClusterHKPR require performing a large number of random walks. In fact, this is consistent with the experimental results in prior work [12, 22]. Moreover, it can be observed that HK-Relax always runs faster than Monte-Carlo and ClusterHKPR and achieves more than 10× speedup on all datasets except *Orkut*, *Twitter*, and *Friendster*. To explain this phenomenon, recall that HK-Relax requires iteratively pushing residuals to $1 - K$-hop nodes from the seed node and $K$ is very large (see Section 6). However, on graphs with large average degrees (*Orkut*, *Twitter*, and *Friendster*) the number of push operations increases dramatically after several hops from the seed node.

Third, TEA+ outperforms HK-Relax by more than 10× speedup on *PLC*, *Orkut*, *Twitter*, and *Friendster*, and more than 4× speedup on the rest of graphs. The speedup is achieved by new termination conditions for HK-Push+ and the residue reduction method for reducing the number random walks. However, we note that the speedup on *DBLP*, *Youtube*, *LiveJournal* and *3D-grid* is not as significant as that on *PLC*, *Orkut*, *Twitter*, and *Friendster*. The reason is that these graphs either have large clustering coefficients [42] or small average degrees. The first one can also be observed in our experiments. With the same parameters as inputs to all three algorithms, the conductance values of output clusters from *PLC*, *Orkut*, *Twitter*, and *Friendster* are clearly greater than those from the remaining four graphs (i.e., *DBLP*, *Youtube*, *LiveJournal* and *3D-grid*). This implies that nodes in these four graphs are more likely to cluster together, and then residues on these graphs tend to be propagated within a small cluster of nodes when HK-Push+ is performed. Now consider the second reason. Recall that HK-Push+ iteratively distributes residues to neighbors evenly before any termination condition is satisfied. Since the average degrees are small, the residues that each node receives will be large. Additionally, it needs more iterations to distribute the residues to more nodes, which may not be done before termination. As a result of these two factors, a few nodes will hold large residues rather than many nodes holding small residues. Consequently, the residue reduction method in TEA+ fails to significantly reduce the number of random walks. Note that HK-Relax, TEA and TEA+ all terminate very quickly on *3D-grid* (less than 10 milliseconds), which is consistent with the observation in [22]. This is due to the fact that each node in *3D-grid* has six neighbors and the residues will drop below the residue threshold quickly after performing several rounds of push operations.

In addition, we also note that TEA fails to achieve considerable speedup compared with HK-Relax. Especially on *DBLP*, *Youtube*, *LiveJournal*, and *3D-grid*, TEA's performance degrades to the same level as that of HK-Relax. This is also caused by aforementioned high clustering coefficients and small average degrees of these graphs. TEA+ is around 4× faster than TEA on *Orkut*, *Twitter*, and *Friendster*. *In summary, our experiments demonstrate the power of new termination condition of* HK-Push+ *and residue reduction method, which reduce many push operations and random walks without sacrificing the cluster qualities.*

Figure 4 shows the memory overheads (including the space required to store the input graph) for two representative datasets by varying the error thresholds. The results for other datasets are qualitatively similar and are reported in [1]. First, we observe that the memory overheads on *Twitter* increase with the reduction in error thresholds as more space is required to store residues and HKPR values. However, memory overheads on *3D-grid* remain stable for all algorithms. As each node in *3D-grid* is connected to six neighbors, ClusterHKPR, Monte-Carlo, HK-Relax, TEA and TEA+ easily detect the large set of nodes around the seed nodes and the size of memory allocated for storing HKPR values for this set of nodes remains stable. Second, we observe that the memory overheads of all algorithms are roughly comparable because space usage is dominated by the storage of the input graph.

## 8 CONCLUSIONS

In this paper, we propose two novel heat-kernel-based local clustering algorithms, TEA and TEA+, for computing approximate HKPR values and local graph clustering efficiently. Our algorithms bridge deterministic graph traversal with Monte-Carlo random walks in a non-trivial way, thereby overcoming the drawbacks of both and achieving significant gain in performance in comparison to the state-of-the-art local clustering techniques. Our experiments demonstrate that TEA+ significantly outperforms the state-of-the-art heat-kernel-based algorithm by at least 4 times on small graphs and up to one order of magnitude on large graphs in terms of computational time when producing clusters with the same qualities.

## 9 ACKNOWLEDGEMENTS

This research is supported in part by MOE, Singapore under grant MOE2015-T2-2-069 and RG128/18 (M4012086.020), by NUS, Singapore under an SUG, by NTU, Singapore under grant M4082311.020, and by National Natural Science Foundation of China (No. 61832017, No. 61772346 and U1809206).

# REFERENCES

[1] *Technical Report.* Available at: http://arxiv.org/abs/1904.02707.

[2] http://snap.stanford.edu.

[3] Reid Andersen, Fan Chung, Kevin Lang. 2006. Local Graph Partitioning Using Pagerank Vectors. In *FOCS*, pages 475-486.

[4] Reid Andersen, Yuval Peres. 2009. Finding Sparse Cuts Locally Using Evolving Sets. In *STOC*, pages 235-244.

[5] Haim Avron, Lior Horesh. 2015. Community Detection Using Time-dependent Personalized Pagerank. In *ICML*, pages 1795-1803.

[6] Siddhartha Banerjee, Peter Lofgren. 2017. Fast Bidirectional Probability Estimation in Markov Models. In *NIPS*, pages 1423-1431.

[7] Bela Bollobas. Modern Graph Theory. 1998.

[8] Fan Chung. 2007. The Heat Kernel as the Pagerank of a Graph. In *PNAS*, pages 19735-19740.

[9] Fan Chung. 2009. A Local Graph Partitioning Algorithm Using Heat Kernel Pagerank. *Internet Mathematics*, pages 315-330.

[10] Fan Chung, Linyuan Lu. 2006. Concentration Inequalities and Martingale Inequalities: A Survey. *Internet Mathematics*, pages 79-127.

[11] Fan Chung, Olivia Simpson. 2014. Computing Heat Kernel Pagerank and a Local Clustering Algorithm. *IWOCA*, pages 110-121.

[12] Fan Chung, Olivia Simpson. 2015. Computing Heat Kernel Pagerank and a Local Clustering Algorithm. *arXiv preprint arXiv:1503.03155.*

[13] Fan Chung, Olivia Simpson. 2015. Distributed Algorithms for Finding Local Clusters Using Heat Kernel Pagerank. In *WAW*, pages 177-189.

[14] Persi Diaconis, James Allen Fill. 1990. Strong Stationary Times via a New Form of Duality. *The Annals of Probability.*

[15] Glen Jeh, Jennifer Widom. 2003. Scaling Personalized Web Search. In *WWW*, pages 271-279.

[16] Kwak *et al.* 2010. What is Twitter, a Social Network or a News Media? In *WWW*, pages 591-600.

[17] Kyle Kloster, David Gleich. 2014. Heat Kernel Based Community Detection. In *KDD*, pages 1386-1395.

[18] Liao *et al.* 2009. IsoRankN: Spectral Methods for Global Alignment of Multiple Protein Networks. In *Bioinformatics*, pages 253-258.

[19] Lofgren *et al.* 2016. Personalized Pagerank Estimation and Search: A Bidirectional Approach. In *WSDM*, pages 163-172.

[20] Page *et al.* 1999. The PageRank Citation Ranking: Bringing Order to the Web.

[21] Daniel A Spielman, Shang-Hua Teng. 2004. Nearly-linear Time Algorithms for Graph Partitioning, Graph sparsification, and Solving Linear Systems. In *STOC*, pages 81-90.

[22] Shun *et al.* 2016. Parallel Local Graph Clustering. In *VLDB*, pages 1041-1052.

[23] Tolliver *et al.* 2006. Graph Partitioning by Spectral Rounding: Applications in Image Segmentation and Clustering. In *CVPR*, pages 1053-1060.

[24] Wang et al. 2015. Community Detection In Social Networks: An In-depth Benchmarking Study with a Procedure-oriented Framework. In *VLDB*, pages 998-1009.

[25] Wang *et al.* 2016. HubPPR: Effective Indexing for Approximate Personalized Pagerank. In *VLDB*, pages 205-216.

[26] Wang *et al.* 2017. Capacity Releasing Diffusion for Speed and Locality. In *ICML*, pages 3598-3607.

[27] Wang *et al.* 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *KDD*, pages 505-514.

[28] Wei *et al.* 2018. Topppr: Top-k Personalized Pagerank Queries with Precision Guarantees on Large Graphs. In *SIGMOD*, pages 441-456.

[29] Pedro F Felzenszwalb, Daniel P Huttenlocher. 2004. Efficient Graph-based Image Segmentation. In *IJCV*, pages 167-181.

[30] Santo Fortunato. 2010. Community Detection in Graphs. *Physics Reports*, pages 75-174.

[31] Gharan *et al.* 2012. Approximating the Expansion Profile and Almost Optimal Local Graph Clustering. In *FOCS*, pages 187-196.

[32] Taher H Haveliwala. 2002. Topic-sensitive Pagerank. In *WWW*, pages 517-526.

[33] Kalervo Järvelin, Jaana Kekäläinen. 2000. IR Evaluation Methods for Retrieving Highly Relevant Documents. In *SIGIR*, pages 41-48.

[34] Eugene L Lawler. 2001. Combinatorial Optimization: Networks and Matroids.

[35] Leskovec *et al.* 2010. Empirical Comparison of Algorithms for Network Community Detection. In *WWW*, pages 631-640.

[36] Lorenzo Orecchia, Zeyuan Allen Zhu. 2014. Flow-based Algorithms for Local Graph Clustering. In *SODA*, pages 1267-1286.

[37] Pascal Pons, Matthieu Latapy. 2005. Computing Communities in Large Networks Using Random Walks. In *ISCIS*, pages 284-293.

[38] Daniel A Spielman, Shang-Hua Teng. 2013. A Local Clustering Algorithm for Massive Graphs and Its Application to Nearly Linear Time graph partitioning. In *SICOMP*, pages 1-26.

[39] Veldt *et al.* 2016. A Simple and Strongly-local Flow-based Method for Cut Improvement. In *ICML*, pages 1938-1947.

[40] Konstantin Voevodski, Shang-Hua Teng, Yu Xia. 2009. Finding Local Communities in Protein Networks. In *BMC Bioinformatics*, page 297.

[41] Alastair J. Walker. 1974. New Fast Method for Generating Discrete Random Numbers with Arbitrary Frequency Distributions. *Electronics Letters*, page 127-128.

[42] Jaewon Yang, Jure Leskovec. 2012. Defining and Evaluating Network Communities Based on Ground-truth. In *KDD Workshop.*

[43] Zhu *et al.* 2013. A Local Algorithm for Finding Well-Connected Clusters. In *ICML*, pages 396-404.

# A PROOFS

## A.1 Proof of Theorem 1

Proof. Let $Y_i$ be as defined in the context of Equation (9), and let $Y = \sum_{i=1}^{n_r} Y_i$. By Line 12 of TEA, $\widehat{\boldsymbol{\rho}}_s[v] = \mathbf{q}_s[v] + \frac{Y \cdot \alpha}{n_r}$. By Equation (10), the expectation of $Y$ is

$$\mathbb{E}[Y] = \mathbb{E}[\sum_{i=1}^{n_r} Y_i] = \frac{n_r}{\alpha}\left(\boldsymbol{\rho}_s[v] - \mathbf{q}_s[v]\right) \leq \frac{n_r}{\alpha} \cdot \boldsymbol{\rho}_s[v].$$

Let $\lambda = \frac{n_r \epsilon_r}{\alpha} \cdot \boldsymbol{\rho}_s[v]$ and $n_r$ be as defined in Line 8 of TEA. By the Chernoff bound (see Lemma 5), for any node $v$ in $V$ with $\boldsymbol{\rho}_s[v] > d(v) \cdot \delta$, we have

$$\begin{aligned} \mathbb{P}[|Y - \mathbb{E}[Y]| \geq \lambda] &= \mathbb{P}\left[\left|\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v]\right| \geq \epsilon_r \cdot \boldsymbol{\rho}_s[v]\right] \\ &\leq \exp\left(-\frac{n_r \cdot \epsilon_r^2 \cdot \boldsymbol{\rho}_s[v]}{2\alpha \cdot (1+\epsilon_r/3)}\right) \leq (p'_f)^{d(v)}. \end{aligned}$$

On the other hand, let $\lambda = \frac{n_r \epsilon_r \delta d(v)}{\alpha}$. Then, for any node $v$ in $V$ with $\boldsymbol{\rho}_s[v] \leq d(v) \cdot \delta$,

$$\begin{aligned} \mathbb{P}[|Y - \mathbb{E}[Y]| \geq \lambda] &= \mathbb{P}\left[\left|\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v]\right| \geq d(v) \cdot \epsilon_r \delta\right] \\ &\leq \exp\left(-\frac{n_r \cdot \epsilon_r^2 \delta^2 d^2(v)}{2\alpha(1+\epsilon_r/3) \cdot \boldsymbol{\rho}_s[v]}\right) \leq (p'_f)^{d(v)}. \end{aligned}$$

By the union bound, for $V_1 = \{v | v \in V \ s.t.\ \boldsymbol{\rho}_s[v] > d(v) \cdot \delta\}$, we have

$$\mathbb{P}\left[\bigcup_{v \in V_1}\left\{\left|\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v]\right| \geq \epsilon_r \cdot \boldsymbol{\rho}_s[v]\right\}\right] \leq \sum_{v \in V_1} (p'_f)^{d(v)},$$

and for $V_2 = \{v | v \in V \ s.t.\ \boldsymbol{\rho}_s[v] \leq d(v) \cdot \delta\}$, we have

$$\mathbb{P}\left[\bigcup_{v \in V_2}\left\{\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \geq \epsilon_r \delta\right\}\right] \leq \sum_{v \in V_2} (p'_f)^{d(v)}.$$

Therefore, we have the following results, respectively. With probability at least $1 - \sum_{v \in V} (p'_f)^{d(v)}$, for every node $v$ in $V$ with $\frac{\boldsymbol{\rho}_s[v]}{d(v)} > \delta$, $\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \le \epsilon_r \cdot \frac{\boldsymbol{\rho}_s[v]}{d(v)}$, and for every node $v$ in $V$ with $\frac{\boldsymbol{\rho}_s[v]}{d(v)} \le \delta$, $\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \le \epsilon_r \delta$.

By the definition of $p'_f$ in Equation (6), if $\sum_{v \in V} p_f^{d(v)-1} \le 1$, then $p'_f = p_f$, which leads to $\sum_{v \in V} (p'_f)^{d(v)} = \sum_{v \in V} p_f^{d(v)} \le p_f$; otherwise, $p'_f = \frac{p_f}{\sum_{v \in V} p_f^{d(v)-1}}$, hence $\sum_{v \in V} (p'_f)^{d(v)} < \sum_{v \in V} \frac{p_f^{d(v)}}{\sum_{v \in V} p_f^{d(v)-1}} = p_f$. Namely, $\widehat{\boldsymbol{\rho}}_s$ is a $(d, \epsilon_r, \delta)$-approximate HKPR vector with probability at least $1 - p_f$. □

## A.2 Proof of Theorem 2

Proof. By Lemma 6 and the definition of $\mathbf{h}_u^{(k)}[v]$ in Equation 5, we have $\frac{\mathbf{h}_u^{(k)}[v]}{d(v)} = \frac{\mathbf{h}_v^{(k)}[u]}{d(u)}$. Then we can rewrite Equation (4) as follows

$$\begin{aligned}\boldsymbol{\rho}_s[v] - \mathbf{q}_s[v] &= d(v) \cdot \textstyle\sum_{u \in V} \sum_{k=0}^{K} \left[\frac{\mathbf{r}_s^{(k)}[u]}{d(u)} \cdot \mathbf{h}_v^{(k)}[u]\right] \\ &\le d(v) \cdot \textstyle\sum_{k=0}^{K} \left[\max_{u \in V} \left\{\frac{\mathbf{r}_s^{(k)}[u]}{d(u)}\right\} \cdot \sum_{u \in V} \mathbf{h}_v^{(k)}[u]\right].\end{aligned}$$

Now we prove that $\sum_{u \in V} \mathbf{h}_v^{(k)}[u] = 1$ for any node $v \in V$ and $k \in [0, K]$.

$$\begin{aligned}\textstyle\sum_{u \in V} \mathbf{h}_v^{(k)}[u] &= \textstyle\sum_{\ell=0}^{\infty} \left[\frac{\eta(k+\ell)}{\psi(k)} \sum_{u \in V} \mathbf{P}^{\ell}[v, u]\right] \\ &= \textstyle\sum_{\ell=0}^{\infty} \frac{\eta(k+\ell)}{\psi(k)} = 1.\end{aligned}$$

Hence, $\boldsymbol{\rho}_s[v] - \mathbf{q}_s[v] \le d(v) \cdot \sum_{k=0}^{K} \max_{u \in V} \left\{\frac{\mathbf{r}_s^{(k)}[u]}{d(u)}\right\}$. Once Inequality (11) held, we have $\frac{\boldsymbol{\rho}_s[v]}{d(v)} - \frac{\mathbf{q}_s[v]}{d(v)} \le \epsilon_a$, which completes the proof. □

## A.3 Proof of Theorem 3

Proof. First, consider the case where TEA+ terminates at Line 7, i.e., $\sum_{k=0}^{K} \max_{u \in V} \left\{\frac{\mathbf{r}_s^{(k)}[u]}{d(u)}\right\} \le \epsilon_a$. In that case, by Theorem 2, for any node $v \in V$, $\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \le \epsilon_a = \epsilon_r \cdot \delta$. This indicates that, for any node $v$ with $\boldsymbol{\rho}_s[v] > d(v) \cdot \delta$, $\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \le \epsilon_r \cdot \frac{\boldsymbol{\rho}_s[v]}{d(v)}$. In addition, for any node $v$ with $\boldsymbol{\rho}_s[v] \le d(v) \cdot \delta$, $\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \le \epsilon_r \cdot \delta$. Thus, $\widehat{\boldsymbol{\rho}}_s$ is a $(d, \epsilon_r, \delta)$-approximate HKPR vector.

Now consider the case where TEA+ does not terminate at Line 7. Let $Y_i$ be as defined in the context of Equation (9), and let $Y = \sum_{i=1}^{n_r} Y_i$. By Lines 17 and 19 of TEA+,

$$\widehat{\boldsymbol{\rho}}_s[v] = \mathbf{q}_s[v] + \frac{Y \cdot \alpha}{n_r} + \frac{\epsilon_r \delta}{2} \cdot d(v).$$

By Equation (10), the expectation of $Y$ is

$$\mathbb{E}[Y] = \mathbb{E}[\textstyle\sum_{i=1}^{n_r} Y_i] = \frac{n_r}{\alpha}\left(\boldsymbol{\rho}_s[v] - \mathbf{q}_s[v] - \mathbf{b}_s[v]\right) \le \frac{n_r}{\alpha} \cdot \boldsymbol{\rho}_s[v].$$

Let $\lambda = \frac{n_r \epsilon_r}{2 \cdot \alpha} \cdot \boldsymbol{\rho}_s[v]$ and $n_r$ be as defined in Line 13 of TEA+. By the Chernoff bound (see Lemma 5), for any node $v$ in $V$ with $\boldsymbol{\rho}_s[v] > d(v) \cdot \delta$, we have

$$\begin{aligned}&\mathbb{P}[|Y - \mathbb{E}[Y]| \ge \lambda] \\ &= \mathbb{P}\left[\left|\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v] + \mathbf{b}_s[v] - \tfrac{\epsilon_r \delta}{2} \cdot d(v)\right| \ge \tfrac{\epsilon_r}{2} \cdot \boldsymbol{\rho}_s[v]\right] \\ &\le \exp\left(-\tfrac{n_r \cdot \epsilon_r^2 \cdot \boldsymbol{\rho}_s[v]}{8\alpha \cdot (1 + \epsilon_r/6)}\right) \le (p'_f)^{d(v)}.\end{aligned}$$

On the other hand, let $\lambda = \frac{n_r \epsilon_r \delta d(v)}{2\alpha}$. Then, for any node $v$ in $V$ with $\boldsymbol{\rho}_s[v] \le d(v) \cdot \delta$, we have

$$\begin{aligned}&\mathbb{P}[|Y - \mathbb{E}[Y]| \ge \lambda] \\ &= \mathbb{P}\left[\left|\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v] + \mathbf{b}_s[v] - \tfrac{\epsilon_r \delta}{2} \cdot d(v)\right| \ge \tfrac{\epsilon_r \delta}{2} \cdot d(v)\right] \\ &\le \exp\left(-\tfrac{n_r \cdot \epsilon_r^2 \delta^2 d^2(v)}{8\alpha(1 + \epsilon_r/6) \cdot \boldsymbol{\rho}_s[v]}\right) \le (p'_f)^{d(v)}.\end{aligned}$$

By union bound, for $V_1 = \{v | v \in V\ s.t.\ \boldsymbol{\rho}_s[v] > d(v) \cdot \delta\}$, we have

$$\begin{aligned}&\mathbb{P}\left[\textstyle\bigcup_{v \in V_1} \left\{\left|\frac{\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v] + \mathbf{b}_s[v]}{d(v)} - \frac{\epsilon_r \delta}{2}\right| \ge \frac{\epsilon_r}{2} \cdot \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right\}\right] \\ &\le \textstyle\sum_{v \in V_1} (p'_f)^{d(v)},\end{aligned}$$

and for $V_2 = \{v | v \in V\ s.t.\ \boldsymbol{\rho}_s[v] \le d(v) \cdot \delta\}$, we have

$$\mathbb{P}\left[\textstyle\bigcup_{v \in V_2} \left\{\left|\frac{\widehat{\boldsymbol{\rho}}_s[v] - \boldsymbol{\rho}_s[v] + \mathbf{b}_s[v]}{d(v)} - \frac{\epsilon_r \delta}{2}\right| \ge \frac{\epsilon_r \delta}{2}\right\}\right] \le \textstyle\sum_{v \in V_2} (p'_f)^{d(v)}.$$

By Inequality (19), $\left|\frac{\mathbf{b}_s[v]}{d(v)} - \frac{\epsilon_r \delta}{2}\right| \le \frac{\epsilon_r \delta}{2}$. Then, we have the following results, respectively. With probability at least $1 - \sum_{v \in V} (p'_f)^{d(v)}$, for every node $v$ in $V$ with $\boldsymbol{\rho}_s[v] > d(v) \cdot \delta$,

$$\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \le \frac{\epsilon_r}{2} \cdot \frac{\boldsymbol{\rho}_s[v]}{d(v)} + \frac{\epsilon_r \delta}{2} \le \epsilon_r \cdot \frac{\boldsymbol{\rho}_s[v]}{d(v)},$$

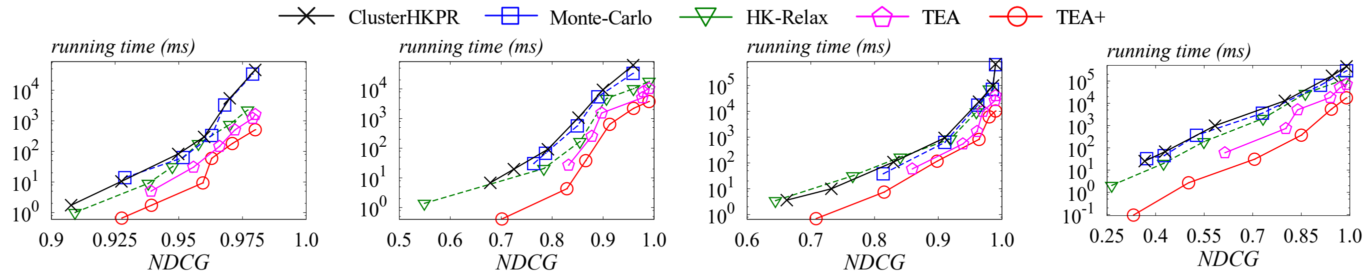
and for every node $v$ in $V$ with $\boldsymbol{\rho}_s[v] \le d(v) \cdot \delta$,

$$\left|\frac{\widehat{\boldsymbol{\rho}}_s[v]}{d(v)} - \frac{\boldsymbol{\rho}_s[v]}{d(v)}\right| \le \epsilon_r \delta.$$

By the definition of $p'_f$ in Equation (6), the total failure probability will be at most $\sum_{v \in V} (p'_f)^{d(v)} \le p_f$. Therefore, $\widehat{\boldsymbol{\rho}}_s$ is a $(d, \epsilon_r, \delta)$-approximate HKPR vector with probability at least $1 - p_f$. □

# B ADDITIONAL EXPERIMENTS

**Ranking Accuracy of Normalized HKPR.** In this set of experiments, we evaluate the accuracy and efficiency of each method for computing normalized HKPR values (e.g., $\frac{\boldsymbol{\rho}_s[v]}{d(v)}$). First, we randomly select 50 seed nodes and apply the power method [20] with 40 iterations to compute the ground-truth normalized HKPR values (we omit the large datasets due to time and memory limitations). Following the experimental settings in Section 7.1, we run HK-Relax, ClusterHKPR, Monte-Carlo, TEA and TEA+ to generate normalized HKPR values for the selected seed nodes with varied error thresholds. Specifically, we vary



(a) *DBLP* (b) *Youtube* (c) *PLC* (d) *Orkut*

**Figure 5: Running time vs. NDCG for computing normalized HKPR (best viewed in color).**

$\epsilon$ in $\{10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ for HK-Relax, $\epsilon$ in $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.3\}$ for ClusterHKPR, and we set $\epsilon_r = 0.5$ and $\delta$ is varied in $\{2\times10^{-8}, 2\times10^{-7}, 2\times10^{-6}, 2\times10^{-5}, 2\times10^{-4}, 2\times10^{-3}\}$ for Monte-Carlo, TEA and TEA+, respectively. Then, we evaluate the accuracy of each method by using *Normalized Discounted Cumulative Gain (NDCG)* [33], which is a classic metric for evaluating ranking results.

Figure 5 reports the performance of each method on four datasets. We can make the following observations. First, as we reduce the error thresholds, both the running time and NDCG of each method increase markedly, which is consistent with their theoretical guarantees. Second, TEA+ consistently incurs least running time while achieving the same NDCG compared to the competing methods. In addition, TEA is $2\times -8\times$ slower than TEA+ while HK-Relax runs even more slowly. Especially on *PLC* and *Orkut* datasets, HK-Relax's performance degrades to the same level of ClusterHKPR and Monte-Carlo. Although ClusterHKPR and Monte-Carlo also provide relative-error guarantees, they still incur the highest overheads because they require a large number of random walks. Third, we note that the efficiency and ranking accuracy results accord with the efficiency and clustering quality results reported in Section 7.4. This demonstrates the relationship between ranking accuracy of normalized HKPR and the quality of HKPR-based clustering algorithms, emphasizing why our methods produce clusters with smaller conductance than the competing ones.

**Clusters Produced vs. Ground-truth.** We collect the top 5,000 ground-truth communities in *DBLP*, *Youtube*, *LiveJournal* and *Orkut* datasets from [2]. We select 100 seed nodes from 100 known communities of size greater than 100 randomly as the query set. For all algorithms, we vary $t$ from 3 to 10 ($t > 10$ would give us clusters with substantially lower quality) and their error thresholds respectively to produce clusters with highest average $F_1$-measure (i.e., harmonic mean of precision and recall). More specifically, we vary $\epsilon$ from 0.005 to 0.35 for ClusterHKPR, $\epsilon$ from $10^{-8}$ to $10^{-1}$ for HK-Relax. In addition, we fix $\epsilon_r = 0.5$ and vary $\delta$ from $10^{-8}$ to $10^{-1}$ for Monte-Carlo, TEA and TEA+.

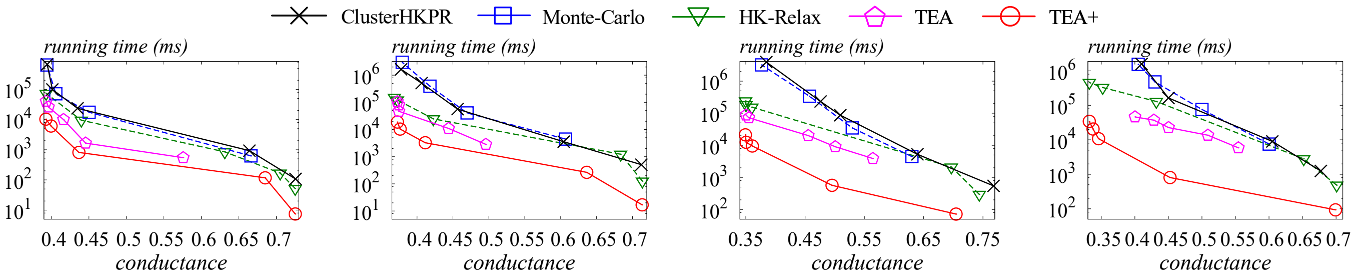
Table 5 reports the highest $F_1$ measure of each algorithm and their corresponding running times. TEA+ consistently produces clusters with the best average $F_1$-measures and least running times for all datasets except *DBLP*. On *DBLP*, TEA produces clusters with the best average $F_1$-measure while TEA+ produces clusters with slightly smaller $F_1$ measure but significantly faster. We also observe that ClusterHKPR and Monte-Carlo generate very similar results for all datasets. They run significantly slower than TEA and TEA+ and also produce clusters with slightly smaller average $F_1$-measures than our methods. In addition, HK-Relax has the worst performance on most datasets. The only exception occurs on *Orkut*, where it produces clusters with the second best $F_1$-measure but 4× slower than TEA+.

**Sensitivity Analysis to the Subgraph Characteristics.** Next, we study the impact of query sets generated from subgraphs of different characteristics on clustering quality and efficiency. First, from each dataset of *Youtube*, *PLC* and *Orkut*, we select 250 subgraphs with different densities [34] randomly. Then we sort the subgraphs by their densities in descending order (denoted as $\{SG_1, SG_2, \cdots, SG_{250}\}$). We pick 50 nodes from $SG_1, \cdots, SG_{50}$ respectively to form a query set referred to as *high-density* seed nodes, 50 nodes from $SG_{100}, \cdots, SG_{150}$ respectively as *medium-density* seed nodes, and 50 nodes from $SG_{200}, \cdots, SG_{250}$ respectively as *low-density* seed nodes. We run ClusterHKPR, Monte-Carlo, HK-Relax, TEA and TEA+ with the same parameter settings as in Section 7.4 on these three query sets.

Figure 7 plots the average conductance of the output clusters and the average running times of all algorithms under different error thresholds for the three query sets. We report the results on *Youtube* and *PLC* here. The results on the remaining datasets are qualitatively similar and are reported in [1]. We can make the following observations. First, TEA and TEA+ are consistently faster than the existing approaches for all query sets. Second, the conductance values of each graph in Figures 7e and 7f are higher than the rest. This is because subgraphs with high densities have low conductance. Also, both ClusterHKPR and Monte-Carlo show similar results on all query sets for all datasets whereas HK-Relax, TEA and TEA+ are sensitive to the subgraph densities. Since

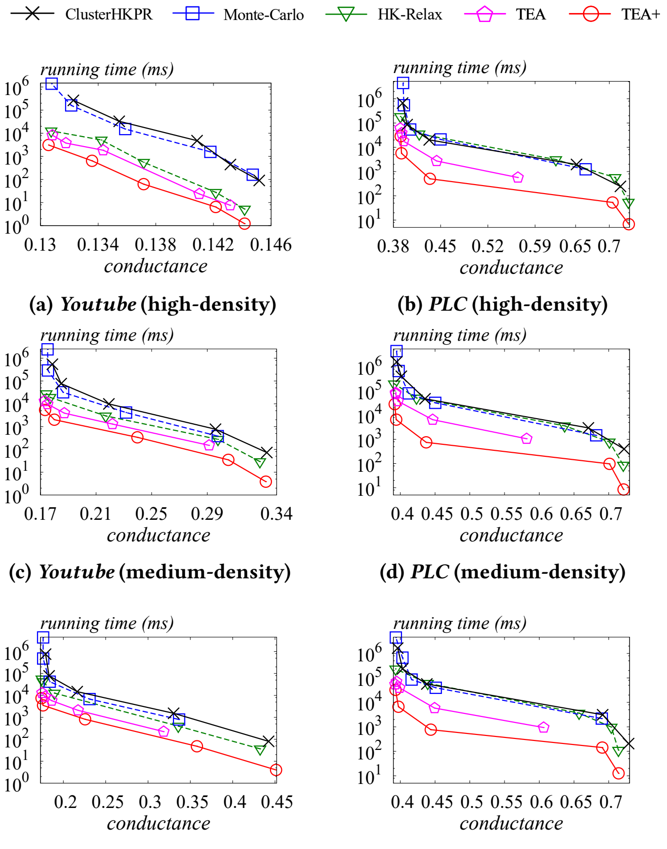
**Table 5: The result of evaluating all algorithms on finding real-world communities.**

| Data | $F_1$-measure | | | | | Running Time (ms) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ClusterHKPR | Monte-Carlo | HK-Relax | TEA | TEA+ | ClusterHKPR | Monte-Carlo | HK-Relax | TEA | TEA+ |
| *DBLP* | 0.13655 | 0.13631 | 0.13592 | **0.13679** | 0.136699 | 3053.95 | 2891.64 | 297.78 | 176 | 109.66 |
| *Youtube* | 0.10113 | 0.10097 | 0.09858 | 0.10133 | **0.10334** | 7.76 | 7.2 | 8.11 | 2.49 | 2 |
| *LiveJournal* | 0.64644 | 0.65105 | 0.64516 | 0.64959 | **0.67** | 1.34665 | 1.2 | 3.57 | 0.55 | 0.29 |
| *Orkut* | 0.18497 | 0.18464 | 0.19375 | 0.19333 | **0.19636** | 29.95 | 29.35 | 62.17 | 24.78 | 14.86 |



(a) $t = 5$ (b) $t = 10$ (c) $t = 20$ (d) $t = 40$

**Figure 6: Effect of heat constant $t$ on *PLC* (best viewed in color).**



**(a) *Youtube* (high-density)** **(b) *PLC* (high-density)**

**(c) *Youtube* (medium-density)** **(d) *PLC* (medium-density)**

**(e) *Youtube* (low-density)** **(f) *PLC* (low-density)**

**Figure 7: Effect of subgraph densities.**

seed nodes picked from subgraphs with high densities would have many neighbors, the residues in HK-Relax, TEA and TEA+ will drop quickly as push operations are performed, making them terminate quickly.

**Effects of Heat Constant $t$.** Lastly, we investigate the impact of the heat constant $t$. Using the same parameter settings and query set in Section 7.4, we run all algorithms on *DBLP* and *PLC* datasets by varying $t$ in $\{5, 10, 20, 40\}$. Figure 6 plots the average running time and average conductance of the output clusters of each algorithm on *PLC*. The results are qualitatively similar on *DBLP* and are reported in [1]. Observe that the running time of each algorithm increases as we increase $t$, which is consistent with their time complexities. ClusterHKPR and Monte-Carlo are the slowest as $t$ changes. We further observe that the advantage of TEA+ over competing methods is more prominent as $t$ becomes larger. More specifically, TEA+ is around 8 times faster than HK-Relax when $t = 5$ and the speedup goes up to two orders of magnitude when $t = 40$. In addition, we find that the conductance values of clusters produced by each algorithm with larger $t$ are smaller than those with smaller $t$. This shows that we can obtain clusters with small conductance by choose a large $t$. However, our *"Clusters Produced vs. Ground Truth"* experiment reveals that clusters produced by all algorithms with a large $t$ are very different from the ground-truth. This is because algorithms with a large $t$ tend to give us a cluster of nodes that are far from the seed node. As a result, choosing a good $t$ is paramount for finding high quality clusters.