Edge-based Local Push for Personalized PageRank

Hanzhi Wang
Renmin University of China
Beijing, China
hanzhi_wang@ruc.edu.cn

Zhewei Wei*
Renmin University of China
Beijing, China
zhewei@ruc.edu.cn

Junhao Gan
University of Melbourne
Melbourne, Australia
junhao.gan@unimelb.edu.au

Ye Yuan
Beijing Institute of Technology
Beijing, China
yuan-ye@bit.edu.cn

Xiaoyong Du
Renmin University of China
Beijing, China
duyong@ruc.edu.cn

Ji-Rong Wen
Renmin University of China
Beijing, China
jrwen@ruc.edu.cn

ABSTRACT

Personalized PageRank (PPR) is a popular node proximity metric in graph mining and network research. A single-source PPR (SSPPR) query asks for the PPR value of each node on the graph. Due to its importance and wide applications, decades of efforts have been devoted to the efficient processing of SSPPR queries. Among existing algorithms, *LocalPush* is a fundamental method for SSPPR queries and serves as a cornerstone for subsequent algorithms. In *LocalPush*, a *push* operation is a crucial primitive operation, which distributes the probability at a node u to ALL u's neighbors via the corresponding edges. Although this *push* operation works well on *unweighted* graphs, unfortunately, it can be rather inefficient on *weighted* graphs. In particular, on *unbalanced* weighted graphs where only a few of these edges take the majority of the total weight among them, the *push* operation would have to distribute "insignificant" probabilities along those edges which just take the minor weights, resulting in expensive overhead.

To resolve this issue, in this paper, we propose the *EdgePush* algorithm, a novel method for computing SSPPR queries on weighted graphs. *EdgePush* decomposes the aforementioned *push* operations in *edge-based push*, allowing the algorithm to operate at the edge level granularity. As a result, it can flexibly distribute the probabilities according to edge weights. Furthermore, our *EdgePush* allows a fine-grained termination threshold for each individual edge, leading to a superior complexity over *LocalPush*. Notably, we prove that *EdgePush* improves the theoretical query cost of *LocalPush* by an order of up to $O(n)$ when the graph's weights are *unbalanced*. Our experimental results demonstrate that *EdgePush* significantly outperforms state-of-the-art baselines in terms of query efficiency on large motif-based and real-world weighted graphs.

PVLDB Reference Format:
Hanzhi Wang, Zhewei Wei, Junhao Gan, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Edge-based Local Push for Personalized PageRank. PVLDB, 15(7): 1376-1389, 2022.
doi:10.14778/3523210.3523216

*Zhewei Wei is the corresponding author.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 15, No. 7 ISSN 2150-8097.
doi:10.14778/3523210.3523216

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/wanghzccls/EdgePush.

1 INTRODUCTION

Personalized PageRank (PPR), as a variant of PageRank [46], has become a classic node proximity measure. It effectively captures the relative importance of all the nodes with respect to a source node in a graph. One particular interest is the single-source PPR (SSPPR) query. Given a source node s in a graph $G = (V, E)$ with n nodes and m edges, the SSPPR query aims to return an SSPPR vector $\boldsymbol{\pi} \in \mathbb{R}^n$, where $\boldsymbol{\pi}(u)$ denotes the PPR value of node $u \in V$ with respect to the source node s. We can consider the SSPPR vector $\boldsymbol{\pi}$ as a probability distribution, with $\boldsymbol{\pi}(u)$ defined as the probability that an α-random walk starting from the source node s stops at node u. Specifically, the α-random walk [46] represents a random walk process that at each step, the walk either moves to a random neighbor with probability $1 - \alpha$, or stops at the current node with probability α. The teleport probability α is a constant in $(0, 1)$.

SSPPR queries has been widely adopted in various graph mining and network analysis tasks. For example, the seminal local clustering paper [10] and its variant [22, 65] identify clusters based on the SSPPR queries with the seed node as the source node. Additionally, the recommendations in social networks employ SSPPR values to evaluate the relative importance of other users regarding the target user, such as the Point-of-Interest recommendation [26], the connection prediction [12], the topical experts finding application [34] and the Who-To-Follow recommendation in Twitter [27]. Recently, several graph representation learning tasks [14, 17, 30, 68] compute SSPPR queries to propagate initial node features in the graph.

In this paper, we focus on efficient SSPPR queries on *weighted* graphs. Weighted graphs are extremely common in real life, where the weight of each edge indicates the distance, similarity or other strength measures of the relationships between two nodes. Various real applications are in dire need of the SSPPR results on weighted graphs. For instance, the personalized ranking results incorporating user preference or feedback embedded in the edge weight are highly valued in social network [20, 25, 57]. Additionally, to rank web pages by SSPPR queries, taking into account the importance of pages' links shows increasingly significance for the performance of page ranking [58]. In the local clustering application, computing SSPPR queries on motif-based weighted graphs[1] can effectively

[1] A motif is defined as a small subgraph (e.g. a triangle).

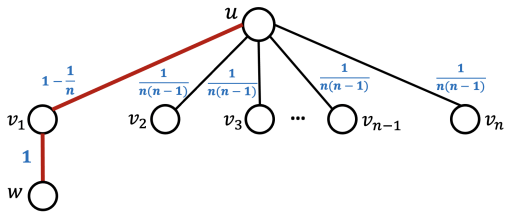

Figure 1: A bad case for the *LocalPush*. The number on each edge is the edge weight.

capture the high-order information of network structure which is crucial to the clustering quality [65].

Despite the large-scale applications of SSPPR queries on weighted graphs, this topic are less studied in literature due to its hardness. The state-of-the-art algorithm is MAPPR [65], which is a version of *LocalPush* on weighted graphs. *LocalPush* [10] is a crucial and fundamental method for SSPPR queries, which has been regarded as a cornerstone method for advanced developments [22, 28, 54–56]. The main idea of *LocalPush* is to approximate SSPPR results by deterministically *pushing* the probabilities on the graph. The *push* operation in *LocalPush* restricts the computation in a local manner, which achieves remarkable scalability on *unweighted graphs*. Unfortunately, although *LocalPush* works well on unweighted graphs, it can be rather ineffective on weighted graphs, leading to excessive time consumption.

***LocalPush*'s Limitation on Weighted Graphs.** As a crucial primitive operation in *LocalPush*, the *push* operation pushes probability mass from the current node to *all* its neighbors. Whenever the push operation on a node is invoked, it has to touch all the edges incident on the node. While this push strategy works fine on unweighted graphs, unfortunately, it has evident drawbacks on weighted graphs. When the weights of a node's edges are *unbalanced* where only a small number of edges taking a majority portion of the total weight among them, the push operation has to spend a significant cost on just pushing a tiny probability mass, resulting in severe overhead.

Figure 1 shows a toy example of the bad case for *LocalPush*. Consider node u whose total weight of edges is 1. There is an edge (u, v_1) taking a weight $1 - \frac{1}{n}$, merely the total weight, and all the others just share $\frac{1}{n}$ together. When a *push* operation on u is performed, it requires a cost of $n - 1$ just on pushing an extremely tiny probability mass for those "insignificant" edges. As a result, the push operation is extremely inefficient on such severely unbalanced weighted graphs.

It's worth to mention that weighted graphs with severely unbalanced edge weights are common in many real-world applications. Let's take the affinity graph as an example. Affinity graphs are frequently used in a variety of practical tasks [44, 51, 59, 60, 62, 64, 68, 69] to model the affinities between pairwise data points. Nodes in affinity graphs represent high dimensional data points, i.e. $V = \{x_1, ..., x_n\}$, where $x_i \in \mathbb{R}^{\kappa}$. Edges are fully connected and weighted, and the weight of edge (x_i, x_j) indicates the affinity between data points x_i and x_j, defined as $\mathbf{A}_{ij} = \exp\left(-\|x_i - x_j\|^2/2\sigma^2\right)$. Here $\|x_i - x_j\|$ denotes the Euclidean distance between data points x_i and x_j, and σ^2 denotes the variance of all data points in V. We note that the value of distance $\|x_i - x_j\|^2$ is exponential to the edge weight $\mathbf{A}_{ij}$. Thus, small differences among pairwise distances can lead to significantly-skewed edge weights distribution. On the other hand, computing PPR values on affinity graphs is a commonly adopted technique in various tasks, such as label propagation [68], spectral clustering [64], image segmentation [62] and relationship profiling [60]. Therefore, to apply *LocalPush* for PPR computation on such heavily unbalanced weighted graphs can invoke expensive but unnecessary time cost.

Our Contributions. To remedy the above issue of *LocalPush* on weighted graphs, we make the following contributions:

- Edge-based Push Method. We propose *EdgePush*, a novel edge-based push method for SSPPR queries. Our *EdgePush* further decomposes the aforementioned atomic *push* operation of *LocalPush* into *separate edge-based push* operations. As a result, *EdgePush* can flexibly select edges to push probability mass based on the edge weights.
- Theoretical Analysis. *EdgePush* admits a fine-grained *individual* termination threshold $\theta(u, v)$ for each edge. With careful choices of $\theta(u, v)$, *EdgePush* achieves superior query efficiency in terms of the trade-offs between the approximation error and the expected overall running time. In this paper, we analyze the time complexity of *EdgePush* and present the suggested choice of $\theta(u, v)$ with two specific error measurements: the ℓ_1-error and the normalized additive error. In particular, when the edge weights are unbalanced (as shown in Figure 1), with the optimal setting of $\theta(u, v)$, *EdgePush* can approximate PPR values in time $o(m)$, *sub-linear* to the number of edges, with specified ℓ_1-error. In other words, in this case, we can solve the approximate PPR with ℓ_1-error even without touching every edge in the graph.
- Superiority Illustration. We demonstrate that *EdgePush* achieves a superior expected time complexity over *LocalPush* on arbitrary graphs as shown in Table 1. For the ease of illustration, here we present superior results for a relatively restricted case, where all the nodes in the graph are (a, b)-*unbalanced* (which is defined next). However, it should be noted that as proved in Section 5, the conditions for *EdgePush* strictly outperforming *LocalPush* are actually more general and less restrictive. Specifically, the notion of (a, b)-unbalanced node is used to quantify the unbalancedness of the weighted graph. A node is said to be (a, b)-unbalanced if a fraction of its adjacency edges take b fraction of its edge weights, where $0 \le a \le b \le 1$. Based on the (a, b)-unbalanced definition, we summarize three theoretical implications in the following. Here we assume the source node is chosen according to the node degree distribution.
  - The overall running time bound of *EdgePush* is no worse than that of *LocalPush* even on unweighted graphs, regardless of whether the ℓ_1-error or the normalized additive error.
  - When the edge weights are unbalanced, *EdgePush* achieves superior query efficiency over *LocalPush*. And the superiority of *EdgePush* over *LocalPush* can be quantified by the unbalancedness of the weighted graphs.
  - When the graph G is a complete graph with n nodes, $a = 1/n$ and $b = 1 - 1/n$, *EdgePush* outperforms *LocalPush* by a $O(n)$ factor, both for the ℓ_1 and normalized additive error.
  - When $a = o(1)$ and $b = 1 - o(1)$, *EdgePush* compute SSPPR queries in time *sub-linear* to the number of edges in the graph with any specified ℓ_1-error.

- Extensive Experiments. We conduct comprehensive experiments to show the effectiveness of our *EdgePush* on both motif-based and real-world weighted graphs. The experimental results demonstrate that when achieving the same approximation error (e.g. normalized additive error or ℓ_1-error), *EdgePush* outperforms *LocalPush* on large real-world graphs by orders of magnitude in terms of query efficiency. Notably, even on the graphs with less unbalanced edge weights, *EdgePush* still shows superior performances over the state-of-the-art baselines.

2 PRELIMINARIES

Notations. Consider an *undirected* and *weighted* graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges. We define $\bar{E}$ as the set of bi-directional edges of G, that is, for every edge $(u, v) \in E$, there are two directed edges $\langle u, v\rangle$ and $\langle v, u\rangle$ in $\bar{E}$, and these two edges are treated differently. We use $\mathbf{A}$ to denote the *adjacency matrix* of graph G, and $\mathbf{A}_{uv}$ to denote the *weight* of edge $\langle u, v\rangle \in \bar{E}$. Furthermore, we assume that each $\mathbf{A}_{uv}$ is a non-negative real number. For $\forall \langle u, v\rangle \notin \bar{E}$, we have $\mathbf{A}_{uv} = 0$. As a result, $\|\mathbf{A}\|_1 = \sum_{\langle u,v\rangle \in \bar{E}} \mathbf{A}_{uv}$ denotes the total weights of all edges. For every edge $\langle u, v\rangle \in \bar{E}$, we say v is a *neighbor* of u. For each node $u \in V$, we denote the set of all the *neighbors* of u by $N(u)$, and $n(u) = |N(u)|$, the neighborhood size of u. Moreover, $d(u) = \sum_{v \in N(u)} \mathbf{A}_{uv}$ denotes the (*weighted*) *degree* of node u, and $\mathbf{D}$ denotes the diagonal degree matrix with $\mathbf{D}_{uu} = d(u)$. Finally, the *transition matrix* is denoted by $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$.

In this paper, we use $\boldsymbol{\pi}_s \in \mathbb{R}^n$ to denote the SSPPR vector w.r.t node s as the source node. The u-th coordinate $\boldsymbol{\pi}_s(u)$ records the PPR value of node $u \in V$ w.r.t s. Unless specified otherwise, we denote node s as the source node by default and omit the subscript in $\boldsymbol{\pi}_s$ and $\boldsymbol{\pi}_s(u)$ for short (i.e. $\boldsymbol{\pi}$ and $\boldsymbol{\pi}(u)$). In Section 3, we use $\boldsymbol{r} \in \mathbb{R}^n$ and $\hat{\boldsymbol{\pi}} \in \mathbb{R}^n$ to denote the residue and reserve vectors in *LocalPush*, respectively. In Section 4, we define three variables: node income vector $\boldsymbol{q} \in \mathbb{R}^n$, edge expense matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$ and edge residue matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$ for the *EdgePush* algorithm. Additionally, in this paper, we analyze all of the theoretical complexities under the word RAM model [23] (a brief introduction of the Word RAM Model is deferred to the technical report [1]). Following the aforementioned convention of the model, we assume that every the numerical value, such as the edge weight $\mathbf{A}_{uv}$ or the constant teleport probability α in PPR computation, can fit into $O(1)$ words of $O(\log n)$ bits.

Single-Source Personalized PageRank (SSPPR). PageRank [46] is first proposed by Google to rank the overall importance of nodes in the graph. Personalized PageRank (PPR) is a variant of PageRank, which evaluates each node's *relative* importance w.r.t a given source node. The single-source PPR (SSPPR) query is a type of PPR computations, which aims to return all the PPR values (w.r.t the source node) in the graph. More precisely, given node s as the source node, the SSPPR query aims to derive an SSPPR vector $\boldsymbol{\pi} \in \mathbb{R}^n$, where the u-th coordinate $\boldsymbol{\pi}(u)$ represents the PPR value of node u.

In the seminal paper of PPR [46], the SSPPR vector $\boldsymbol{\pi}$ w.r.t source node s is defined as the solution to the recursive equation:

$$\boldsymbol{\pi} = (1-\alpha)\mathbf{P}\boldsymbol{\pi} + \alpha \boldsymbol{e}_s, \tag{1}$$

where $\alpha \in (0, 1)$ is a constant teleport probability, $\mathbf{P}$ is the transition matrix that $\mathbf{P} = \mathbf{A}\mathbf{D}^{-1}$, and $\boldsymbol{e}_s$ is an one-hot vector that $\boldsymbol{e}_s(s) = 1$ and $\boldsymbol{e}_s(u) = 0$ if $u \neq s$. By applying a power series expansion [11], the SSPPR vector $\boldsymbol{\pi}$ can be derived as:

$$\boldsymbol{\pi} = \alpha \cdot (\mathbf{I} - (1-\alpha)\mathbf{P})^{-1} \cdot \boldsymbol{e}_s = \sum_{i=0}^{\infty} \alpha(1-\alpha)^i \mathbf{P} \cdot \boldsymbol{e}_s. \tag{2}$$

Note that this expansion provides an alternative interpretation of PPR values: the SSPPR vector $\boldsymbol{\pi}$ can be regarded as a probability distribution, where $\boldsymbol{\pi}(u)$ denotes the probability that an α-random walk from the given source node s stops at u [38, 46]. Each step of an α-random walk either stops at the current node with probability α, or stays *alive* to move forward with $(1-\alpha)$ probability. Specifically, if an α-random walk is currently alive at node u, then in the next step, the walk will move to one of u's neighbors $v \in N(u)$ with the probability proportional to the edge weight $\mathbf{A}_{uv}$, i.e., with probability $\frac{(1-\alpha)\cdot \mathbf{A}_{uv}}{d(u)}$.

Problem Definition. As shown in Equation (2), the exact computation of SSPPR vector involves the inverse of the $n \times n$ matrix: $(\mathbf{I} - (1-\alpha)\mathbf{P})$, where n is the number of graph nodes. This is infeasible on large graphs with millions of nodes. Thus, in this paper, we aim to approximate the SSPPR vector $\boldsymbol{\pi}$ on large graphs with specified approximation error. Specifically, we consider the following two problems:

Definition 1 (Approximate SSPPR with normalized additive error). *Given an undirected and weighted graph $G = (V, E)$, a source node $s \in V$ and a normalized additive error tolerance $r_{\max} \in (0, 1)$, the goal of an approximate SSPPR query (w.r.t the source node s) with normalized additive error is to return an estimated SSPPR vector $\hat{\boldsymbol{\pi}}$ such that for each $u \in V$, $\left|\frac{\boldsymbol{\pi}(u)}{d(u)} - \frac{\hat{\boldsymbol{\pi}}(u)}{d(u)}\right| \leq r_{\max}$.*

The normalized additive error is a commonly used evaluation metric in local clustering tasks. More precisely, the majority of existing local clustering algorithms [10, 18, 19, 22, 31, 48, 63, 65] operate in two stages: first, they treat the given seed node as the source node and calculate the approximate SSPPR vector $\hat{\boldsymbol{\pi}}$ (or other scores to rank nodes' relative importance w.r.t the source node). Then the they feed the vector $\hat{\boldsymbol{\pi}}$ in a *sweep* process to identify local cluster around the seed node. The detailed steps in the sweep process are given below:

- (i) Put all the nodes with non-zero $\frac{\hat{\boldsymbol{\pi}}(u)}{d(u)}$ into a set S.
- (ii) Sort each node $u \in S$ in the descending order by $\frac{\hat{\boldsymbol{\pi}}(u)}{d(u)}$, such that $S = \{v_1, v_2, ..., v_j\}$ and $\frac{\hat{\boldsymbol{\pi}}(v_1)}{d(v_1)} \geq \frac{\hat{\boldsymbol{\pi}}(v_2)}{d(v_2)} \geq ... \geq \frac{\hat{\boldsymbol{\pi}}(v_j)}{d(v_j)}$.
- (iii) Scan the set S from v_1 to v_j and find the subset with minimum *conductance* among all the partial sets $S_i = \{v_1, v_2, ..., v_i\}$ for $i = 1, 2, ..., j$.

In the third step, we calculate the *conductance* of all partial sets. Conductance is a popular measure to evaluate the cluster quality. More precisely, given a cluster set $S_i \subseteq V$, the conductance of set S_i is defined as $\Phi(S_i) = \frac{cut(S_i)}{\min\{vol(S_i), vol(V\setminus S_i)\}}$, where $vol(S_i) = \sum_{u\in S_i} d(u)$ denotes the volume of set S_i, and $cut(S_i)$ denotes the sum of edge weights for those edges crossing S_i and $V \setminus S_i$. Thus, the conductance values are the smaller, the better.

Reviewing the sweep process, we note that the quality of the identified clusters heavily depends on the approximation accuracy of the normalized PPR values (i.e. $\frac{\boldsymbol{\pi}(u)}{d(u)}$ for each u). Therefore, in

Table 1: The comparison between the complexities of *LocalPush* and *EdgePush*. The "Improvements" column quantifies the superiority of *EdgePush* over *LocalPush* in terms of the expected time complexities when the source node is chosen according to the degree distribution. φ and φ_v denote specific angles on weighted graphs, which are formally illustrated in Section 5.

	LocalPush	*EdgePush*	Improvements
ℓ_1-error ε	$O\left(\frac{m}{\alpha\varepsilon}\right)$	$O\left(\frac{(1-\alpha)}{\alpha\varepsilon\|\mathbf{A}\|_1}\cdot\left(\sum_{\langle u,v\rangle\in E}\sqrt{\mathbf{A}_{uv}}\right)^2\right)=O\left(((1-\alpha)\cos^2\varphi)\cdot\frac{m}{\alpha\varepsilon}\right)$	$(1-\alpha)\cos^2\varphi$
normalized additive error $r_{\max}$	$O\left(\frac{m}{\alpha r_{\max}\cdot\|\mathbf{A}\|_1}\right)$	$O\left(\frac{(1-\alpha)}{\alpha r_{\max}\|\mathbf{A}\|_1}\cdot\sum_{v\in V}\frac{\left(\sum_{x\in N(v)}\sqrt{\mathbf{A}_{xv}}\right)^2}{d(v)}\right)=O\left(\left(\frac{1-\alpha}{m}\cdot\sum_{v\in V}n(v)\cos^2\varphi_v\right)\cdot\frac{m}{\alpha r_{\max}\|\mathbf{A}\|_1}\right)$	$\frac{(1-\alpha)}{2m}\cdot\sum_{v\in V}n(v)\cos^2\varphi_v$

Algorithm 1: The *LocalPush* Algorithm [65]

Input: undirected and weighted Graph $G=(V,E)$ with adjacency matrix $\mathbf{A}$, source node s, constant teleport probability $\alpha\in(0,1)$, termination threshold θ
Output: an estimation $\hat{\boldsymbol{\pi}}$ of SSPPR vector $\boldsymbol{\pi}$ w.r.t s

$\hat{\boldsymbol{\pi}}\leftarrow\mathbf{0}$, $\boldsymbol{r}\leftarrow\boldsymbol{e}_s$;
while *there exists a node* u *with* $\boldsymbol{r}(u)\geq d(u)\cdot\theta$ **do**
  $\hat{\boldsymbol{\pi}}(u)\leftarrow\hat{\boldsymbol{\pi}}(u)+\alpha\cdot\boldsymbol{r}(u)$;
  for *every neighbors* $v\in N(u)$ **do**
    $\boldsymbol{r}(v)\leftarrow\boldsymbol{r}(v)+(1-\alpha)\boldsymbol{r}(u)\cdot\frac{\mathbf{A}_{uv}}{d(u)}$;
  $\boldsymbol{r}(u)\leftarrow 0$;
return $\hat{\boldsymbol{\pi}}$ as the estimator for $\boldsymbol{\pi}$;

this paper, we introduce normalized additive error as one of the evaluation criteria. Additionally, we employ a classic evaluation metric, ℓ_1-error, to assess the approximation quality of each algorithm:

Definition 2 (Approximate SSPPR with ℓ_1-error). *Given an undirected weighted graph $G=(V,E)$, a source node $s\in V$, a constant teleport probability α, and an ℓ_1-error tolerance $\varepsilon\in(0,1)$, the goal of an approximate PPR query with respect to s is to return an estimated PPR vector $\hat{\boldsymbol{\pi}}$ such that $\|\hat{\boldsymbol{\pi}}-\boldsymbol{\pi}\|_1=\sum_{u\in V}|\hat{\boldsymbol{\pi}}(u)-\boldsymbol{\pi}(u)|\leq\varepsilon$.*

2.1 The *LocalPush* Algorithm

Among existing algorithms for SSPPR queries, the *LocalPush* algorithm [10, 65] is a fundamental method which serves as a cornerstone in various subsequent algorithms [22, 28, 54–56]. The basic idea of *LocalPush* is to "*simulate*" α-random walks in a deterministic way by *pushing* the probability mass from a node to its neighbors. More specifically, given an undirected and weighted graph $G=(V,E)$, a source node s, a constant teleport probability $\alpha\in(0,1)$ and a global termination threshold θ, *LocalPush* maintains two variables for each node $u\in V$ during the executing process:

- Residue $\boldsymbol{r}(u)$: the probability mass currently on u and will be distributed to other nodes. Alternatively, in an α-random walk process, $\boldsymbol{r}(u)$ records the probability mass of the α-random walk from s alive at u at the current state. Note that if a walk has not yet stopped, we say the walk is *alive* at the current node;
- Reserve $\hat{\boldsymbol{\pi}}(u)$: the probability mass that stays at node u. $\hat{\boldsymbol{\pi}}(u)$ is an underestimate of the real PPR value $\boldsymbol{\pi}(u)$.

The *push* Operation. *push* is a critical primitive operation that is repeated throughout the *LocalPush* process. A *push* operation consists of three steps (see the left side of Figure 2 for illustration):

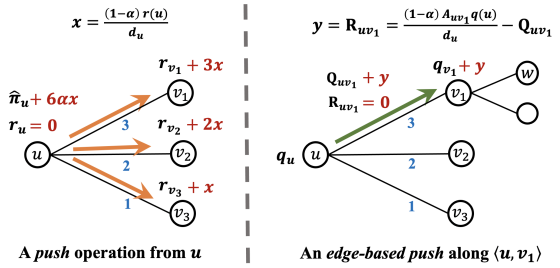

Figure 2: A sketch for comparing *push* and *edge-based push*. The number on each edge is the edge weight.

- convert α portion of $\boldsymbol{r}(u)$ to $\hat{\boldsymbol{\pi}}(u)$, i.e., $\hat{\boldsymbol{\pi}}(u)\leftarrow\hat{\boldsymbol{\pi}}(u)+\alpha\cdot\boldsymbol{r}(u)$, simulating the fact that α portion of the random walk alive at u has stopped here at u;
- distribute the rest $(1-\alpha)$ portion of $\boldsymbol{r}(u)$ to each neighbor v proportional to the corresponding edge weight by increasing the residue of v, i.e., $\boldsymbol{r}(v)\leftarrow\boldsymbol{r}(v)+(1-\alpha)\boldsymbol{r}(u)\cdot\frac{\mathbf{A}_{uv}}{d(u)}$; this essentially simulates that $(1-\alpha)$ portion of the random walk will move one step forward to each u's neighbor with the probability proportional to the edge weights;
- reset the residue $\boldsymbol{r}(u)$ to 0, indicating that, after the above two steps, there is no α-random walk alive at u at the moment.

Invariant. The analysis of localpush algorithm is built upon an invariant between the residue and the reserve, which is formalized in the following lemma:

Lemma 1 (Invariant by LocalPush). *For each node t in the graph, the reserve $\boldsymbol{\pi}_t$ and residues satisfy the following invariant after each push operation:*

$$\boldsymbol{\pi}(t)=\hat{\boldsymbol{\pi}}(t)+\sum_{u\in V}\boldsymbol{r}(u)\cdot\boldsymbol{\pi}_u(t),\tag{3}$$

where $\boldsymbol{\pi}(t)$ denotes the PPR value of node t w.r.t the source node s (by default) and $\boldsymbol{\pi}_u(t)$ denotes the PPR value of t w.r.t node u.

We defer the proof of Lemma 1 to the technical report [1] due to the space limit. Intuitively, $\hat{\boldsymbol{\pi}}(t)$ is the probability mass that stays at t. Aside from $\hat{\boldsymbol{\pi}}(t)$, $\boldsymbol{\pi}(t)$ also includes probability mass that is currently on other nodes and will be delivered to t. To calculate this part, recall that $\boldsymbol{r}(u)$ refers to the probability mass that is currently at u but will be distributed to other nodes. Given that $\boldsymbol{\pi}_u(t)$ is the probability of a random walk starting at u and ending at t, $\boldsymbol{r}(u)\cdot\boldsymbol{\pi}_u(t)$ is the probability mass now residing at u and to be distributed to t. Summing over all nodes $u\in V$ and the lemma follows. For a illustration, see the left part of Figure 3.

Algorithm Descriptions of *LocalPush*. Initially, the node residue vector $\boldsymbol{r}$ is initialized as $\boldsymbol{r}=[\boldsymbol{r}(u)]_{u\in V}=\boldsymbol{e}_s$, indicating that, at the

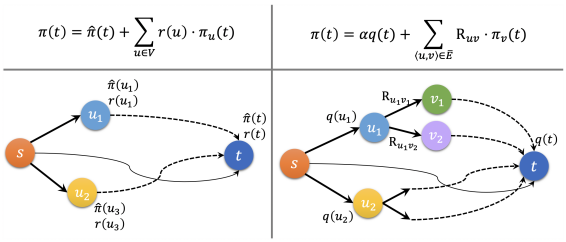

Figure 3: A sketch for comparing the invariants of *LocalPush* and *EdgePush*.

beginning, the walk is only alive at the source node s (with probability 1). Meanwhile, $\hat{\boldsymbol{\pi}} = [\hat{\boldsymbol{\pi}}(u)]_{u \in V} = \mathbf{0}$. During the *LocalPush* process, *LocalPush* repeatedly performs the *push* operations until there is no node u with residue $\boldsymbol{r}(u) \geq d(u) \cdot \theta$. After the termination, *LocalPush* returns the reserve vector $\hat{\boldsymbol{\pi}}$ as the approximation of SSPPR vector $\boldsymbol{\pi}$. The pseudo code of the *LocalPush* algorithm is illustrated in Algorithm 1.

Error Analysis and Time Complexity of *LocalPush*. On unweighted graphs, it is known that *LocalPush* costs $O\left(\frac{m}{\alpha\varepsilon}\right)$ and $O\left(\frac{1}{\alpha r_{\max}}\right)$ to answer the approximate SSPPR queries with ℓ_1-error ε and normalized additive error $r_{\max}$, respectively [10, 54, 56]. With a slight modification, we can derive the time complexities of *LocalPush* on weighted graphs:

Fact 1. *By setting the termination threshold θ in Algorithm 1 as $\theta = \varepsilon / \|\mathbf{A}\|_1$, LocalPush answers the SSPPR query with an ℓ_1-error ε. When the source node is randomly chosen according to the degree distribution, the expected cost of LocalPush is bounded by $O\left(\frac{m}{\alpha\varepsilon}\right)$.*

Fact 2. *By setting the termination threshold θ in Algorithm 1 as $\theta = r_{\max}$, LocalPush returns an approximation of SSPPR vector with normalized additive error $r_{\max}$. When the source node is randomly chosen according to the degree distribution, the expected cost of LocalPush is bounded by $O\left(\frac{m}{\alpha r_{\max} \cdot \|\mathbf{A}\|_1}\right)$.*

Considering Fact 2, if the number of edges m equals the total weights $\|\mathbf{A}\|_1$, which always holds for unweigthed graphs, the overall running time bound can be simplified to $O(\frac{1}{\alpha r_{\max}})$. For the sake of readability, we defer the proofs of Fact 1 and Fact 2 to our technical report [1]. Note that in these two facts, we assume the source node is randomly chosen according to the degree distribution for the sake of simplicity. This is also a common assumption in the context of local clustering application (i.e. the seed node is sampled according to the node degree).

3 RELATED WORK

Power Method. Power Method [46] is an iterative method to compute the SSPPR vector $\boldsymbol{\pi}$. Recall that in Equation (2), we present a power series expansion to compute $\boldsymbol{\pi}$, which alternatively suggests an iterative algorithm:

$$\boldsymbol{\pi}^{(\ell+1)} = (1-\alpha)\mathbf{P}\boldsymbol{\pi}^{(\ell)} + \alpha \boldsymbol{e}_s, \tag{4}$$

where $\boldsymbol{\pi}^{(\ell)}$ denotes the estimated SSPPR vector after the ℓ-th iteration. Power Method employs the recursive formula L times and regards $\boldsymbol{\pi}^{(L)}$ as the approximation of $\boldsymbol{\pi}$, leading to an $O(mL)$ time complexity. By setting $L = \log \frac{1}{\varepsilon}$ and $L = \log \frac{1}{r_{\max}}$, Power Method can achieve an ℓ_1 error of ε and a normalized additive error of $r_{\max}$, respectively. Note that the settings of L show logarithmic dependence on the error parameters (i.e. ε or $r_{\max}$), which enables Power Method to answer high-precision SSPPR queries. However, in each iteration, Power Method needs to touch the every edge on the graph, resulting in a $\Theta(m)$ time cost per iteration. This can severely limit the scalability of Power Method on large graphs.

Monte-Carlo Sampling. Monte-Carlo sampling [21, 29] estimates SSPPR vector $\boldsymbol{\pi}$ by simulating the random walk process. More precisely, recall that the PPR value $\boldsymbol{\pi}(u)$ of node u equals to the probability that an α-random walk from the given source node s stops at node u. Based on this interpretation, Monte-Carlo sampling first generates multiple α-random walks from the source node s, then uses the fraction that the number of walks terminates at u as an approximation of $\boldsymbol{\pi}(u)$. However, due to the uncertainty of random walks, Monte-Carlo method is inefficient to achieve small approximation error on large graphs [54].

FORA. Existing methods [36, 41, 53, 54] adopt various approaches to improve the efficiency of Monte-Carlo Sampling. As a representative algorithm, FORA [54] combines *LocalPush* with Monte-Carlo Sampling to approximate the SSPPR vector. By theoretical analysis, FORA provides the optimal settings of the error parameters in *LocalPush* and Monte-Carlo sampling to balance the two phases. Additionally, FORA introduces an index scheme, as well as a module for top-k selection with high pruning power, which, however, is out of the scope of this paper.

PowForPush and SpeedPPR. Recently, Wu et al. [56] proposes PowForPush to accelerate the efficiency of high-precision SSPPR queries. PowForPush is based on the Power Method and adopts two optimization techniques, sequential scanning and dynamic ℓ_1-error threshold, for better performance from an engineering point of view. However, the time complexity of PowForPush is still the same as that of Power Method, which still has a linear dependency on the number of edges m.

As a variant of PowForPush, SpeedPPR [56] combines PowForPush with Monte-Carlo Sampling for efficient approximation of SSPPR queries. As shown in [56], SpeedPPR achieves superior time complexity over FORA under relative error constraints. The success of SpeedPPR demonstrates that *LocalPush* serves as a cornerstone approach for SSPPR queries, and hence, any improvement on *LocalPush* can be applied to the subsequent methods for advanced developments.

Other Related Work. Except for SSPPR queries, there are other lines of researches concerning single-target PPR queries [8, 9, 41, 52], single-pair PPR queries [24, 39, 40, 42, 53], distributed PPR queries [28, 37, 43, 47, 70], or SSPPR computation on dynamic graphs [45, 66, 67]. However, these works are orthogonal to the focus of this paper.

4 EDGE-BASED LOCAL PUSH

An Overview. At a high level, *EdgePush* decomposes the atomic *push* operation into edge level granularity, which enables to flexibly distribute probabilities according to the edge weights. To demonstrate the superiority of *edge-based push*, consider the toy graph shown in Figure 1. We note that the atomic *push* operation hinders *LocalPush* to flexibly arrange the push order in the finer-grained edge granularity, leading to the $O(n)$ time cost even after the first

push step (i.e. from node u to $v_1, v_2, \ldots, v_n$). However, due to the unbalanced edge weights, the probability mass distributed from u to $v_2, \ldots, v_n$ are "insignificant" compared to the probability distributed to v_1. Consequently, in this toy example, the optimal push strategy is to directly distribute the probability mass at node u along edge $\langle u, v_1 \rangle$, then along edge $\langle v_1, w \rangle$, and ignore the other edges, which is allowed in our *EdgePush* framework.

4.1 A primitive operation: *edge-based push*

Notations. Before presenting the algorithm structure of *EdgePush*, we first define three variables which are conceptually maintained in the *EdgePush* process.

- A *node income* $\boldsymbol{q}(v)$ for each node $v \in V$: it records the total probability mass received by v so far. Therefore, $\alpha \boldsymbol{q}(v)$ is indeed the reserve $\hat{\boldsymbol{\pi}}(v)$ in the context of *LocalPush*.
- An *edge expense* $\mathbf{Q}_{uv}$ for each edge $\langle u, v \rangle \in \bar{E}$: this variable records the total probability mass that has been transferred from u to v via the edge $\langle u, v \rangle$. By definition, the node income of node v is the sum of all the expenses of edges $\forall \langle u, v \rangle \in \bar{E}$, i.e., $\boldsymbol{q}(v) = \sum_{u \in N(v)} \mathbf{Q}_{uv}$.
- An *edge residue* $\mathbf{R}_{uv}$ for each edge $\langle u, v \rangle \in \bar{E}$: it records the probability mass that is to be transferred from u to v at the moment. Thus, by definition, we have:

$$\mathbf{R}_{uv} = (1-\alpha)\boldsymbol{q}(u) \cdot \frac{\mathbf{A}_{uv}}{d(u)} - \mathbf{Q}_{uv}, \tag{5}$$

  where $(1-\alpha)\boldsymbol{q}(u) \cdot \frac{\mathbf{A}_{uv}}{d(u)}$ indicates the total probability mass that should be transferred from u to v so far.

Correctness. Similar to the invariant maintained by *LocalPush*, we can prove an analogous invariant for *EdgePush*:

Lemma 2 (Invariant by EdgePush). *For each node t in the graph, the node income $\boldsymbol{q}(t)$ and edge residues satisfy the following invariant during the EdgePush process:*

$$\boldsymbol{\pi}(t) = \alpha \boldsymbol{q}(t) + \sum_{\langle u,v \rangle \in \bar{E}} \mathbf{R}_{uv} \cdot \boldsymbol{\pi}_v(t), \tag{6}$$

where $\boldsymbol{\pi}_v(t)$ denotes the PPR value of node t w.r.t node v as the source node. When the source node is s, we use $\boldsymbol{\pi}(t)$ by default.

A sketch for the invariant is shown in the right side of Figure 3. To see the correctness of this invariant, recall that in *LocalPush*, the probability mass to be distributed from each node u is recorded by the *node residue* $\boldsymbol{r}(u)$. In contrast, in our *EdgePush*, we further decompose the probability mass maintained by *node residue* into edge level granularity and use *edge residue* to record it. Hence, the PPR value of node t is divided to two types of probabilities. $\alpha \boldsymbol{q}(t)$ maintains the probability mass that has been received by node t. Except that, $\sum_{\langle u,v \rangle \in \bar{E}} \mathbf{R}_{uv} \cdot \boldsymbol{\pi}_v(t)$ records the probability mass to be distributed to t. The formal proof of this invariant is deferred to the Technical Report [1] due to the space limit.

The *Edge-based Push* Operation. In an *edge-based push* along edge $\langle u, v_1 \rangle$ as shown in the right side of Figure 2, *EdgePush* transfers the probability mass at $\mathbf{R}_{uv_1}$ to node v_1 (i.e. to the edge residues of all edges $\langle v_1, w \rangle$), and reset $\mathbf{R}_{uv_1}$ to 0.

Notably, in the *EdgePush* process, we don't explicitly modify the edge residue. Instead, we update the node income and edge expense with the edge residue maintained implicitly according to Equation (5). For example, in the *edge-based push* along $\langle u, v_1 \rangle$ as shown in Figure 2, we first calculate $\mathbf{R}_{uv_1}$ at current stage by $\mathbf{R}_{uv_1} = \frac{(1-\alpha)\mathbf{A}_{uv}\boldsymbol{q}(u)}{d(u)} - \mathbf{Q}_{uv_1}$. Then we increase $\boldsymbol{q}(v_1) \leftarrow \boldsymbol{q}(v_1) + \mathbf{R}_{uv_1}$ and $\mathbf{Q}_{uv_1} \leftarrow \mathbf{Q}_{uv_1} + \mathbf{R}_{uv_1}$. By this means, $\mathbf{R}_{uv_1}$ is implicitly set to 0 and for $\forall \langle v_1, w \rangle$, $\mathbf{R}_{v_1 w}$ is increased simultaneously because of the increment of $q(v_1)$.

Algorithm 2: The *EdgePush* Algorithm

Input: Graph $G=(V,E)$, source node $s \in V$, teleport probability α, termination threshold $\theta(u,v)$ for $\forall \langle u, v \rangle \in \bar{E}$
Output: $\hat{\boldsymbol{\pi}}$ as the estimation of SSPPR vector $\boldsymbol{\pi}$

```
q ← e_s, Q ← 0_{n×n};
C = { ⟨u,v⟩ ∈ Ē | (1−α) q(u) · A_uv/d(u) − Q_uv ≥ θ(u,v) };
while C is not empty do
    pick an edge ⟨u,v⟩ ∈ C;
    y ← (1−α) q(u) · A_uv/d(u) − Q_uv;
    Q_uv ← Q_uv + y;
    q(v) ← q(v) + y;
    Update the set C;
return π̂ = αq as the estimation of π;
```

4.2 The *EdgePush* Algorithm

In Algorithm 2, we present the pseudo code of the *EdgePush* algorithm. More precisely, given a weighted and undirected graph $G = (V, E)$, a source node s, a constant teleport probability $\alpha \in (0, 1)$ and the termination threshold $\theta(u, v)$ for $\forall \langle u, v \rangle \in \bar{E}$, *EdgePush* initializes the node income vector $\boldsymbol{q}$ as $\boldsymbol{e}_s$ and the edge expense matrix $\mathbf{Q}$ as $\mathbf{0}_{n \times n}$. During the *EdgePush* process, the *EdgePush* algorithm *conceptually* maintains a set C for the candidate edges, which is defined as $C = \{\langle u, v \rangle \in \bar{E} \mid \mathbf{R}_{uv} \geq \theta(u, v)\}$. *EdgePush* repeatedly picks edges from the candidate set C to perform *edge-based push* operations until $C = \emptyset$. Specifically, *EdgePush* repeats the following process until the termination.

- Pick an arbitrary edge $\langle u, v \rangle \in C$ and let $y \leftarrow \mathbf{R}_{uv}$;
- Perform an *edge-based push* operation on $\langle u, v \rangle$ by "pushing" the edge residue $\mathbf{R}_{uv}$ along the edge $\langle u, v \rangle$ from u to v; as a result, both the node income of v and the edge expense of $\langle u, v \rangle$ are increased by an amount of y, i.e., $\boldsymbol{q}(v) \leftarrow \boldsymbol{q}(v) + y$ and $\mathbf{Q}_{uv} \leftarrow \mathbf{Q}_{uv} + y$, where $y = \mathbf{R}_{uv} = \frac{(1-\alpha)\mathbf{A}_{uv}\boldsymbol{q}(u)}{d(u)} - \mathbf{Q}_{uv}$.
- Conceptually maintain the set C according to the increases of $\boldsymbol{q}(v)$ and $\mathbf{Q}_{uv}$.

When the above process terminates, *EdgePush* returns $\hat{\boldsymbol{\pi}} = \alpha \boldsymbol{q}$ as the estimator of the SSPPR vector $\boldsymbol{\pi}$.

4.3 Maintaining Candidate Set C

It remains to show how to maintain the candidate set C efficiently in Algorithm 2. Clearly, the cost of maintaining C explicitly can be expensive. To see this, consider the moment right after an *edge-based push* operation on edge $\langle u, v \rangle$ is performed. First, due to the increment of $\boldsymbol{q}(v)$, according to Equation (5), the edge residues of all v's adjacency edges $\langle v, w \rangle \in \bar{E}$ are increased, and thus, can be possibly inserted to C. This cost can be as large as $\Omega(n(v))$. Second, due to the increment of $\mathbf{Q}_{uv}$, the edge residue $\mathbf{R}_{uv}$ is equivalently

reduced to 0. As a result, $\langle u, v\rangle$ will be removed from C. Therefore, maintaining C explicitly would lead to a $\Omega(n(v)+1)$ cost, which is hardly considered as more efficient than a *push* operation in *LocalPush*.

Fortunately, a key observation is that the *EdgePush* algorithm just needs to pick an *arbitrary* edge in C to perform an *edge-based push* operation. There is actually no need to maintain C explicitly. Based on this observation, we propose a two-level structure to efficiently find "eligible" edges in C to push. As we shall show shortly, this two-level structure dramatically brings down the aforementioned $\Omega(n(v)+1)$ cost to $O(1)$ amortized.

A Two-Level Structure. Recall that in the *EdgePush* algorithm, the candidate set C is defined as $C = \{\langle u, v\rangle \in \vec{E} \mid \mathbf{R}_{uv} \geq \theta(u, v)\}$. According to Equation (5) that $\mathbf{R}_{uv} = (1-\alpha)\boldsymbol{q}(u) \cdot \frac{\mathbf{A}_{uv}}{d(u)} - \mathbf{Q}_{uv}$, we can rewrite the definition of C as:

$$C = \left\{\langle u, v\rangle \in \vec{E} \mid \frac{(1-\alpha)\boldsymbol{q}(u)}{d(u)} \geq \frac{1}{\mathbf{A}_{uv}} \cdot (\mathbf{Q}_{uv} + \theta(u, v))\right\}. \quad (7)$$

We note that in the above inequality, the right side focuses on the probability mass distributed to node u's adjacency edges, at a local level. The left side of the inequality maintains the probability mass of each node u in the graph, that is, at a global level. Thus, we propose a two-level structure to update the set C at the local and global level separately:

- At the local level, for each node $u \in V$, we maintain a *priority queue* of all the neighbors of u, denoted by $\mathcal{Q}(u)$, where the priority of each neighbor v is defined as:

$$k_u(v) = \frac{1}{\mathbf{A}_{uv}} \cdot (\mathbf{Q}_{uv} + \theta(u, v)) \ . \quad (8)$$

- At the global level, for $\forall u \in V$, we define the key of node u as:

$$K_u = -\frac{(1-\alpha)\boldsymbol{q}(u)}{d(u)} + \mathcal{Q}(u).\text{top}, \quad (9)$$

  where $\mathcal{Q}(u)$.top is the smallest priority value in $\mathcal{Q}(u)$. We maintain a *linked list* $\mathcal{L}$ for storing all the nodes u such that $K_u \leq 0$.

According to the definition of the two-level structure, we have a crucial observation given below:

Observation 1. *Let v be the neighbor of u with the smallest priority in $\mathcal{Q}(u)$. Then the edge $\langle u, v\rangle \in C$ if and only if $K_u \leq 0$.*

Proof. Since v is the neighbor of u with the smallest priority in $\mathcal{Q}(u)$, we have

$$K_u = -\frac{(1-\alpha)\boldsymbol{q}(u)}{d(u)} + \mathcal{Q}(u).\text{top} = -\frac{(1-\alpha)\boldsymbol{q}(u)}{d(u)} + \frac{(\mathbf{Q}_{uv} + \theta(u, v))}{\mathbf{A}_{uv}}.$$

As a result, $K_u \leq 0$ if and only if $\frac{(1-\alpha)\boldsymbol{q}(u)}{d(u)} - \frac{1}{\mathbf{A}_{uv}} \cdot (\mathbf{Q}_{uv} + \theta(u, v)) \geq 0$, which is concurs with the definition of C given in Equation (7). Thus, the observation follows. □

Based on the two-level structure and Observation 1, we can pick edges from the candidate set C without updating the edge residues of all adjacency edges.

Conceptually Pick an Edge from C. To pick an edge from C, by Observation 1, it suffices to first pick an arbitrary node u from the linked list $\mathcal{L}$, and then, take the edge $\langle u, v\rangle$ with v having the smallest priority in $\mathcal{Q}(u)$. This can be implemented easily by taking a node from a linked list and by invoking the *find-min* operation of the priority queue.

Conceptually Maintain C. Let $\langle u, v\rangle$ be the edge picked. After the edge-based push operation on $\langle u, v\rangle$, our *EdgePush* performs the following steps: (i) invoke an *increase-key* operation for v in $\mathcal{Q}(u)$ (we discuss how to implement this with allowed priority queue operations shortly); (ii) check the key K_u: if $K_u > 0$, remove u from the linked list $\mathcal{L}$; and (iii) check the key K_v: if $K_v \leq 0$, add v to $\mathcal{L}$.

Correctness of the two-level structure. The correctness of the two-level structure for maintaining the candidate set C can be proved based on two facts. First, we observe that the linked list $\mathcal{L}$ always keeps all the nodes u with $K_u \leq 0$. Second, the edge expense $\mathbf{Q}_{uv}$ can be increased only and this happens only when a push is performed on the edge $\langle u, v\rangle$. According to Equation (8), if an edge of u is eligible for a push, it will eventually appear at the top of $\mathcal{Q}(u)$, and then be captured by Observation 1.

Cost per *edge-based push*. Recall that we assume the word RAM model where each edge weight value $\mathbf{A}_{uv}$ and each priority value $k_u(v)$ can be represented by $O(\log n)$ bits. In this model, we can sort all the edge weights $\mathbf{A}_{uv}$ and all the priorities $k_u(v)$ in $O(m)$ time with the standard Radix sort [32]. The key idea is to perform Counting Sort on every $\log n$ bits rather than on every bit, leading to the number of passes as $O(1)$. For the detailed proof on this complexity, please refer to the technical report [1] due to the space limit. Furthermore, we have the following fact:

Fact 3 (Theorem 1 in [50]). *If all the m priorities $k_u(v)$ can be sorted in $O(m)$ time, there exists a priority queue $\mathcal{Q}$ with capacity of m which supports each: (i)* find-min *operation in $O(1)$ worst-case time, (ii)* delete *operation (removing an element from $\mathcal{Q}$) in $O(1)$ amortized time, and (iii)* restricted insert *operation (inserting an element with priority $> \mathcal{Q}$.top) in $O(1)$ amortized time.*

In the above implementation, each edge-based push operation only involves: one *find-min* and one *increase-key* in the priority queues, and $O(1)$ standard operations in the linked list $\mathcal{L}$. As the *increase-key* operation can be implemented by a *restricted insert* followed by a *delete* operation in $\mathcal{Q}(u)$, the cost of *increase-key* is bounded by $O(1)$ amortized. Thus, we can derive the following theorem:

Theorem 1. *The cost of each edge-based push operation is bounded by $O(1)$ amortized.*

Pre-processing. As the edge weights can be sorted in $O(m)$ time and by Fact 3, we can pre-process the input graph G in $O(m)$ time, such that for each node $u \in V$: (i) all the out-going edges of u are sorted by their weights $\mathbf{A}_{uv}$, and (ii) the priority queue $\mathcal{Q}(u)$ is constructed. Furthermore, we may also store certain aggregated information in memory such as $\|\mathbf{A}\|_1$ and $\sum_{\langle u,v\rangle \in \vec{E}} \sqrt{\mathbf{A}_{uv}}$ (which we shall see shortly in the analyses).

5 THEORETICAL ANALYSIS

In this section, we analyze the theoretical error and time complexity of *EdgePush*. Additionally, we provide a novel notion, $\cos^2 \varphi$, to characterize the unbalancedness of weighted graphs and to assist in evaluating the theoretical advantage of *EdgePush* over *LocalPush*. Due to the space limit, we defer all proofs in this section to the technical report [1].

5.1 Analysis for the *EdgePush* Algorithm

Overall Time Complexity. To bound the overall time cost of *EdgePush*, recall that Theorem 1 states each edge push operation takes amortized constant time. Consequently, it suffices to bound the total number of edge push operations as the overall time complexity.

Lemma 3 (Time cost of EdgePush). *The overall running time of EdgePush is bounded by* $O\left(\sum_{\langle u,v\rangle\in\bar{E}} \frac{(1-\alpha)\boldsymbol{\pi}(u)\mathbf{A}_{uv}}{\alpha\cdot d(u)\cdot\theta(u,v)}\right)$. *In particular, when the source node is randomly chosen according to the degree distribution, the expected overall running time of EdgePush is bounded by* $O\left(\sum_{\langle u,v\rangle\in\bar{E}} \frac{(1-\alpha)\mathbf{A}_{uv}}{\alpha\cdot\|\mathbf{A}\|_1\cdot\theta(u,v)}\right)$.

Error Analysis. Recap the Invariant (6) shown in Section 4. By using $\alpha\boldsymbol{q}(t)$ as an approximate PPR value of $\boldsymbol{\pi}(t)$, we have two straightforward observations: (i) $\alpha\boldsymbol{q}(t)$ is an underestimate because all edge residuals are non-negative, and (ii) the additive error $\boldsymbol{\pi}(t)-\alpha\boldsymbol{q}(t)$ is bounded by $\sum_{\langle u,v\rangle\in\bar{E}}\mathbf{R}_{uv}\cdot\boldsymbol{\pi}_v(t)$, the edge residuals to be distributed to t. Summing over all possible target node t, we have the following lemma about the bound on the ℓ_1-error of *EdgePush*.

Lemma 4 (ℓ_1-error). *The EdgePush method shown in Algorithm 2 returns an approximate SSPPR vector within an* ℓ_1*-error* $\sum_{\langle u,v\rangle\in\bar{E}}\theta(u,v)$.

Moreover, in fact, the proof of Lemma 4 also derives the additive error bound for *EdgePush*. We have the following Lemma.

Lemma 5 (Normalized Additive Error). *For each node* $t\in V$ *in the graph, the EdgePush method answers the SSPPR queries within a normalized additive error* $\frac{1}{d(t)}\cdot\sum_{\langle u,v\rangle\in\bar{E}}\theta(u,v)\cdot\boldsymbol{\pi}_v(t)$ *for each node* $t\in V$.

5.2 Settings of the Termination Threshold

As shown in Algorithm 2, our *EdgePush* admits an individual termination threshold $\theta(u,v)$ for each edge $\langle u,v\rangle\in\bar{E}$. As we shall show in the following, by setting $\theta(u,v)$ carefully, the *EdgePush* achieves superior query efficiency over *LocalPush*.

Before illustrating the setting of $\theta(u,v)$, we first present an important inequality: Cauchy-Schwarz Inequality [49], which serves as the basis of the following analysis.

Fact 4 (Cauchy-Schwarz Inequality [49]). *Given two vectors* $\boldsymbol{\zeta}=\{\boldsymbol{\zeta}(1),\boldsymbol{\zeta}(2),...,\boldsymbol{\zeta}(m)\}\in\mathbb{R}^m$, $\boldsymbol{\chi}=\{\boldsymbol{\chi}(1),\boldsymbol{\chi}(2),...,\boldsymbol{\chi}(m)\}\in\mathbb{R}^m$, *the Cauchy-Schwarz Inequality states that:*

$$\left(\sum_{i=1}^{m}\boldsymbol{\zeta}(i)\cdot\boldsymbol{\chi}(i)\right)^2\le\left(\sum_{i=1}^{m}\boldsymbol{\zeta}^2(i)\right)\left(\sum_{i=1}^{m}\boldsymbol{\chi}^2(i)\right),\tag{10}$$

where the equality holds when $\frac{\boldsymbol{\zeta}(1)}{\boldsymbol{\chi}(1)}=\frac{\boldsymbol{\zeta}(2)}{\boldsymbol{\chi}(2)}=...=\frac{\boldsymbol{\zeta}(m)}{\boldsymbol{\chi}(m)}$.

The Choice of $\theta(u,v)$. We first present the optimal choice of $\theta(u,v)$ for *EdgePush* with ℓ_1-error ε. Let us recap two results. Firstly, Lemma 4 shows that the ℓ_1-error is bounded by $\sum_{\langle u,v\rangle\in\bar{E}}\theta(u,v)$. On the other hand, by Lemma 3, when the source node is chosen according to the degree distribution, the overall expected running time is bounded by $O\left(\sum_{\langle u,v\rangle\in\bar{E}}\frac{(1-\alpha)\mathbf{A}_{uv}}{\alpha\cdot\|\mathbf{A}\|_1\cdot\theta(u,v)}\right)$. Clearly, there is a trade-off between the error and the running time cost via the values of $\theta(u,v)$ for all $\langle u,v\rangle\in\bar{E}$. As a result, it suffices to aim at a setting of all $\theta(u,v)$'s such that: (i) the overall ℓ_1-error $\sum_{\langle u,v\rangle\in\bar{E}}\theta(u,v)=\varepsilon$, and (ii) the quantity $Cost\triangleq\sum_{\langle u,v\rangle\in\bar{E}}\frac{(1-\alpha)\mathbf{A}_{uv}}{\alpha\|\mathbf{A}\|_1\cdot\theta(u,v)}$ is minimized. Consequently, we prove the following theorem:

Theorem 2. *By setting* $\theta(u,v)=\frac{\varepsilon\cdot\sqrt{\mathbf{A}_{uv}}}{\sum_{\langle x,y\rangle\in\bar{E}}\sqrt{\mathbf{A}_{xy}}}$ *for each* $\langle u,v\rangle\in\bar{E}$, *the EdgePush algorithm returns an approximate SSPPR vector within* ℓ_1*-error at most* ε. *In particular, when the source node is randomly chosen according to the degree distribution, the expected overall running time is bounded by* $O\left(\frac{(1-\alpha)}{\alpha\varepsilon\|\mathbf{A}\|_1}\cdot\left(\sum_{\langle u,v\rangle\in\bar{E}}\sqrt{\mathbf{A}uv}\right)^2\right)$.

Likewise, we can derive the optimal choice of $\theta(u,v)$ for *EdgePush* with normalized additive error $r_{\max}$, illustrated in Theorem 3.

Theorem 3. *By setting* $\theta(u,v)=\frac{r_{\max}\cdot d(v)\sqrt{\mathbf{A}_{uv}}}{\sum_{x\in N(v)}\sqrt{\mathbf{A}_{xv}}}$ *for each* $\langle u,v\rangle\in\bar{E}$, *the EdgePush algorithm returns an approximate SSPPR vector within normalized additive error at most* r_{max}. *When the source node is randomly chosen according to the degree distribution, the expected overall running time is bounded by* $O\left(\frac{(1-\alpha)}{\alpha r_{\max}\|\mathbf{A}\|_1}\cdot\sum_{v\in V}\frac{\left(\sum_{x\in N(v)}\sqrt{\mathbf{A}_{xv}}\right)^2}{d(v)}\right)$.

5.3 Comparison to the *LocalPush* Algorithm

Next, we show the superiority of *EdgePush* over *LocalPush*. To facilitate our analysis, we first define the following four *characteristic vectors* of a given undirected weighted graph:

Definition 3 (characteristic vectors on weighted graphs). *Consider an undirected weighted graph* $G=(V,E)$ *with* n *nodes,* m *edges and* $\bar{E}$ *being the set of the bi-directional edges of every edge in* E; *clearly,* $|\bar{E}|=2|E|=2m$. *Denote the (weighted) adjacency matrix by* $\mathbf{A}$. *We define four characteristic vectors* $\boldsymbol{\zeta}$, $\boldsymbol{\chi}$, $\boldsymbol{\zeta}_v$ *and* $\boldsymbol{\chi}_v$ *of* G *as follows:*

- $\boldsymbol{\zeta}\in\mathbb{R}^{2m}$*: the vector whose the* i^{th} *entry* $\boldsymbol{\zeta}(i)=\sqrt{\mathbf{A}_{uv}}$ *corresponds to the* i^{th} *edge* $\langle u,v\rangle\in\bar{E}$;
- $\boldsymbol{\chi}\in\mathbb{R}^{2m}$*: an all-one vector in the* $2m$*-dimensional space;*
- $\boldsymbol{\zeta}_v\in\mathbb{R}^{n_v}$ *for each node* $v\in V$*: the vector whose the* j^{th} *entry* $\boldsymbol{\zeta}_v(j)=\sqrt{\mathbf{A}_{uv}}$ *corresponds to the* j^{th} *neighbor node* u *in* N_v;
- $\boldsymbol{\chi}_v\in\mathbb{R}^{n_v}$ *for each node* $v\in V$*: an all-one vector in the* n_v*-dimensional space.*

Then the improvement of *EdgePush* over *LocalPush* can be quantified by the the above characteristic vectors.

Lemma 6 (Superiority of EdgePush with ℓ_1-error). *For the approximate SSPPR queries with* ℓ_1*-error* ε, *we have*

$$\frac{(1-\alpha)}{\alpha\varepsilon\|\mathbf{A}\|_1}\cdot\left(\sum_{\langle u,v\rangle\in\bar{E}}\sqrt{\mathbf{A}_{uv}}\right)^2=\left((1-\alpha)\cos^2\varphi\right)\cdot\frac{2m}{\alpha\varepsilon},\tag{11}$$

where φ *is the angle between the characteristic vectors* $\boldsymbol{\zeta}$ *and* $\boldsymbol{\chi}$.

We note that the left hand side of Equation (11) is the overall expected running time of *EdgePush* and $\frac{2m}{\alpha\varepsilon}$ is that of *LocalPush*, both expressed by ignoring the Big-Oh notation. To see the correctness of Lemma 6, observe that $2m\cdot\|\mathbf{A}\|_1\cdot\cos^2\varphi=\left(\sum_{\langle u,v\rangle\in\bar{E}}1\right)\cdot\left(\sum_{\langle u,v\rangle\in\bar{E}}\mathbf{A}_{uv}\right)\cdot\cos^2\varphi=\|\boldsymbol{\chi}\|^2\cdot\|\boldsymbol{\zeta}\|^2\cdot\cos^2\varphi=\langle\boldsymbol{\chi},\boldsymbol{\zeta}\rangle^2=\left(\sum_{\langle u,v\rangle\in\bar{E}}\sqrt{\mathbf{A}_{uv}}\right)^2$. A more detailed proof can be found in the Technical Report [1].

Likewise, the superiority of *EdgePush* for the SSPPR queries with normalized additive error can be quantified as follows.

Table 2: Real-World Datasets.

Dataset	n	m	Edge weight		$\cos^2\varphi$
			mean	max	
YouTube (YT)	1,138,499	2,795,228	6.6	4,034	0.65
LiveJournal (LJ)	4,847,571	71,062,058	24	4,445	0.51
IndoChina (IC)	7,414,768	295,191,370	1,221	178,448	0.31
Orkut-Links (OL)	3,072,441	202,392,682	18	9,145	0.69
Tags (TA)	49,945	8,294,604	13	469,258	0.27
Threads (TH)	2,321,767	42,012,344	1.1	546	0.97
blockchair (BC)	595,753	1,773,544	5.2	17,165	0.5
Spotify (SP)	3,604,308	3,854,964,026	8.6	2,878,970	0.29

Lemma 7 (Superiority of *EdgePush* with normalized additive error). *For the approximate SSPPR queries with specified normalized additive error $r_{\max}$, the expected overall running time of EdgePush is at most a portion $\frac{(1-\alpha)}{2m} \cdot \left(\sum_{v\in V} n_v \cdot \cos^2\varphi_v\right)$ of the LocalPush's running time cost, where φ_v is the angle between vectors $\boldsymbol{\zeta}_v$ and $\boldsymbol{\chi}_v$. Specifically, we have*

$$\frac{(1-\alpha)}{\alpha r_{\max}\|\mathbf{A}\|_1}\cdot\sum_{v\in V}\frac{\left(\sum_{x\in N(v)}\sqrt{\mathbf{A}_{xv}}\right)^2}{d(v)} = \frac{(1-\alpha)\cdot\left(\sum_{v\in V} n(v)\cdot\cos^2\varphi_v\right)}{2m}\cdot\frac{2m}{\alpha r_{\max}\|\mathbf{A}\|_1}.$$

Superiority of *EdgePush* over *LocalPush*. Based on Lemma 6 and Lemma 7, we can derive several interesting observations:

- First, $\cos^2\varphi \le 1$ holds for all values of φ (also applies for $\forall\varphi_v$). This implies that the overall expected running time bound of *EdgePush* is never worse than that of *LocalPush*, regardless of the SSPPR queries with either ℓ_1-error or normalized additive error.
- Second, when $\cos^2\varphi = \Theta(1/n)$ (resp., $\cos^2\varphi_v = \Theta(1/n)$ for $\forall v$), *EdgePush* outperforms *LocalPush* in terms of efficiency by a $\Theta(n)$ factor for answering SSPPR queries with ℓ_1-error (resp., normalized additive error). This case could happen (but not necessarily) when all the nodes in a complete graph G are (a, b)-unbalanced with $a = 1/n$ and $b = 1 - 1/n$. As an example, one can consider the case that each node in G shares the same structure as node u shown in Figure 1.
- Third, when $\cos^2\varphi = o(1)$, e.g. $\cos^2\varphi = 1/\log m$, *EdgePush* can achieve a *sub-linear* expected time complexity $o(\frac{m}{\alpha\varepsilon})$ for solving the approximate SSPPR problem with specified ℓ_1-error ε. This is impressive because *EdgePush* can answer the SSPPR query even without "touching" all the edges of G. It can be verified that the aforementioned complete graph example satisfies this condition with $\cos^2\varphi = 1/n$.

Intuitively, both the notions of $\cos^2\varphi$ (resp., $\cos^2\varphi_v$) and the (a, b)-unbalanced (mentioned in Introduction) capture the *unbalance* of the undirected weighted graph G. However, we note that the former is actually more general than the latter. For more detailed discussions, please refer to our technical report [1].

6 EXPERIMENTS

In this section, we conduct experiments to show the effectiveness of *EdgePush* on large real-world datasets. Additionally, we evaluate the sensitivity of *EdgePush* to the unbalancedness of weighted graphs. Due to the space limit, we defer the sensitivity studies to the technical report [1]

Experiment Environment. We conduct all the experiments on a machine with an Intel(R) Xeon(R) Gold 6126@2.60GHz CPU and 500GB memory in Linux OS. All the methods are implemented in C++ compiled by g++ with O3 turned on.

Datasets. In the experiments, we use eight real-world datasets: Youtube [61], LiveJournal [35], IndoChina [15, 16], Orkut-Links [61], Tags [13], Threads [13], BlockChair [2] and Spotify [33]. All the datasets are available at [3–6]. The first four datasets (i.e. YT, LJ, IC and OL) are unweighted and undirected graphs. Following [65],we convert the four unweighted graphs to motif-based weighted graphs by counting the number of motifs. More precisely, we first calculate the motif number $\phi(e)$ of each edge $e \in E$, which is defined as the number of motifs that e participates in. Then we set the weight of e as $\phi(e)$ to obtain a weighted graph. Note that $\phi(e)$ might be 0 if e doesn't participate in any motif. In the experiments, we set the type of motif as "clique3" defined in [7]. The other four datasets (i.e. TA, TH, BC and SP) are real-world weighted graphs. Specifically, Tags (TA) and Threads (TH) are two question-and-answer (Q&A) datasets. BlockChair (BC) is a bitcoin transaction dataset, and Spotify (SP) is a music streaming dataset. We defer the detailed descriptions of the four real weighted datasets to the technical report [1] due to the space limit.

In Table 2, we list the meta data of all these datasets. We also report the average and maximum edge weight and the value of $\cos^2\varphi$ to quantify the unbalancedness of graphs. Recall that the $\cos^2\varphi$ notation is defined in Lemma 6. The smaller the $\cos^2\varphi$ is, the more unbalanced the graph is.

Ground Truths and Query Sets. In the experiments, we employ Power Method [46] to compute the ground truth results. More precisely, we compute Equation (4) for 100 iterations and regard the returned results as ground truths for comparison. For each dataset, we randomly generate 10 source nodes for SSPPR queries according to the degree distribution. We issue one SSPPR query from each query node and report the average performances over the 10 query nodes for each method and each set of parameters.

6.1 SSPPR with Normalized Additive Error

In this subsection, we evaluate the effectiveness of *EdgePush* with normalized additive error. Furthermore, we apply *EdgePush* to the local clustering application to achieve better efficiency.

Evaluation Metrics. We adopt three metrics for evaluation.

- *normalized MaxAddErr*: In the experiments, we calculate the maximum of the normalized additive error for each node to evaluate the approximation quality of the SSPPR queries. More precisely, we define *normalized MaxAddErr* as $\max_{u\in V}\left|\frac{\boldsymbol{\pi}(u)}{d(u)} - \frac{\hat{\boldsymbol{\pi}}(u)}{d(u)}\right|$, where $\boldsymbol{\pi}(u)$ and $\hat{\boldsymbol{\pi}}(u)$ denote the ground-truth and estimated PPR value of u (w.r.t the source node s by default), respectively.
- *normalized precision@k*: We use *normalized precision@k* to evaluate the performances for identifying top-k results. Specifically, for an SSPPR vector $\boldsymbol{\pi}$, we define $V_k\left(\mathbf{D}^{-1}\boldsymbol{\pi}\right)$ (resp. $V_k\left(\mathbf{D}^{-1}\hat{\boldsymbol{\pi}}\right)$) as the set of the top-$k$ nodes u with the largest $\frac{\boldsymbol{\pi}(u)}{d(u)}$ (resp. $\frac{\hat{\boldsymbol{\pi}}(u)}{d(u)}$) among all nodes in the graph. The *normalized precision@k* is defined as the fraction of nodes in $V_k\left(\mathbf{D}^{-1}\boldsymbol{\pi}\right)$ that concurs with $V_k\left(\mathbf{D}^{-1}\hat{\boldsymbol{\pi}}\right)$. We set $k = 50$ in the experiments.
- Conductance: We employ *conductance* to measure the quality of the clusters, which is defined in Section 2.

Methods. For the SSPPR queries with normalized additive error, we compare our *EdgePush* (dubbed as *EdgePush*-Add) against

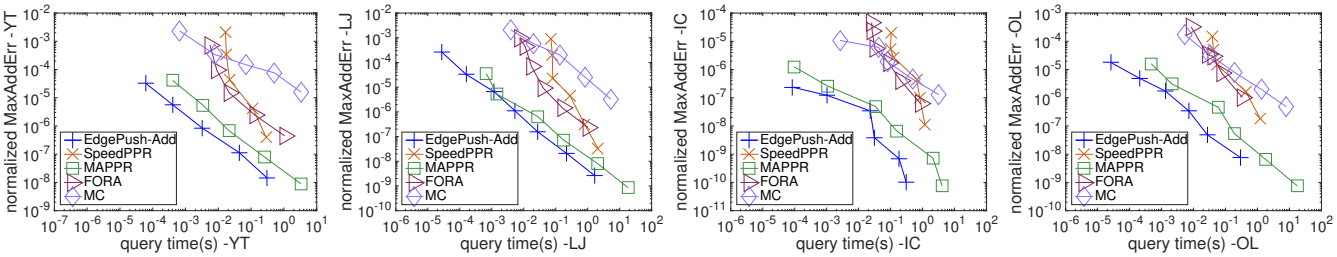

Figure 4: *normalized MaxAddErr* v.s. query time on motif-based weighted graphs.

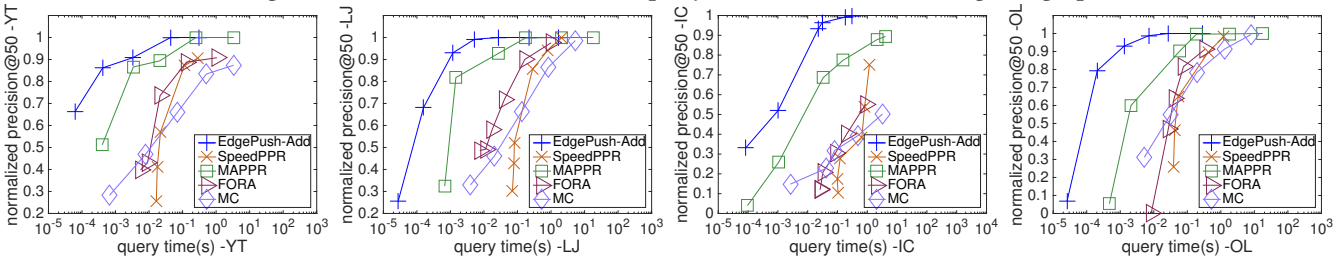

Figure 5: *normalized precision@50* v.s. query time on motif-based weighted graphs.

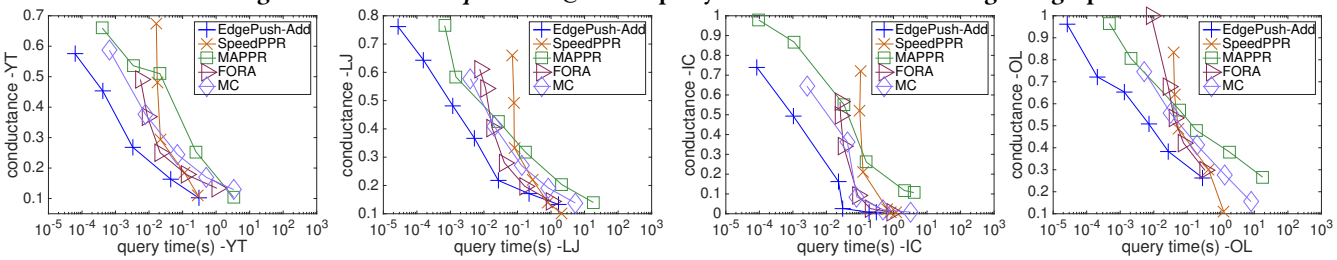

Figure 6: *conductance* v.s. query time on motif-based weighted graphs.

four competitor methods: (i) MAPPR [65]: a version of the *LocalPush* algorithm on weighted graphs; (ii) Monte-Carlo sampling [21, 29]; (iii) FORA [54]: an approximate SSPPR algorithm which combines the strength of *LocalPush* and Monte-Carlo smapling; (iv) SpeedPPR [56]: an approximate SSPPR algorithm which combines PowForPush (an optimized version of Power Method) and Monte-Carlo sampling.

According to Algorithm 1, *LocalPush* only has one parameter: the termination threshold θ. We vary θ in $[10^{-3}, 10^{-9}]$ on both motif-based and real-world weighted graphs. For Monte-Carlo sampling, FORA and SpeedPPR, they all have three parameters: the relative error threshold δ, the relative error ε_r and the failure probability p_f. Following [54, 56], we fix $\varepsilon_r = 0.5$ and $p_f = \frac{1}{n}$, where n is the number of node in the graph. For Monte-Carlo sampling and FORA, we vary δ in $[10^{-1}, 10^{-5}]$. For SpeedPPR, we vary δ in $[5 \times 10^{-1}, 5 \times 10^{-5}]$ on motif-based weighted graphs, and in $[10^{-1}, 10^{-5}]$ on real weighted graphs. For our *EdgePush*, as shown in Algorithm 2, each edge $\langle u, v\rangle \in \bar{E}$ has an individual termination threshold $\theta(u, v)$. According to Theorem 3, we set $\theta(u, v) = \frac{r_{\max} \cdot \sqrt{\mathbf{A}_{uv}}}{\sum_{\langle x,y\rangle \in \bar{E}} \sqrt{\mathbf{A}_{xy}}}$ for normalized additive error $r_{\max}$ and vary $r_{\max}$ from 10^{-3} to 10^{-9}. All of the decay step is 0.1. Additionally, we set the teleport probability α to 0.2 in all the experiments.

Results. In Figure 4 and Figure 7, we draw the trade-off plots between the query time and the normalized maximum additive error (denoted as *normalized MaxAddErr*) on motif-based weighted graphs and real weighted graphs, respectively. Due to the out-of-memory problem, we omit the experimental results of FORA on SP. We observe that under the same *normalized MaxAddErr*, *EdgePush* costs the smallest query time among all these methods on all datasets. In particular, even on Threads (TH) whose $\cos^2(\varphi) = 0.97$, *EdgePush* still outperforms all baselines in terms of query efficiency, which demonstrates the effectiveness of *EdgePush*. Moreover, in Figure 5 and Figure 8, we show the trade-offs between *normalized precision@50* and query time. For the eight datasets, an overall observation is that *EdgePush* outperforms all competitors by achieving higher precision results with less query time. Most notably, on the Orkut-Links (OL) dataset, *EdgePush* achieves a *normalized precision@50* of 0.8 using a query time of 0.0002 seconds, while the closest competitor, MAPPR, achieves a *normalized precision@50* of 0.6 using 0.026 seconds. Additionally, in Figure 8, we observe that compared to the performance of *EdgePush* on TH, the superiority of *EdgePush* over *LocalPush* are more clear on TA, BC and SP. This concurs with the analysis that the superiority of *EdgePush* changes with the unbalancedness of edge weights.

Furthermore, Figure 6 and Figure 9 show the trade-offs between *conductance* and the query time on motif-based and real weighted graphs. Again, our *EdgePush* outperforms other competitors by achieving smaller conductance values under the same query time. Additionally, we note that FORA and SpeedPPR gradually outperforms MAPPR in terms of the query efficiency for *conductance*. However, in the trade-off plots between query time and *normalized MaxAddErr* or *normalized precision@50*, MAPPR costs less query

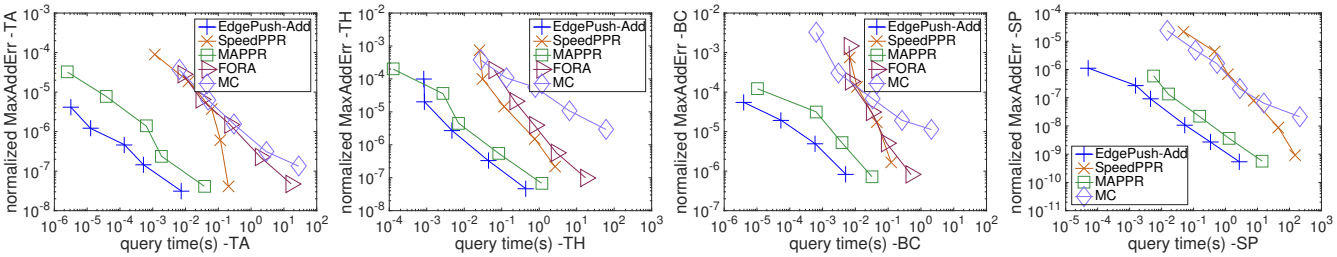

Figure 7: *normalized MaxAddErr* v.s. query time on real weighted graphs.

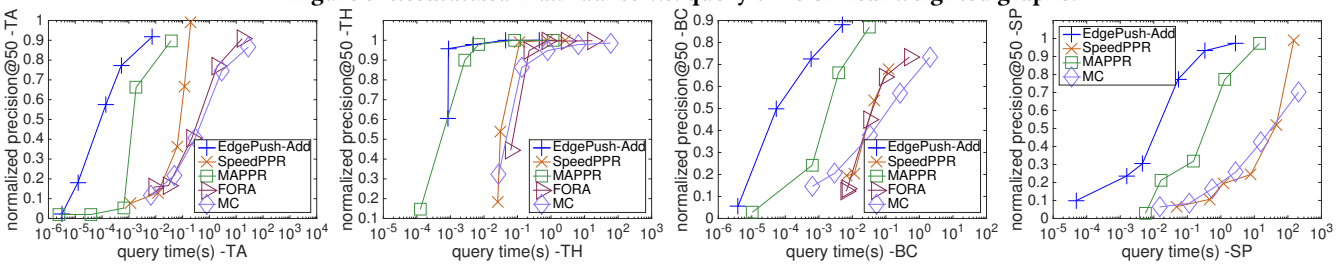

Figure 8: *normalized precision@50* v.s. query time on real weighted graphs.

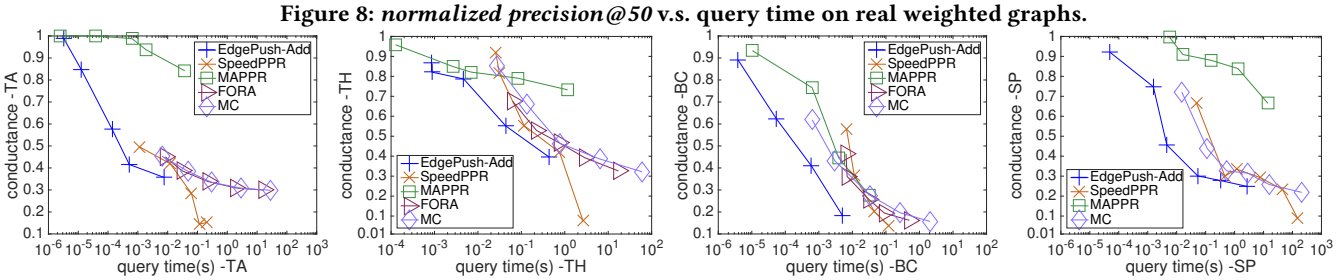

Figure 9: *conductance* v.s. query time on real weighted graphs.

time compared to FORA or SpeedPPR under the same *normalized MaxAddErr* or *normalized precision@50*. Recall that FORA and SpeedPPR all combines *LocalPush* with the Monte-Carlo sampling process. This suggests that the Monte-Carlo sampling method is in favor of the conductance criterion, while the *LocalPush* process benefits from the *normalized MaxAddErr* and *normalized precision@50*.

6.2 SSPPR with ℓ_1-Error

Next, we demonstrate the effectiveness of *EdgePush* with ℓ_1-error.

Evaluation Metrics. To compare the query efficiency of *EdgePush* against other competitors, we adopt three metrics, *actual ℓ_1-error*, *MaxAddErr* and *precision@50*, for overall evaluation.

- *actual ℓ_1-error*: As a classic evaluation metric, *actual ℓ_1-error* is defined as: $\|\hat{\boldsymbol{\pi}} - \boldsymbol{\pi}\|_1 = \sum_{u \in V} |\hat{\boldsymbol{\pi}}_u - \boldsymbol{\pi}_u|$, where $\boldsymbol{\pi}$ and $\hat{\boldsymbol{\pi}}$ is the ground-truth and estimated SSPPR vectors, respectively.
- *MaxAddErr*: To evaluate the maximum additive error of each SSPPR approximation, *MaxAddErr* is defined as $\max_{u \in V} |\boldsymbol{\pi}_u - \hat{\boldsymbol{\pi}}_u|$.
- *precision@50*: To evaluate the relative order of the estimated top-k nodes with the highest SSPPR values, *precision@50* is defined as the percentage of the nodes in $V_k(\hat{\boldsymbol{\pi}})$ that coincides with the actual top-k results $V_k(\boldsymbol{\pi})$. Here $V_k(\boldsymbol{\pi})$ and $V_k(\hat{\boldsymbol{\pi}})$ denote the top-k node sets for the ground-truth and estimated SSPPR values, respectively. Similarly, we set $k = 50$ in the experiments.

Methods. In this section, we compare the performance of *EdgePush* with ℓ_1-error against two algorithms: Power Method [46] and PowForPush [56]. Recall that Power Method computes SSPPR queries by iteratively computing Equation (4). Thus, we vary the number of iterations from 3 to 15 with an interval of 2. PowForPush is the state-of-the-art algorithm for high-precision SSPPR queries, which gradually switches *LocalPush* to Power Method with decreasing ℓ_1-error. Specifically, in the first phase, PowForPush adopts *LocalPush* to compute the SSPPR queries. When the current number of *active* nodes is greater than a specified *scanThreshold*, PowForPush switches to Power Method by performing a sequential scan technique to access active nodes for the *push* operation. A node is called active if its residue is larger than the global termination threshold θ. The rationale behind this switching mechanism is that sequential scan is often more efficient if the number of random access is relatively large. In our experiments, we vary θ from 10^{-3} to 10^{-12} with 0.1 decay step. Inspired by PowForPush, we apply the same switching technique to our *EdgePush* for the fairness of comparison. Specifically, when the number of edges in the candidate set C is significantly great, we switch *EdgePush* to Power Method by performing sequential scanning to access active edges and stop maintaining the two-level structure for each node. An edge $\langle u, v \rangle \in \bar{E}$ is active if its edge residue is larger than the termination threshold $\theta(u, v)$. According to Theorem 3, by setting $\theta(u, v) = \frac{\varepsilon \cdot \sqrt{\mathbf{A}_{uv}}}{\sum_{\langle x,y \rangle \in \bar{E}} \sqrt{\mathbf{A}_{xy}}}$ for $\forall \langle u, v \rangle \in \bar{E}$, *EdgePush* achieves the minimum of the expected overall running time subjected to the ℓ_1-error constraint. To align with the global termination threshold θ adopted in PowForPush, we vary $\theta(u, v)$ for $\forall \langle u, v \rangle \in \bar{E}$ from $\frac{10^{-3} \cdot \|\mathbf{A}\|_1 \cdot \sqrt{\mathbf{A}_{uv}}}{\sum_{\langle x,y \rangle \in \bar{E}} \sqrt{\mathbf{A}_{xy}}}$ to $\frac{10^{-11} \cdot \|\mathbf{A}\|_1 \cdot \sqrt{\mathbf{A}_{uv}}}{\sum_{\langle x,y \rangle \in \bar{E}} \sqrt{\mathbf{A}_{xy}}}$ with 0.1 decay step. To understand the variation

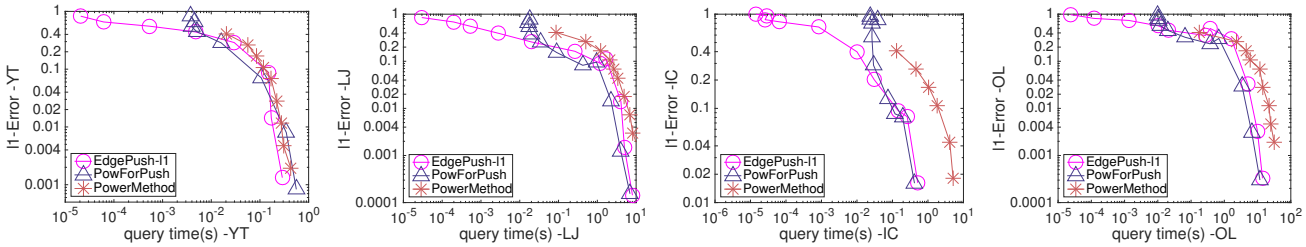

Figure 10: ℓ_1-*error* v.s. query time on motif-based weighted graphs.

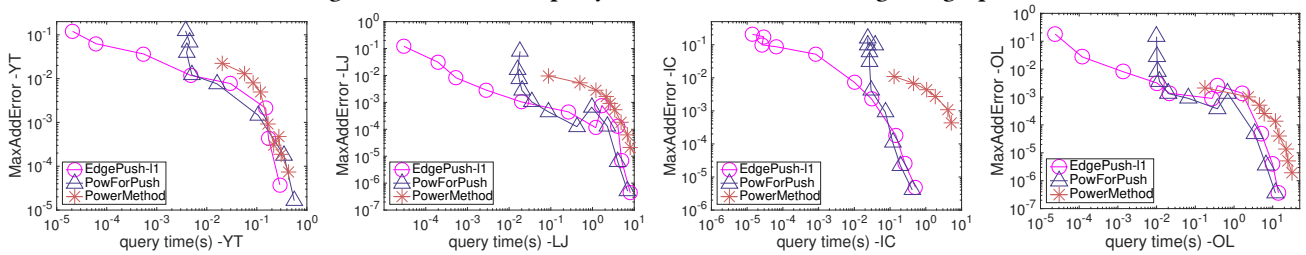

Figure 11: *MaxAddErr* v.s. query time on motif-based weighted graphs.

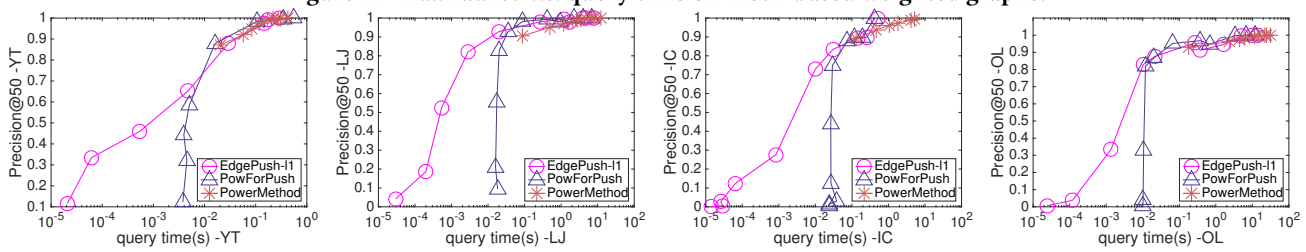

Figure 12: *precision@50* v.s. query time on motif-based weighted graphs.

interval of $\theta(u, v)$, note that on unweighted graphs where $\mathbf{A}_{uv} = 1$, $\theta(u, v)$ is varied in $[10^{-3}, 10^{-12}]$, which concurs with the variation of termination threshold θ in PowForPush.

Results. In Figure 10, we plot the trade-offs between query time and actual ℓ_1-error on motif-based weighted graphs. We observe that under relatively large ℓ_1-error, *EdgePush* costs the smallest query time on all datasets. Additionally, we observe that the plot curves of the three methods gradually overlap with the decreasing of ℓ_1-error. This is because with strict ℓ_1-error constraints, *EdgePush* has to touch most of edges in the graph, which have actually become PowForPush. On the other hand, recall that the ℓ_1-error is defined as the sum of additive error over all nodes in the graph (i.e. $\sum_{u \in V} |\hat{\boldsymbol{\pi}}(u) - \boldsymbol{\pi}(u)|$). Hence, even with some relatively large ℓ_1-error, the (normalized) additive error can be small enough for real applications (e.g., local clustering [10]) Due to the space limit, we defer the experimental results of *EdgePush* with ℓ_1-error on real-world weighted graphs to our technical report [1].

7 CONCLUSION

In this paper, we propose a novel edge-based local push method *EdgePush* for approximating the SSPPR vector on weighted graphs. *EdgePush* decomposes the *push* operation in *LocalPush* into separate *edge-based push* operations, each of which can be performed in $O(1)$ amortized time. We show that when the source node is randomly chosen according to the node degree distribution, the expected running time complexity of *EdgePush* is never worse than that of *LocalPush* within certain ℓ_1-error and normalized additive error. In particular, when the graph is dense and the edge weights are unbalanced, *EdgePush* can achieve a time complexity sub-linear to m, and can outperform *LocalPush* by up to a $O(n)$ factor, where n and m are the numbers of nodes and edges in the graph. Our experimental results show that when achieving the same approximation error, *EdgePush* outperforms *LocalPush* on large-scale real-world datasets by orders of magnitude in terms of efficiency.

ACKNOWLEDGMENTS

Zhewei Wei works at Gaoling School of Artificial Intelligence, Beijing Key Laboratory of Big Data Management and Analysis Methods, MOE Key Lab DEKE and Peng Cheng Laboratory. This research was supported in part by National Natural Science Foundation of China (No. 61972401, No. 61932001, No. 61832017, No. 62072458 and No. 61932004), by the major key project of PCL (PCL2021A12), by Beijing Natural Science Foundation (No. 4222028), by Beijing Outstanding Young Scientist Program No. BJJWZYJH012019100020098, by Alibaba Group through Alibaba Innovative Research Program, by CCF-Baidu Open Fund (No.2021PP15002000), by China Unicom Innovation Ecological Cooperation Plan, and by the Huawei-Renmin University joint program on Information Retrieval. Junhao Gan was supported in part by Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA) DE190101118. Hanzhi Wang was supported by the Outstanding Innovative Talents Cultivation Funded Programs 2020 of Renmin Univertity of China. We also wish to acknowledge the support provided by Intelligent Social Governance Interdisciplinary Platform, Major Innovation & Planning Interdisciplinary Platform for the "Double-First Class" Initiative, Public Policy and Decision-making Research Lab, Public Computing Cloud, Renmin University of China.

REFERENCES

[1] [n.d.]. https://arxiv.org/pdf/2203.07937.pdf.
[2] [n.d.]. https://blockchair.com.
[3] [n.d.]. http://snap.stanford.edu/data.
[4] [n.d.]. http://law.di.unimi.it/datasets.php.
[5] [n.d.]. http://www.cs.cornell.edu/~arb/data/.
[6] [n.d.]. https://gz.blockchair.com/bitcoin-cash/transactions/.
[7] [n.d.]. http://snap.stanford.edu/mappr/code.html.
[8] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng. 2007. Local computation of PageRank contributions. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 150–165.
[9] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcroft, Kamal Jain, Vahab Mirrokni, and Shanghua Teng. 2008. Robust pagerank and locally computable spam detection features. In *Proceedings of the 4th international workshop on Adversarial information retrieval on the web*. 69–76.
[10] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *FOCS*. 475–486.
[11] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, and Natalia Osipova. 2007. Monte Carlo methods in PageRank computation: When one iteration is sufficient. *SIAM J. Numer. Anal.* 45, 2 (2007), 890–904.
[12] Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*. 635–644.
[13] Austin R Benson, Rediet Abebe, Michael T Schaub, Ali Jadbabaie, and Jon Kleinberg. 2018. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences* 115, 48 (2018), E11221–E11230.
[14] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA.
[15] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In *Proceedings of the 20th international conference on World Wide Web*, Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar (Eds.). ACM Press, 587–596.
[16] Paolo Boldi and Sebastiano Vigna. 2004. The WebGraph Framework I: Compression Techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*. ACM Press, Manhattan, USA, 595–601.
[17] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable Graph Neural Networks via Bidirectional Propagation. *arXiv preprint arXiv:2010.15421* (2020).
[18] Fan Chung and Olivia Simpson. 2015. Distributed algorithms for finding local clusters using heat kernel pagerank. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 177–189.
[19] Fan Chung and Olivia Simpson. 2018. Computing heat kernel pagerank and a local clustering algorithm. *European Journal of Combinatorics* 68 (2018), 96–119.
[20] Wei Feng and Jianyong Wang. 2012. Incorporating heterogeneous information for personalized tag recommendation in social tagging systems. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1276–1284.
[21] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics* 2, 3 (2005), 333–358.
[22] Kimon Fountoulakis, Farbod Roosta-Khorasani, Julian Shun, Xiang Cheng, and Michael W Mahoney. 2019. Variational perspective on local graph clustering. *Mathematical Programming* 174, 1-2 (2019), 553–573.
[23] Michael L Fredman and Dan E Willard. 1993. Surpassing the information theoretic bound with fusion trees. *Journal of computer and system sciences* 47, 3 (1993), 424–436.
[24] Yasuhiro Fujiwara, Makoto Nakatsuji, Takeshi Yamamuro, Hiroaki Shiokawa, and Makoto Onizuka. 2012. Efficient personalized pagerank with accuracy assurance. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 15–23.
[25] Bin Gao, Tie-Yan Liu, Wei Wei, Taifeng Wang, and Hang Li. 2011. Semi-supervised ranking on very large graphs with rich metadata. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 96–104.
[26] Qing Guo, Zhu Sun, Jie Zhang, Qi Chen, and Yin-Leng Theng. 2017. Aspect-aware point-of-interest recommendation with geo-social influence. In *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*. 17–22.
[27] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. 2013. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*. 505–514.
[28] Guanhao Hou, Xingguang Chen, Sibo Wang, and Zhewei Wei. 2021. Massively Parallel Algorithms for Personalized PageRank. *PROCEEDINGS OF THE VLDB ENDOWMENT* 14, 9 (2021), 1668–1680.
[29] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*. 271–279.
[30] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.
[31] Kyle Kloster and David F Gleich. 2014. Heat kernel based community detection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1386–1395.
[32] Donald E Knuth. 1998. The Art of computer programming, Volume 3: Sorting and searching (1973). *Google Scholar Google Scholar Digital Library Digital Library* (1998).
[33] Raunak Kumar, Paul Liu, Moses Charikar, and Austin R Benson. 2020. Retrieving Top Weighted Triangles in Graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 295–303.
[34] Preethi Lahoti, Gianmarco De Francisci Morales, and Aristides Gionis. 2017. Finding topical experts in Twitter via query-dependent personalized PageRank. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*. 155–162.
[35] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.
[36] Dandan Lin, Raymond Chi-Wing Wong, Min Xie, and Victor Junqiu Wei. 2020. Index-Free Approach with Theoretical Guarantee for Efficient Random Walk with Restart Query. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 913–924.
[37] Wenqing Lin. 2019. Distributed algorithms for fully personalized pagerank on large graphs. In *The World Wide Web Conference*. 1084–1094.
[38] Peter Lofgren. 2015. *EFFICIENT ALGORITHMS FOR PERSONALIZED PAGERANK*. Ph.D. Dissertation. STANFORD UNIVERSITY.
[39] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2015. Bidirectional pagerank estimation: From average-case to worst-case. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 164–176.
[40] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized pagerank estimation and search: A bidirectional approach. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. 163–172.
[41] Peter Lofgren and Ashish Goel. 2013. Personalized pagerank to a target node. *arXiv preprint arXiv:1304.4658* (2013).
[42] Peter A Lofgren, Siddhartha Banerjee, Ashish Goel, and C Seshadhri. 2014. Fast-ppr: Scaling personalized pagerank estimation for large graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1436–1445.
[43] Siqiang Luo. 2019. Distributed pagerank computation: An improved theoretical study. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4496–4503.
[44] Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*. 849–856.
[45] Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. 2015. Efficient pagerank tracking in evolving networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 875–884.
[46] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: bringing order to the web. (1999).
[47] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. 2013. Fast distributed pagerank computation. In *International Conference on Distributed Computing and Networking*. Springer, 11–26.
[48] Daniel A Spielman and Shang-Hua Teng. 2004. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC*. 81–90.
[49] J Michael Steele. 2004. *The Cauchy-Schwarz master class: an introduction to the art of mathematical inequalities*. Cambridge University Press.
[50] Mikkel Thorup. 1995. *Equivalence between sorting and priority queues*. Technical Report. Citeseer.
[51] Chu Wang, Babak Samari, Vladimir G Kim, Siddhartha Chaudhuri, and Kaleem Siddiqi. 2020. Affinity graph supervision for visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8247–8255.
[52] Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. 2020. Personalized pagerank to a target node, revisited. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 657–667.
[53] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. Hubppr: effective indexing for approximate personalized pagerank. *Proceedings of the VLDB Endowment* 10, 3 (2016), 205–216.

[54] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: simple and effective approximate single-source personalized pagerank. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 505–514.

[55] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. 2018. Topppr: top-k personalized pagerank queries with precision guarantees on large graphs. In *Proceedings of the 2018 International Conference on Management of Data*. 441–456.

[56] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. *arXiv preprint arXiv:2101.03652* (2021).

[57] Wenlei Xie, David Bindel, Alan Demers, and Johannes Gehrke. 2015. Edge-weighted personalized pagerank: Breaking a decade-old performance barrier. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1325–1334.

[58] Wenpu Xing and Ali Ghorbani. 2004. Weighted pagerank algorithm. In *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.* IEEE, 305–314.

[59] Rakesh Kumar Yadav, Shekhar Verma, S Venkatesan, et al. 2021. Cross-covariance based affinity for graphs. *Applied Intelligence* 51, 6 (2021), 3844–3864.

[60] Carl Yang and Kevin Chang. 2019. Relationship profiling over social networks: Reverse smoothness from similarity to closeness. In *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 342–350.

[61] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (2015), 181–213.

[62] Lei Yang, Xiaohang Zhan, Dapeng Chen, Junjie Yan, Chen Change Loy, and Dahua Lin. 2019. Learning to cluster faces on an affinity graph. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2298–2306.

[63] 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 *Proceedings of the 2019 International Conference on Management of Data*. 1339–1356.

[64] Jianfeng Ye, Qilin Li, Jinlong Yu, Xincheng Wang, and Huaming Wang. 2020. Affinity Learning Via Self-Supervised Diffusion for Spectral Clustering. *IEEE Access* 9 (2020), 7170–7182.

[65] Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. 2017. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 555–564.

[66] Weiren Yu and Julie McCann. 2016. Random walk with restart over dynamic graphs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 589–598.

[67] Hongyang Zhang, Peter Lofgren, and Ashish Goel. 2016. Approximate personalized pagerank on dynamic graphs. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1315–1324.

[68] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. *Advances in neural information processing systems* 16, 16 (2004), 321–328.

[69] Xiatian Zhu, Chen Change Loy, and Shaogang Gong. 2014. Constructing robust affinity graphs for spectral clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1450–1457.

[70] Yangbo Zhu, Shaozhi Ye, and Xing Li. 2005. Distributed PageRank computation based on iterative aggregation-disaggregation methods. In *Proceedings of the 14th ACM international conference on Information and knowledge management*. 578–585.