Accelerating Personalized PageRank Vector Computation

Zhen Chen
zhenchen21@m.fudan.edu.cn
Fudan University,
Shanghai Key Laboratory of Data
Science, Shanghai, China

Xingzhi Guo
xingzguo@cs.stonybrook.edu
Stony Brook University
Stony Brook, USA

Baojian Zhou[*]
bjzhou@fudan.edu.cn
Fudan University,
Shanghai Key Laboratory of Data
Science, Shanghai, China

Deqing Yang
yangdeqing@fudan.edu.cn
Fudan University,
Shanghai Key Laboratory of Data
Science, Shanghai, China

Steven Skiena
skiena@cs.stonybrook.edu
Stony Brook University
Stony Brook, USA

ABSTRACT

Personalized PageRank Vectors are widely used as fundamental graph-learning tools for learning graph embeddings, training graph neural networks, and detecting anomalous spammers. The well-known *local* FwdPush algorithm [6] approximates PPVs and has a sublinear rate of $O\left(\frac{1}{\alpha\epsilon}\right)$. A recent study [89] found that when high precision is required, FwdPush is similar to the power iteration method, and its run time is pessimistically bounded by $O\left(\frac{m}{\alpha}\log\frac{1}{\epsilon}\right)$. This paper looks closely at calculating PPVs for both directed and undirected graphs. By leveraging the *linear invariant property*, we show that FwdPush is a variant of Gauss-Seidel and propose a Successive Over-Relaxation based method, FwdPushSOR to speed it up by slightly modifying FwdPush. Additionally, we prove Fwd-Push has *local linear convergence* rate $O\left(\frac{\text{vol}(\mathcal{S})}{\alpha}\log\frac{1}{\epsilon}\right)$ enjoying advantages of two existing bounds. We also design a new local heuristic push method that reduces the number of operations by 10-50 percent compared to FwdPush. For undirected graphs, we propose two momentum-based acceleration methods that can be expressed as one-line updates and speed up nonacceleration methods by $O(1/\sqrt{\alpha})$. Our experiments on six real-world graph datasets confirm the efficiency of FwdPushSOR and the acceleration methods for directed and undirected graphs, respectively.

KEYWORDS

Personalized PageRank, large-scale graph, local linear convergence, Successive Over-Relaxation

[*]Corresponding Author.

1 INTRODUCTION

As fundamental graph-learning tools, Personalized PageRank Vectors (PPVs) [52] have been widely used in classic graph mining tasks such as detecting anomalous spammers [4, 5, 14, 81], and modern graph representation learning methods such as graph embeddings [44, 76, 86] and graph neural networks [17, 18, 25, 31, 35, 36, 46, 59, 84, 85, 93]. PPVs effectively capture the local proximity of graph nodes, making them useful for training improved graph neural network models and designing effective clustering algorithms [6]. As a result, efficient computation of PPVs is crucial for the current field of graph representation learning.

The well-known FwdPush algorithm [6, 15] is a widely used tool for computing PPVs due to its effectiveness in approximating PPVs, easy implementation, and local nature. The cost of each iteration of FwdPush is dependent only on the volumes of nodes near the target node, and its total run time complexity can be bounded by $O\left(\frac{1}{\alpha\epsilon}\right)$, where α is the damping factor, and precision parameter ϵ controls the precision of the per-entry of PPV. This time complexity bound is independent of the graph structure, making FwdPush a preferred method over the power iteration method, which requires access to the entire graph in each iteration. However, a recent study [89] showed that when high precision is required, FwdPush behaves more like the power iteration method, with a pessimistically bounded run time complexity of $O\left(\frac{m}{\alpha}\log\frac{1}{\epsilon}\right)$, where m is the number of edges in the graph. This bound only holds for $\epsilon < (2m)^{-1}$, and it is unclear whether there exists a logarithmic factor bound for $\epsilon \geq (2m)^{-1}$.

A natural question we address in this paper is **Q1.** whether Fwd-Push has a *locally linear convergence rate*, meaning that the per-epoch complexity is locally dependent on the graph, but the total number of epochs is still bounded by a logarithmic factor $O\left(\log\frac{1}{\epsilon}\right)$. To answer this question, a key observation is that when α is close to 1, the local push method will not explore nodes far from the current target node, and thus the total run time per epoch remains local. It has been proven that when the graph is undirected, Fwd-Push is essentially a coordinate descent method, and computing an approximate PPV corresponds to an ℓ_1-regularized optimization problem, which has a sparse solution [33]. This suggests that

FwdPush is truly local. However, this equivalence is based on the assumption that the graph is undirected and is still unknown **Q2.** whether a similar optimization algorithm equivalence exists for directed graphs.

Questions Q1 and Q2 motivate us to study the PPV computation further. In this paper, we show for the first time that the well-known FwdPush algorithm is a variant of Gauss-Seidel when the underlying graph is directed. This is due to the *linear invariant property* of FwdPush, which means that the updates of Gauss-Seidel for each coordinate of the target linear system are equal to the residual updates of FwdPush. We then propose to use the Successive Over Relaxation (SOR) based method to speed up FwdPush, namely FwdPushSOR. The advantage of FwdPushSOR is that it speeds up the original method, and it can be naturally applied to other variants of FwdPush [89], even ones for dynamic graph settings [92]. Furthermore, to study the convergence rate better, we prove a *locally linear convergence rate* $O\left(\frac{\text{vol}(\mathcal{S})}{\alpha} \log \frac{1}{\epsilon}\right)$ where the precision is any positive number $\epsilon > 0$ and $\text{vol}(\mathcal{S})$ is the expected volume of a subset of *active nodes* explored. Our analysis is simplified by adding a dummy node to the queue and considering only the non-zero residual nodes. For undirected graphs with small α, momentum acceleration-based methods can accelerate by a factor of $O(1/\sqrt{\alpha})$. Both acceleration-based methods can be implemented in one-line iteration updates. Our contributions are summarized as follows

- For the first time, we show that the well-known FwdPush algorithm is a variant of Gauss-Seidel. To further improve its performance, we propose FwdPushSOR, *a speed-up local method* of the original FwdPush based on the SOR technique. FwdPushSOR is as simple as that of FwdPush.
- We prove a *locally linear convergence rate* of FwdPush with a complexity of $O\left(\frac{\text{vol}(\mathcal{S})}{\alpha} \log \frac{1}{\epsilon}\right)$, which combines the advantages of existing bounds where the expected volume of explored nodes, $\text{vol}(\mathcal{S})$, is locally dependent on the underlying graph. Based on this insight, we design FwdPushMean, a new local push method variant that reduces the number of operations by 10-50% for most target nodes.
- For undirected graphs, we propose using the Heavy-Ball (HB) and Nesterov Acceleration Gradient (NAG) method to calculate high-precision PPR vectors, which are $O(1/\sqrt{\alpha})$ times faster than the power iteration and FwdPush variants. Our methods can be implemented in just one line of code.
- We conduct experiments on six real-world graphs and find that, FwdPushSOR and acceleration methods can significantly reduce run time and the number of operations required. For example, these local SOR methods are about 3 times faster on undirected graphs and about 2 times faster on directed graphs, respectively, when $\alpha = 0.15$.

The rest of the paper is organized as follows: In Sec. 2, we discuss related work. Notations and the definition of PPV are presented in Sec. 3. Sec. 4 gives the locality analysis of FwdPush. The accelerated algorithms for PPVs are presented in Sec. 5. Finally, we present our experimental results and conclusions in Sec. 6 and Sec. 7, respectively. Our code and datasets are available at https://github.com/ccczhen/AccPPR.

2 RELATED WORK

PPVs and power iteration-based accelerations. The Personalized PageRank computation traces its roots to the work of Jeh and Widom [52], who proposed using a personalized vector as a starting point for PageRank calculation instead of the uniform distribution used in the original PageRank algorithm [74]. Acceleration methods for directed graphs are mainly based on power iteration, accessing the whole graph once per iteration, resulting in $O(m)$ per-iteration updates. Arasu et al. [9] used Gauss-Seidel method to globally accelerate power iteration. Kamvar et al. [55] proposed to use the Aitken extrapolation to speed up the PPVs calculation, but its effectiveness largely depends on the underlying graph and can sometimes lead to insignificant improvements. Other acceleration techniques, including the inner-outer loop approach [39], and methods proposed by Lee et al. [61] and systematically studied by Langville and Meyer [60], have also been proposed to improve the power iteration-based method. Different from these methods, this paper mainly focuses on accelerating local methods.

FwdPush and its variants. The work of Andersen et al. [6] proposed FwdPush (also known as an Approximate PageRank Approximation, APPR) algorithm for approximating PageRank personalization vectors. Later, it was used to approximate columns of the PPV matrix [4]. The algorithm has a sublinear time complexity bound of $O\left(\frac{1}{\epsilon\alpha}\right)$ due to significant residuals being pushed out from the residual vector per iteration. It is worth mentioning that the essentially same idea as the local push method is also proposed in Berkhin [15]. Recent work by Wu et al. [89] showed that for $\epsilon < (2m)^{-1}$, FwdPush converges more like power iteration methods, leading to a total run time complexity of $O\left(\frac{m}{\alpha} \log \frac{1}{\epsilon}\right)$. However, it remains unknown if there is an exponential improvement when $\epsilon > (2m)^{-1}$, which corresponds to local approximation. We take a step further in this direction.

A recent study by Fountoulakis et al. [33] found that computing PPV is equivalent to solving an ℓ_1-regularized problem that can be treated as a variant of the coordinate descent algorithm. The reason is that the linear system can be reformulated as a quadratic strongly convex optimization problem when the graph is undirected. However, this analysis only works for undirected graphs, as the objective of the optimization requires a symmetric matrix. It is unclear how to apply the analysis to directed graph settings. When the graph is undirected, alternative global methods, such as the conjugate gradient method, exist but are complex to implement. Therefore, we explore using NAG and HB-based methods to speed up the computation. The question of whether there exists a locally independent bound $O\left(\frac{1}{\sqrt{\alpha}\epsilon}\right)$ for the accelerated methods such as Accelerated Coordinate Descents [3, 62], linear coupling [2] remains open asked by Fountoulakis and Yang [34].

Other related methods. Many Personalized PageRank-related algorithms [38, 48, 52, 64, 87] have been proposed, including for dynamic graph settings [12, 92]. These works mainly focus on computing the PPR for a single or subset of entries. The generalized PageRank problem has been reviewed in [38] and symmetrically studied [60, 63]. One can find more details in [38, 60, 63] and references therein. Our technique may be of independent interest to these directions.

3 PRELIMINARY

Notations. We consider unweighted simple graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{1, 2, \ldots, n\}$ is a set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges with $m = |\mathcal{E}|$. The underlying graph $\mathcal{G}$ is either a directed graph or an undirected graph depending on the context. Bold lower letters are column vectors, e.g., $\boldsymbol{p} \in \mathbb{R}^n$. Bold capital letters, e.g., $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ are matrices. $\boldsymbol{D} = \text{diag}(d_1, d_2, \ldots, d_n)$ is the diagonal out-degree matrix of $\mathcal{G}$.[1] The minimal and maximal degree is denoted as $d_{\min}$ and $d_{\max}$. $\text{Nei}(u)$ is the set of neighbors of u. $\boldsymbol{A}$ is denoted as the adjacency matrix of $\mathcal{G}$. The column stochastic matrix associated with $\boldsymbol{A}$ is defined as $\boldsymbol{P} := \boldsymbol{A}^\top \boldsymbol{D}^{-1}$.[2] The teleportation parameter (a.k.a *dumping factor*) $\alpha \in (0, 1)$ (usually $\alpha \in (0.0, 0.5)$ in practice). A vector $\boldsymbol{r}$ at time t is denoted as $\boldsymbol{r}^t := [r_1^t, r_2^t, \ldots, r_n^t]^\top$. The volume of $\mathcal{S} \subseteq \mathcal{V}$ is defined as $\text{vol}(\mathcal{S}) \triangleq \sum_{v \in \mathcal{S}} d_v$. The support of $\boldsymbol{r}$ is the set of nonzero indices, i.e., $\text{supp}(\boldsymbol{r}) = \{v : r_v \neq 0, v \in \mathcal{V}\}$. For any matrix $\boldsymbol{M}$, we denote m_{ij} as the element of $\boldsymbol{M}$ at i-th row and j-th column. An indicating vector $\boldsymbol{e}_v \in \{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\}$ has value 1 in v-th entry and 0 otherwise. Similarly, $(\boldsymbol{a})_u$ is an indicator vector with a_u at u-th column and 0 otherwise. $\boldsymbol{M}_{i,:}$ is the i-th row of $\boldsymbol{M}$.

3.1 Personalized PageRank Vector

Given an underlying graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an initial indicating vector $\boldsymbol{e}_s$, and a teleportation parameter $\alpha \in (0, 1)$, PPV of a target node s is a probability vector $\boldsymbol{x}$ such that

$$\boldsymbol{x} = \alpha \boldsymbol{e}_v + (1 - \alpha) \boldsymbol{A}^\top \boldsymbol{D}^{-1} \boldsymbol{x}, \qquad (1)$$

where we call $\boldsymbol{x}$ satisfying Equ. (1) a PPV. The above equation is essential to access the v-th column of a nonnegative matrix, that is

$$\boldsymbol{x} = \alpha \left(\boldsymbol{I} - (1 - \alpha) \boldsymbol{A}^\top \boldsymbol{D}^{-1} \right)^{-1} \boldsymbol{e}_v.$$

The definition of PPV is a generalization of Google matrix computation where $\boldsymbol{e}_s = \boldsymbol{1}/n$ used [74]. The standard method of solving the linear system (1) is the fixed-point iteration working as the following

$$\boldsymbol{x}^{t+1} = (1 - \alpha) \boldsymbol{A}^\top \boldsymbol{D}^{-1} \boldsymbol{x}^t + \alpha \boldsymbol{e}_v.$$

As shown in [40], if we use $\boldsymbol{x}^0 = 0$, one immediately obtains that $\|\boldsymbol{x}^t - \boldsymbol{x}^*\|_1 = (1 - \alpha)^t$ where we denote $\boldsymbol{x}^*$ as the true solution of the PPV for the target node v. However, the above iteration method needs to access the whole graph, thus resulting in $O(m)$ run time complexity per iteration. Next, we introduce the local FwdPush method and explain how it can obtain a good approximation of PPV by only exploring a small set of nodes.

3.2 FwdPush algorithm

An efficient first-in-first-out queue-based implementation of Fwd-Push is presented in Algo. 1. At a higher level, FwdPush iteratively accesses nodes and their neighbors and moves a distribution $\boldsymbol{r}$ to another distribution $\boldsymbol{x}$. At each iteration, each node u in Q satisfies $r_u \geq \epsilon d_u$. We call such a node u an *active node*. Similarly, if a node v has a low residual, i.e., $r_v < \epsilon d_v$, then it is *inactive*. Initially, Fwd-Push has the source node s in Q. With each iteration, it maintains the updates of two vectors $\boldsymbol{r}$ and $\boldsymbol{x}$, where $\boldsymbol{r}$ is a *residual vector*, and $\boldsymbol{x}$ is the *estimation vector*. For each active node u, it pushes α magnitudes of r_u to $\boldsymbol{x}$ and spreads $(1 - \alpha)r_u$ uniformly to $\text{Nei}(u)$.

Algorithm 1 FwdPush$(\mathcal{G}, \epsilon, \alpha, s)$

1: Initialization: $\boldsymbol{r} = \boldsymbol{e}_s, \boldsymbol{x} = \boldsymbol{0}$
2: $Q = [s]$
3: **while** $Q \neq \emptyset$ **do**
4:   $u = Q.\text{pop}()$
5:   $x_u = x_u + \alpha \cdot r_u$
6:   **for** $v \in \text{Nei}(u)$ **do**
7:     $r_v = r_v + \frac{(1-\alpha)r_u}{d_u}$
8:     **if** $r_v \geq \epsilon d_v$ & $v \notin Q$ **then**
9:       $Q.\text{push}(v)$
10:   $r_u = 0$
11: **Return** $\boldsymbol{x}$

It terminates when all $r_u < \epsilon d_u$. During these push operations, one always have $r_u, x_u \geq 0$ and $\|\boldsymbol{r}\|_1 + \|\boldsymbol{x}\|_1 = 1$. It can be shown that returned $\boldsymbol{x}$ is guaranteed by $|x_u - x_u^*| \leq \epsilon d_u, \forall u \in \mathcal{V}$ (See details in [6]). The essential effectiveness of FwdPush is because entries $\boldsymbol{x}$ indexing by nodes near to s have large magnitudes, and entries of $\boldsymbol{x}$ follow a power-law distribution as demonstrated in Fig. 7 in the Appendix. Therefore, FwdPush quickly approximate $\boldsymbol{x}$ by only exploring these nodes *near* to it. We aim to improve the speed of this local method. In the following section, we will present our key findings.

4 LOCALITY ANALYSIS OF FWDPUSH

This section presents the equivalence between FwdPush and a variant of Gauss-Seidel (G-S) and introduces a faster method based on the SOR technique. We demonstrate that FwdPush has a *locally linear convergence rate* and offer insights that could help find more effective variants.

4.1 FwdPush is a variant of Gauss-Seidel

To show that FwdPush is a variant of G-S iteration (See Section 10.1.1 of [41]). Recall, for solving the linear system

$$\boldsymbol{Mx} = \boldsymbol{b}, \qquad (2)$$

the G-S iteration updates $\boldsymbol{x}$ using the following online iteration

$$x_u^{t+1} = \frac{1}{m_{uu}} \left(b_u - \sum_{j=1}^{u-1} m_{uj} x_j^{t+1} - \sum_{j=u+1}^{n} m_{uj} x_j^t \right), u \in \mathcal{S}_t, \quad (3)$$

where t indexes the current super-iteration, x_j^{t+1} are elements have been updated up to time j^{t+1}, and x_j^t are entries will be updated. $\mathcal{S}^t$ presents the set of indices of $\boldsymbol{x}$ updated in t-th super-iteration. Note that we have $\mathcal{S}_t = \mathcal{V}$ for all t in the standard G-S iteration. G-S iteration is usually for solving system (2) when $\boldsymbol{M}$ is *strictly diagonally-dominant*. We note that $\boldsymbol{I} - (1 - \alpha)\boldsymbol{A}^\top \boldsymbol{D}^{-1}$ is a strictly diagonally-dominant matrix for a simple graph. The following theorem presents the equivalence between the variant of the Gauss-Seidel iteration and the FwdPush algorithm.

Theorem 1 (FwdPush is Gauss-Seidel). *Each iteration updates $\boldsymbol{r}$ and $\boldsymbol{x}$ in Algo. 1 of the FwdPush$(\mathcal{G}, \epsilon, \alpha, s)$ algorithm is equivalent to an iteration of the Gauss-Seidel iteration as defined in (3) when $\boldsymbol{b} = \alpha \boldsymbol{e}_s$ and $\boldsymbol{M} = \boldsymbol{I} - (1 - \alpha)\boldsymbol{A}^\top \boldsymbol{D}^{-1}$. Furthermore, $\mathcal{S}_t$ corresponds to the set of active nodes processed in Algo. 1 at t-th epoch.* [3]

PROOF. The key to our proof is to use the *linear invariant property*. We state it as follows: Let $\boldsymbol{x}^t$ and $\boldsymbol{r}^t$ be the estimation and

---

[1] See Appendix A for dealing with dangling nodes.
[2] In case of $d_v = 0$ for some v, $\boldsymbol{D}^{-1} = \boldsymbol{D}^+$ where $\boldsymbol{D}^+$ is Moore–Penrose inverse of $\boldsymbol{D}$.

[3] We will define an epoch of FwdPush by adding a dummy node presented in the next subsection.

residual vector of calling FwdPush($\mathcal{G}, \epsilon, \alpha, s$) at time t, then for all $t \geq 0$, we have the following linear invariant property

$$\alpha \boldsymbol{r}^t = \alpha \boldsymbol{r}^0 - \left(\boldsymbol{I} - (1-\alpha)\boldsymbol{A}^\top \boldsymbol{D}^{-1}\right)\boldsymbol{x}^t. \tag{4}$$

To verify Equ. (4), note that it is trivially true at the initial time $t = 0$ where $\boldsymbol{x}^0 = \boldsymbol{0}$. For all $t \geq 1$ and any active node u, notice that FwdPush updates $\boldsymbol{x}^{t-1}$ and $\boldsymbol{r}^{t-1}$ as the following

$$\boldsymbol{x}^t = \boldsymbol{x}^{t-1} + \alpha r_u^{t-1} \cdot \boldsymbol{e}_u \tag{5}$$

$$\boldsymbol{r}^t = \boldsymbol{r}^{t-1} - r_u^{t-1} \cdot \boldsymbol{e}_u + (1-\alpha)r_u^{t-1}\boldsymbol{A}^\top \boldsymbol{D}^{-1}\boldsymbol{e}_u, \tag{6}$$

where Equ. (5) corresponds to Line 5 and Equ. (6) represents Line6-10 of Algo. 1 with the initial setup $\boldsymbol{x}^0 = \boldsymbol{0}, \boldsymbol{r}^0 = \boldsymbol{e}_s$. To simplify Equ. (6), one can reformulate it as

$$\alpha r_u^{t-1}\boldsymbol{e}_u = \alpha \left(\boldsymbol{I} - (1-\alpha)\boldsymbol{A}^\top \boldsymbol{D}^{-1}\right)^{-1}\left(\boldsymbol{r}^{t-1} - \boldsymbol{r}^t\right).$$

$\boldsymbol{x}^t$ is thus the sum of the left-hand side of the above over t, that is,

$$\boldsymbol{x}^t = \alpha \sum_{i=1}^{t} r_u^{i-1}\boldsymbol{e}_u = \alpha \left(\boldsymbol{I} - (1-\alpha)\boldsymbol{A}^\top \boldsymbol{D}^{-1}\right)^{-1} \sum_{i=1}^{t}(\boldsymbol{r}^{i-1} - \boldsymbol{r}^i)$$

$$= \alpha \left(\boldsymbol{I} - (1-\alpha)\boldsymbol{A}^\top \boldsymbol{D}^{-1}\right)^{-1}\left(\boldsymbol{r}^0 - \boldsymbol{r}^t\right).$$

Move the above-inverted matrix to the left; we see the linear invariant property (4) is valid. Next, we show FwdPush is a variant type of G-S iteration defined in (3). Since we assume $\boldsymbol{M} = \boldsymbol{I} - (1-\alpha)\boldsymbol{A}^\top \boldsymbol{D}^{-1}$ and $\boldsymbol{b} = \alpha\boldsymbol{e}_s$, then $m_{uu} = 1$ for the simple graph. The G-S iteration of (3) can be rewritten as

$$x_u^{t+1} = x_u^t + \left(b_u - \sum_{j=1}^{n} m_{uj}x_j^t\right), \quad //x_j^t = x_j^{t+1} \text{ for } j < u$$

where we can use $\boldsymbol{x}^t$ to represent the updated $\boldsymbol{x}$ up to time i^t. Hence, each update can be represented as a vector form as the following

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t + \left(\boldsymbol{b} - \boldsymbol{M}_{u,:}\boldsymbol{x}^t\right)_u$$

$$= \boldsymbol{x}^t + \left(\alpha\boldsymbol{r}^0 - \left(\boldsymbol{I} - (1-\alpha)\boldsymbol{A}^\top \boldsymbol{D}^{-1}\right)\boldsymbol{x}^t\right)_u$$

$$= \boldsymbol{x}^t + \alpha r_u^t\boldsymbol{e}_u,$$

where the second equality is from the definition of $\boldsymbol{b}$ and $\boldsymbol{M}$, and the last equality follows from the linear invariant property (4). □

When α is small, $\boldsymbol{M}$ has a large condition number corresponding to slow convergence of FwdPush and G-S. Fortunately, Thm. 1 immediately tells us that to speed up FwdPush, the acceleration technique used for G-S can also be applied for FwdPush. To speed up the G-S procedure, we propose to use the Successive Over-Relaxation (SOR) technique [45, 91], a well-known method for accelerating G-S of solving diagonally-dominant matrix. To update $\boldsymbol{x}^t$, by using SOR, we have (note $m_{uu} = 1$)

$$\boldsymbol{x}^{t+1} = (1-\omega)\boldsymbol{x}^t + \omega\left(b_u - \sum_{j=1}^{u-1} m_{uj}x_j^{t+1} - \sum_{j=u+1}^{n} m_{uj}x_j^t\right) \cdot \boldsymbol{e}_u$$

$$= (1-\omega)\boldsymbol{x}^t + \omega\boldsymbol{x}^t + \omega(\boldsymbol{b} - \boldsymbol{M}_{u,:}\boldsymbol{x}^t)_u$$

$$= \boldsymbol{x}^t + \omega\alpha r_u^t\boldsymbol{e}_u,$$

where the relaxation parameter $\omega \in (0, 2)$. The relaxed method is simply different by ω times. The next key is maintaining the invariant property for $\boldsymbol{x}^t$ and $\boldsymbol{r}^t$. To maintain the linear invariant

property, we apply Equ. (5) and (6) ω times of original magnitudes. Hence, the corresponding residual updates of FwdPush become

$$\boldsymbol{x}^t = \boldsymbol{x}^{t-1} + \omega\alpha r_u^{t-1} \cdot \boldsymbol{e}_u \tag{7}$$

$$\boldsymbol{r}^t = \boldsymbol{r}^{t-1} - \omega r_u^{t-1} \cdot \boldsymbol{e}_u + \omega(1-\alpha)r_u^{t-1}\boldsymbol{A}^\top \boldsymbol{D}^{-1}\boldsymbol{e}_u, \tag{8}$$

where we relax the assumption that entries of $\boldsymbol{r}^t$ could be negative. This violation enables us to move more magnitudes of residuals at once to x_u, thereby speeding up the entire procedure. Thus, based on the over-relaxed Equ. (7) and (8), we propose FwdPushSOR shown in Algo. 2, which is simple to implement and requires only the relaxation parameter ω. The key invariant

---

Algorithm 2 FwdPushSOR($\mathcal{G}, \epsilon, s, \alpha, \omega$)

1: Initialization: $\boldsymbol{r} = \boldsymbol{1}_s, \boldsymbol{x} = \boldsymbol{0}$
2: $Q = [s]$
3: **while** $Q \neq \emptyset$ **do**
4:     $u = Q.\text{pop}()$
5:     $x_u = x_u + \omega \cdot \alpha r_u$
6:     **for** $v \in \text{Nei}(u)$ **do**
7:         $r_v = r_v + \frac{\omega(1-\alpha)r_u}{d_u}$
8:         **if** $|r_v| \geq \epsilon d_v$ and $v \notin Q$ **then**
9:             $Q.\text{push}(v)$
10:     $r_u = (1-\omega)r_u$
11: **Return** $\boldsymbol{x}$

---

property of FwdPushSOR is that ωr_u magnitudes are *excessively removed* from r_u and distributed to all its outer neighbors and estimate x_u. Furthermore, this SOR-based method is still local. It is worth noting that this approach can generally be applied to other variants of FwdPush [89, 92]. For instance, the PwrPush algorithm proposed in [89] can easily be modified to incorporate the SOR technique. We call this method PwrPushSOR.

Parameter choosing for ω. First of all, FwdPushSOR exactly recovers FwdPush when $\omega = 1$. Note that $\boldsymbol{M}$ is symmetric and positive-definite for undirected graphs, and SOR would converge on any $\omega \in (0, 2)$ (see Thm. 4.4.12 of [45]). The optimal ω is

$$\omega = 1 + \left(\frac{1-\alpha}{1+\sqrt{1-(1-\alpha)^2}}\right)^2. \tag{9}$$

For directed graphs, choosing ω is more difficult since the matrix $\boldsymbol{M}$ is not easy to characterize. However, to choose ω, one can use the following heuristic way: ω starts from the optimal value; if it fails, we decrease ω by a constant step until it reaches 1. In practice, ω can reach about 1.5 when $\alpha = 0.15$, which is more than two times faster than existing FwdPush. Yet, the convergence of SOR-based methods remains an open problem for future research.

Unlike the standard G-S iteration, FwdPush updates a subset of nodes $\mathcal{S}_t$ of $\mathcal{V}$ at each epoch. In a recent work of Wu et al. [89], it has been proved that the queue-based implementation of FwdPush is similar to the power iteration method, hence for obtaining the final solution $\boldsymbol{x}^t$ with precision $\epsilon < (2m)^{-1}$, it requires $O(\frac{m}{\alpha} \log \frac{1}{\epsilon})$ operations. In the following, we improve the analysis and show that FwdPush is locally linear convergent to $\boldsymbol{x}^*$.

4.2 Local linear convergence of FwdPush

To establish the *locally linear convergence rate* and have a better illustration, we present slightly different FwdPush and add a *dummy node ‡* as presented in Algo. 3 where the dummy node helps to identify the super epochs. The parameter t is indexing the epoch id, and t' is indexing active node processing time, as shown in Fig. 1.
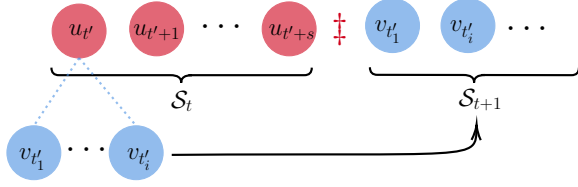
Figure 1: The t-th epoch of FwdPush in Algo. 3. FwdPush maintains a queue Q which contains all active nodes. At the beginning of t-th epoch, S_t contains all active nodes(red), which will be processed in t-th epoch. New active nodes (blue) generated in the current epoch will be processed in the next.

The only difference between Algo. 3 and Algo. 1 is that Algo. 3 always keeps ‡ in the queue until no active nodes pushed into Q. A new epoch begins whenever ‡ pops out and is pushed into Q again. In the next theorem, we prove that FwdPush admits a *local linear convergence* rate, and the total run time of FwdPush is only locally dependent on G. We have the following notations: At the beginning of t-th epoch, the set of active and inactive nodes are denoted as $S_t = \{u : r_u \geq \epsilon \cdot d_u, u \in \mathcal{V}\}$ and $\mathcal{U}_t = \{v : 0 < r_v < \epsilon \cdot d_v, v \in \mathcal{V}\}$, respectively. We also denote the support of r^t as $\mathcal{I}_t = \text{supp}(r^t)$. By these definitions, the total operations of FwdPush are the volume of all active nodes of all epochs, i.e., $\sum_{t=1}^{T} \text{vol}(S_t)$.

Algorithm 3 FwdPush($\mathcal{G}, \epsilon, \alpha, s$) with a dummy node

---
1: Initialization: $r = e_s, x = 0$
2: $Q = [s, ‡]$　　// Dummy node ‡ at the end of Q
3: $t = 0, t' = 0$
4: **while** $Q.\text{size}() \neq 1$ **do**
5:　　$u = Q.\text{pop}()$
6:　　**if** $u == ‡$ **then**
7:　　　　$Q.\text{push}(u)$
8:　　　　$t = t + 1$　　// Next epoch time
9:　　　　**continue**
10:　　$x_u = x_u + \alpha \cdot r_u$
11:　　**for** $v \in \text{Nei}(u)$ **do**
12:　　　　$r_v = r_v + \frac{(1-\alpha)r_u}{d_u}$
13:　　　　**if** $r_v \geq \epsilon d_v$ and $v \notin Q$ **then**
14:　　　　　　$Q.\text{push}(v)$
15:　　$r_u = 0$
16:　　$t' = t' + 1$
17: **Return** x

Theorem 2 (Local linear convergence of FwdPush). *Let S_t and $\mathcal{U}_t$ be the set of active and inactive nodes, respectively. Let x^t be the estimated PPR vector updated by FwdPush after t-th epoch, that is $x^t = \text{FwdPush}(\mathcal{G}, \epsilon, \alpha, s)$. Then, for all $t \geq 0$, the ℓ_1 estimation error $\|x^{t+1} - x^*\|_1$ has locally linear convergence rate, that is*

$$\|x^{t+1} - x^*\|_1 \leq (1 - \alpha\gamma_t)\|x^t - x^*\|_1, \qquad (10)$$

where γ_t is the local convergence factor $\gamma_t := \sum_{u \in S_t} d_u / \sum_{v \in \mathcal{I}_t} d_v$. Furthermore, the total run time of FwdPush is locally dependent on

$\mathcal{G}$, *is bounded by*

$$\sum_{t=1}^{T} \text{vol}(S_t) \leq \frac{\text{vol}(S_{1:T})}{\alpha \cdot \gamma_{1:T}} \log\left(\frac{C_{\alpha,T}}{\epsilon}\right), \qquad (11)$$

where $\text{vol}(S_{1:T})$ is the average volume, $\text{vol}(S_{1:T}) = \sum_{t=1}^{T} \text{vol}(S_t)/T$ and $\gamma_{1:T} = \sum_{t=1}^{T} \gamma_t/T$.

Before we prove the theorem, we introduce a key lemma, a new refinement from Wu et al. [89] as the following.

Lemma 3 (Locally linear decay of x^t). *Let S_t and $\mathcal{U}_t$ be the set of active and inactive nodes at the beginning of t-th epoch with $t \in \{0, 1, \ldots, t\}$, respectively. Then, after t-th epoch, we have $\|x - x^{t+1}\|_1 \leq (1 - \alpha\gamma_t)\|x - x^t\|_1$, where γ_t is the local convergence factor $\gamma_t := \sum_{u \in S_t} d_u / \sum_{v \in \mathcal{I}_t} d_v$.*

PROOF. We prove this lemma by showing that a significant residual has been pushed out from r^t to r^{t+1} and corresponding gain from x^t into x^{t+1}. The set of active nodes S_t has been processed: we use time t' to index these nodes. The updates are the following. For each i-th active node $u_{t'_i}$, the updates are from Line 10 to Line 15 of Algo. 3 give us the following iterations

$$x_t = x_{u_{t'_0}} \xrightarrow{u_{t'_1}} x_{t'_1} \xrightarrow{u_{t'_2}} x_{t'_2} \cdots \xrightarrow{u_{t'_{|S_t|}}} x_{t'_{|S_t|}} = x_{t+1}$$

$$r_t = r_{u_{t'_0}} \xrightarrow{u_{t'_1}} r_{u_{t'_1}} \xrightarrow{u_{t'_2}} r_{u_{t'_2}} \cdots \xrightarrow{u_{t'_{|S_t|}}} r_{u'_{|S_t|}} = r_{t+1}.$$

For t-th epoch, the total amount of residual that had been pushed out is $\alpha \sum_{u_{t'} \in S_t} r_{u_{t'}}$ (Line 10). That is,

$$\|r_t\|_1 - \|r_{t+1}\|_1 \geq \alpha \sum_{u_{t'} \in S_t} r_{u_{t'}}. \qquad (12)$$

By the definition of S_t and $\mathcal{U}_t$, we have

$$\forall u_{t'} \in S_t, r_{u_{t'}} \geq \epsilon \cdot d_{u_{t'}}, \qquad \forall v \in \mathcal{U}_t, 0 < r_v < \epsilon \cdot d_v.$$

Summation above inequalities over all active nodes $u_{t'}$ and inactive nodes v, we have

$$\frac{\sum_{u_{t'} \in S_t} r_{u_{t'}}}{\sum_{u_{t'} \in S_t} d_{u_{t'}}} \geq \epsilon > \frac{\sum_{v \in \mathcal{U}_t} r_v}{\sum_{v \in \mathcal{U}_t} d_v},$$

which indicates

$$\frac{\sum_{u_{t'} \in S_t} r_{u_{t'}}}{\sum_{u_{t'} \in S_t} d_{u_{t'}}} > \frac{\sum_{u_{t'} \in S_t} r_{u_{t'}} + \sum_{v \in \mathcal{U}_t} r_v}{\sum_{u_{t'} \in S_t} d_{u_{t'}} + \sum_{v \in \mathcal{U}_t} d_v}$$

$$= \frac{\sum_{v \in \mathcal{I}_t} r_v}{\sum_{v \in \mathcal{I}_t} d_v} = \frac{\|r_t\|_1}{\sum_{v \in \mathcal{I}_t} d_v}, \qquad (13)$$

where the last equality is due to the fact that $\mathcal{I}_t$ indexes all nonzero entries of r_t, i.e., $\|r_t\|_1 = \sum_{v \in \mathcal{I}_t} r_t(v)$. On the other hand, the t-th iteration error of FwdPush as $\|x^t - x^*\|_1$. Clearly, when $t = 0$, $\|x^t - x^*\|_1 = 1$. For $t \geq 0$, we have

$$\|x^* - x^{t+1}\|_1 = \|x^* - x^t\|_1 - \alpha \sum_{u_{t'} \in S_t} r^t_{u'_t}$$

$$= \left(1 - \frac{\alpha \sum_{u_{t'} \in S_t} r^t_{u'_t}}{\|r^t\|_1}\right)\|x^* - x^t\|_1$$

$$\leq \left(1 - \frac{\alpha \sum_{u \in S_t} d_u}{\sum_{v \in \mathcal{I}_t} d_v}\right)\|x^* - x^t\|_1,$$

where the last inequality follows from (13). □

Our new local convergence factor is $(1 - \alpha\eta_t)$ which is strictly great than $1 - \alpha \sum_{u \in S_t} d_u/m$ from Wu et al. [89], meaning better convergence rate. The other key ingredient of our theorem is to estimate the total number of epochs needed. The observation is that the total amount of residuals left in r^t is still *relatively significant*, so the residual in the last epoch $\|r^T\|_1$ is lower bounded. We state the upper bound of T as the following.

Lemma 4. *Let T be the total epochs used in FwDPush$(\mathcal{G}, \epsilon, \alpha, s)$, then it can be bounded by*

$$T \leq \frac{1}{\alpha \cdot \gamma_{1:T}} \log\left(\frac{C_{\alpha,T}}{\epsilon}\right), \tag{14}$$

where $C_{\alpha,T} = 1/((1-\alpha)|\mathcal{I}_T|)$ and $\gamma_{1:T} = \sum_{t=1}^{T} \gamma_t/T$.

PROOF. After the last epoch T, for each of nonzero node v, there was an active neighbor of v, denote as u, which pushed some residual $\frac{(1-\alpha)r_u}{d_u}$ to v. We denote each of this amount residual as $\tilde{r}_v$, then for all $v \in \mathcal{I}_T$, we have

$$\|r_T\|_1 = \sum_{v \in \mathcal{I}_T} r_v \geq \sum_{v \in \mathcal{I}_T} \tilde{r}_v := \sum_{v \in \mathcal{I}_T} \frac{(1-\alpha)r_u}{d_u}$$

$$\geq \sum_{v \in \mathcal{I}_T} \frac{(1-\alpha)\epsilon d_u}{d_u} = \sum_{v \in \mathcal{I}_T} (1-\alpha)\epsilon = (1-\alpha)\epsilon|\mathcal{I}_T|.$$

From (10) of Lemma 3, the upper bound of $\|r_T\|_1$ is

$$\|r_T\|_1 \leq \prod_{t=1}^{T} \left(1 - \frac{\alpha \sum_{u \in S_t} d_u}{\sum_{v \in \mathcal{I}_T} d_v}\right)\|x^* - x^0\|_1 = \prod_{t=1}^{T}(1 - \alpha\gamma_t).$$

Combine the lower and upper bound, we obtain

$$(1-\alpha)\epsilon|\mathcal{I}_T| \leq \prod_{t=1}^{T}(1 - \alpha\gamma_t).$$

Take log on both sides of the above and use the fact $\log(1 - \alpha\gamma_t) < -\alpha\gamma_t$, we reach

$$T \leq \frac{1}{\alpha \cdot \gamma_{1:T}} \log\left(\frac{C_{\alpha,T}}{\epsilon}\right),$$

where $C_{\alpha,T} = 1/((1-\alpha)|\mathcal{I}_T|)$ and $\gamma_{1:T} = \sum_{t=1}^{T} \gamma_t/T$. □

PROOF OF THEOREM 2. To obtain our main theory, we apply Lemma 3 and Lemma 4, and notice that

$$\sum_{t=1}^{T} \text{vol}(S_t) = T \cdot \text{vol}(S_{1:T}) \leq \frac{\text{vol}(S_{1:T})}{\alpha \cdot \gamma_{1:T}} \log\left(\frac{C_{\alpha,T}}{\epsilon}\right).$$

□

Remark 5. *Our locality analysis provides intuition on the performance of local FwDPush. The total convergence rate is determined by γ_t, which is always a positive number. Note that $\gamma_0 = 1$ for the first epoch. The average volume accessed by FwDPush is certainly controlled by the following*

$$\frac{\text{vol}(\text{supp}(x_{\epsilon,\alpha}))}{T} \leq \text{vol}(S_{1:T}) \leq \text{vol}(\text{supp}(x_{\epsilon,\alpha})), \tag{15}$$

where we simply denote x_T as $x_{\epsilon,\alpha}$ which only depends on α and ϵ when s and $\mathcal{G}$ are fixed. The above inequality indicates that when the
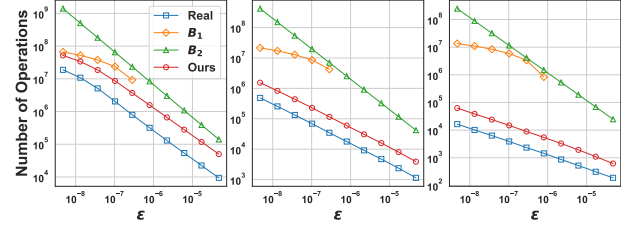

Figure 2: The number of operations estimated for the dblp dataset as a function of ϵ. *Real* stands for actual number of operations used in FwDPush, i.e. *Real*$= \sum_{t=1}^{T} \text{vol}(S_t)$. $B_1 = \frac{m}{\alpha} \log(\frac{1}{\epsilon m}) + m$ provided in [89], $B_2 = \frac{1}{\epsilon\alpha}$, and our new local bound. Left: $\alpha = 0.15$, Middle: $\alpha = 0.5$, and Right: $\alpha = 0.85$. We randomly selected 100 nodes for each experiment and took the average of operations estimated.

solution is sparse, the volume will be much less than m. Another observation is that when $\epsilon \to 0$, we have $\gamma_{1:T} \to 1$, and $\text{vol}(S_{1:T}) \to m$. Hence, it will recover to the power iteration-like method studied in [89]. To see how the bound estimated the total operations, we conduct experiments on the dblp graph applying FwDPush over different ϵ as illustrated in Fig. 2. Compared to two known bounds, our parameterized local bound is tighter. We find a similar pattern on other graph datasets, detailed explained in the appendix.

Heuristic method FwDPush-Mean. Our parameterized bound in Equ. (11) involves a flexible quantity $\eta_T = \frac{\text{vol}(S_{1:T})}{\gamma_{1:T}}$, which can guide us in finding better methods. For example, we can find a method that tries to minimize η_T. By noticing that η_T is a trade-off between the ratio of active nodes processed in each epoch, we can either postpone pushing nodes with a large degree or the magnitudes r_u are small. Doing this can save some operations and push more residuals for the next epoch.

To make this idea concrete, at the beginning of each epoch, we push a subset of active nodes with a large magnitude ratio at each epoch, that is, to measure the *magnitude factor* r_u/d_u. To do this, we check these nodes for each epoch by seeing whether the current magnitude factor of node u is bigger than the mean of these ratios (which is an easy quantity to measure at the beginning of each super epoch). We postpone $u \in S^t$ to the next epoch if its current magnitude factor satisfies

$$\frac{r_u}{d_u} < \bar{r} \triangleq \frac{1}{|S_t|} \sum_{v \in S^t} \frac{r_v}{d_v},$$

which means it is not worth pushing it; we can save u to the next epoch so that u accumulates more residual from u's neighbors. We implemented this idea, called it FwDPush-Mean, and presented it in Algo. 4 of the appendix. Interestingly, FwDPush-Mean can effectively reduce the number of operations of FwDPush. Fig. 3 illustrates the number of operations reduced by the proposed FwDPush-Mean (see details of Algo. 4 in the appendix). Although run time may not be significantly reduced (observed in our experiments), this is still valuable in resource-limited scenarios as a significant amount of operations are reduced. This method could also aid in finding a better heuristic for improving FwDPush. A detailed description of Algo. 4 can be found in the appendix.
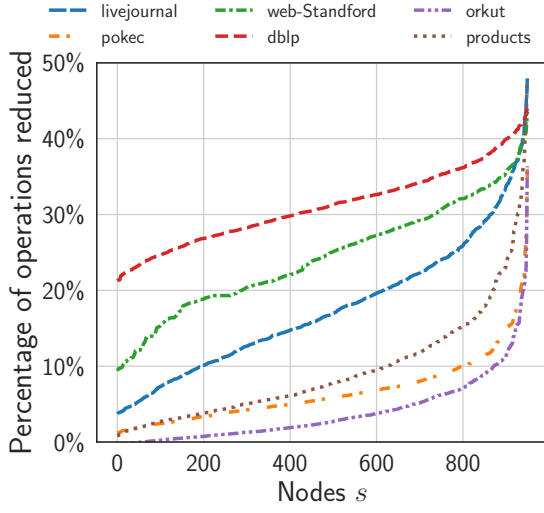
Figure 3: The percentage of total operations reduced by FwdPush-Mean as a function of nodes over all six graphs. We fix $\epsilon = 10^{-6}$, $\alpha = 0.2$ and run both FwdPush-Mean and FwdPush on 1,000 randomly selected nodes from six graphs. The reduced percentage of operations is defined as the difference in the number of operations between two methods divided by the operations of FwdPush.

Our local linear convergence guarantee for FwdPush allows for improvement through a trade-off between the number of active nodes explored in each epoch $\mathcal{S}^t$ and the total number of epochs $O(\frac{1}{\alpha} \log \frac{C_{\alpha,T}}{\epsilon})$. If $\gamma_{1:T}$ is large, the total number of epochs is expected to be small, and $\log \frac{C_{\alpha,T}}{\epsilon}$ will also have a minimal effect. However, the total number of epochs will greatly depend on α. If α is small, FwdPush becomes slow. Using SOR can speed up FwdPush. In the next section, we demonstrate that we can further save $1/\sqrt{\alpha}$ run time by employing global acceleration-based methods if the underlying graph is undirected.

5 MOMENTUM-BASED METHODS FOR PPVS

This section analyzes the calculation of PPV when $\mathcal{G}$ is undirected. It reformulates the computation of PPV as a convex optimization problem and employs an acceleration-based technique to solve the linear system.

5.1 Quadratic optimization lens

We can rewrite the linear system of (1) into strongly convex optimization and then transform the problem into a strongly convex optimization problem. Recall our target linear system is $(I - (1 - \alpha)AD^{-1})x = \alpha e_s$. For an undirected graph, we multiply both sides by $D^{-1/2}$ and rewrite the system such that the right-hand side matrix is symmetric; that is (1) can be reformulated as the following

$$\left(I - (1 - \alpha)D^{-1/2}AD^{-1/2}\right)D^{-1/2}x = \alpha D^{-1/2}e_s.$$

Denote $\tilde{P} := D^{-1/2}AD^{-1/2}$, $y := D^{-1/2}x$, and $\tilde{s} := D^{-1/2}e_s$. Notice that $I - (1 - \alpha)\tilde{P}$ is symmetric normalized Laplacian parameterized by $(1 - \alpha)$. That is, we shall solve the linear system, $(I - (1 - \alpha)\tilde{P})y = $

$\alpha\tilde{s}$. We define the minimization problem of a quadratic objective function f as the following

$$\arg\min_{y \in \mathbb{R}^n} \left\{ f(y) \triangleq \frac{1}{2}y^\top \left(I - (1 - \alpha)\tilde{P}\right)y - \alpha\tilde{s}^\top y \right\}, \quad (16)$$

where $\tilde{P}$ is not a row stochastic matrix any more but $I - (1 - \alpha)\tilde{P}$ is still positive definite. Clearly, f is a strongly convex function, by taking the gradient $\nabla f(y)$ and letting it be zero, i.e. $\nabla f(y) := (I - (1 - \alpha)\tilde{P})y - \alpha\tilde{s} = 0$, we see that f has a unique solution $y^* = D^{-1/2}x^*$. Therefore, the original solution x^* can be recovered from $D^{1/2}y^*$. To characterize the strongly convex and strong smoothness parameters of f (See definitions of strongly convex and smoothness in Appendix B), denote $\lambda_1, \lambda_2, \ldots, \lambda_n$ as eigenvalues of P with $\lambda_1 = 1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n \geq -1$ and let $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_n$ be eigenvalues of $\tilde{P}$. Therefore, by the fact from the graph spectral theory [28], given the normalized $I - D^{-1/2}AD^{-1/2}$, we have its eigenvalues $0 = \tilde{\lambda}_1 \leq \cdots \leq \tilde{\lambda}_n \leq 2$. Therefore, the range of eigenvalues of $I - (1 - \alpha)\tilde{P}$ is $\lambda(I - (1 - \alpha)\tilde{P}) \in [\alpha, 2 - \alpha]$.

The above reformulation is commonly used in the optimization community and has also been studied in Fountoulakis et al. [33]. It was observed that when the graph is undirected, FwdPush is a special case of a coordinate descent. The coordinate descent for the above problem (16) is

$$y^{t+1} = y^t - \eta_t \nabla_u f(y^t) \cdot e_u, \quad (17)$$

where each step size should be chosen such that $\eta_t \leq 1/L_u$, where L_u is the Lipschitz continuous parameter, $|\nabla_u f(y + \delta e_u) - \nabla_u f(y)| \leq L_u \cdot \delta$, for all $y \in \mathbb{R}^n$. Clearly, L_u corresponds to the diagonal of $I - (1 - \alpha)D^{-1/2}AD^{-1/2}$, which we always have $L_u \leq 1$. Replacing $\eta_t = 1$ and letting $-\nabla(y^t) = r^t$ in (17). We can recover FwdPush accordingly. The coordinate descent algorithm could be accelerated by choosing momentum strategy [3, 62].

The challenge of obtaining faster iteration based on the accelerated coordinate descent method remains open due to the lack of linear-invariant property for the momentum method [34]. Nevertheless, we use standard momentum techniques and leverage the advantage of continuous memory. Our momentum-based method can be expressed in a single line, and local linear convergence suggests that the number of iterations is $O\left(\frac{1}{\alpha \cdot \gamma_{1:T}} \log \frac{C_{\alpha,T}}{\epsilon}\right)$. In the next section, we will demonstrate that one can save $1/\sqrt{\alpha}$ run time.

5.2 Accelerated methods

Nesterov's Accelerated Gradient (NAG) Method. Given a strongly convex function f, NAG method [71], at t-th iteration, updates two vectors y^t and z^t. It updates y^t with the help of z^t (with $z^0 = y^0$) as the following

$$z^t = y^{t-1} - \eta_t \nabla f(y^{t-1})$$
$$y^t = z^t + \beta_t(z^t - z^{t-1}).$$

If we set $\eta_t = 1/(2 - \alpha)$ and $\beta_t = (1 - \kappa)/(1 + \kappa)$ with the inverse square root of condition number $\kappa = \sqrt{(2 - \alpha)/\alpha}$, then we can write the iteration in one line as the following iteration procedure

$$\text{NAG:} \quad y^{t+1} = 2\left[\eta(I + \tilde{P})y^t\right] - (1 - \kappa)\left[\eta(I + \tilde{P})y^{t-1}\right] + \kappa^2\tilde{s}, \quad (18)$$

where $\eta = (1 - \alpha)/((2 - \alpha)(1 + \kappa))$ and $\boldsymbol{y}_0 = \boldsymbol{0}, \boldsymbol{y}_1 = \alpha D^{-1/2}\boldsymbol{e}_s$. Notice that $(\boldsymbol{I} + \tilde{\boldsymbol{P}})\boldsymbol{y}^t$ can be used for the next iteration; hence the per-iteration operations are at most m. Then we have the following convergence rate for ℓ_1 estimation error of $\boldsymbol{x}^*$.

Theorem 6. *Let $\boldsymbol{y}^{t+1}$ be the estimated vector returned by NAG method using iteration* (18) *(with $\boldsymbol{z}^0 = \boldsymbol{y}^0 = \boldsymbol{0}$) and let $\boldsymbol{x}^t = D^{1/2}\boldsymbol{y}^t$, then the estimation error of $\boldsymbol{x}^*$ is upper bounded by*

$$\|\boldsymbol{x}^t - \boldsymbol{x}^*\|_1 \le d_{\max}\sqrt{\frac{2n}{\alpha}}\exp\left(-\frac{t-1}{2\sqrt{(2-\alpha)/\alpha}}\right). \qquad (19)$$

And the total number of operations required for ϵ-precision of per-entry of $\boldsymbol{x}^t - \boldsymbol{x}^*$, i.e., $\|\boldsymbol{x}^t - \boldsymbol{x}^*\|_\infty \le \epsilon$ is

$$R_T = O\left(\frac{m}{\sqrt{\alpha}}\log\left(\frac{d_{\max}}{\epsilon}\sqrt{\frac{2n}{\alpha}}\right)\right). \qquad (20)$$

PROOF. The proof can be found at Appendix C. □

Remark 7. *When high precision is required, and α is small, FWDPUSH is likely not a local method, meaning the per-epoch needs to touch the whole graph, hence the total complexity is about $O(\frac{m}{\alpha}\log\frac{1}{\epsilon})$. Compared with the bound provided in Thm. 8, NAG method is $1/\sqrt{\alpha}$ times faster. However, whether there exists a local acceleration-based method like bound $O\left(\frac{\mathrm{vol}(S)}{\sqrt{\alpha}}\log\frac{1}{\epsilon}\right)$ is still an open problem* [34].

Polyak's Heavy Ball (HB) method. Similar to the NAG method, the other popular momentum-based acceleration method is the Heavy Ball method [75]; different from the NAG method, the HB method updates $\boldsymbol{y}^t$ as the following

$$\boldsymbol{y}^{t+1} = \boldsymbol{y}^t - \eta_t\nabla f(\boldsymbol{y}^t) + \beta_t(\boldsymbol{y}^t - \boldsymbol{y}^{t-1}),$$

where we set $\eta_t = 4/(\sqrt{2-\alpha} + \sqrt{\alpha})^2$ and $\beta_t = ((1-\kappa)/(1+\kappa))^2$, we can reach the following update method:

$$\boldsymbol{y}^{t+1} = 2(1-\alpha)\frac{1+\kappa^2}{(1+\kappa)^2}\tilde{\boldsymbol{P}}\boldsymbol{y}^t - \frac{(1-\kappa)^2}{(1+\kappa)^2}\boldsymbol{y}^{t-1} + \frac{2\alpha(1+\kappa^2)}{(1+\kappa)^2}\tilde{\boldsymbol{s}}.$$

By changing variable $D^{1/2}\boldsymbol{y}^{t+1} = \boldsymbol{x}^{t+1}$, we have HB

$$\boldsymbol{x}^{t+1} = \frac{2\alpha(1+\kappa^2)}{(1+\kappa)^2}AD^{-1}\boldsymbol{x}^t - \frac{(1-\kappa)^2}{(1+\kappa)^2}\boldsymbol{x}^{t-1} + \frac{2(1-\alpha)(1+\kappa^2)}{(1+\kappa)^2}\boldsymbol{v}.$$

Compared with local linear methods such as CD and FWDPUSH, the NAG and HB method admits a better convergence rate where the total number of iterations is about $O(m/\sqrt{\alpha})$ compared with linear ones $O(m/\alpha)$. Hence $O(1/\sqrt{\alpha})$ times faster than methods presented in Sec. 4. One can also consider the most popular method, the conjugate gradient method [79], which also has a better convergence rate than the standard gradient descent method and is the same as our momentum-based methods. However, our two methods are easier to implement.

6 EXPERIMENTS

We conduct experiments on 6 real-world benchmark graphs to evaluate our proposed PPV algorithms, including HB, NAG, and FWDPUSHSOR, and PWRPUSHSOR, which is the combination of PWRPUSH [89] and SOR. In the experiments, we aim to answer the following question: How fast are these proposed methods compared

with baselines in terms of run time and the number of operations needed for different settings of α? The results demonstrate the supreme efficiency boosted by SOR, achieving 2-3 times faster than strong baselines when $\alpha = 0.15$.

Datasets. We use both directed graphs (*dblp* [90], *products* [51] and *orkut* [90]) as well as undirected graphs (*web-Stanford* [54], *pokec* [83], *livejournal* [11]). These are the most common datasets for benchmark PPR algorithms. We remove nodes with no in-degrees or out-degrees (dangling nodes), relabel the rest nodes, and use two directed edges to denote an undirected edge. Table 1 presents the detailed statistics.

Table 1: Datasets statistics

Dataset	n	m	Type of $\mathcal{G}$
dblp	317,080	1,049,866	undirected
products	2,449,029	123,718,280	undirected
orkut	3,072,441	117,185,083	undirected
web-Stanford	281,903	2,312,497	directed
livejournal	4,847,571	68,993,773	directed
pokec	1,632,803	30,622,564	directed

Baselines. We compare our proposed algorithms with three state-of-the-art baselines: FWDPUSH [6], POWITR, and PWRPUSH [89], a variant of FWDPUSH. We use the proposed SOR technique to implement both FWDPUSHSOR and PWRPUSHSOR. For choosing the relaxation parameter ω: 1) for undirected graphs, we directly use optimal value in (9); 2) for directed graphs, we take an adaptive strategy described in Sec. 4 to search ω where we set the minimal $\omega = 1$ with step size 0.1 to the maximal value defined in Sec. 4. We only record the time consumed by the best ω.

Experiment Settings. For each graph, we uniformly sample 50 nodes for PPV calculation with $\epsilon = \min\left(1/10^8, 1/m\right)$ (same as [89]) and repeat experiments 5 times, recording the average running time, the number of residual updates, and the corresponding ℓ_1-error. Note that the number of residual updates reflects the theoretical complexity regardless of the overheads incurred by various data structures. We measure PPV precision using ℓ_1 error $\|\boldsymbol{x}^t - \boldsymbol{x}^*\|_1$ which can be measured by $\|\boldsymbol{r}^t\|_1$.

Infrastructure and Implementation. All experiments were conducted on a machine equipped with an Intel Xeon Gold 5218R CPU @ 2.10GHz (80 cores) with 256GB Memory. All algorithms are implemented in Python with the Numba library.[4]

6.1 Experimental results

SOR-based methods are faster and need much less number of operations on both undirected and directed graphs. A-F of Fig. 4 and 5 present the run time and the number of operations of PPV methods when $\alpha = 0.15$, respectively. First, SOR-based methods are more than 2 times faster than their counterparts on both undirected and directed graphs shown in Fig. 4 (A-F). This confirms our SOR-based methods effectively speed up their counterparts. Note that by using a continuous memory access strategy, PWRPUSH and PWRPUSHSOR is, in general, faster than FWDPUSH and FWD-PUSHSOR even if the number of operations for both is similar as shown in Fig. 5 (A-F). Indeed, our SOR-based methods save half of

---

[4]https://numba.pydata.org/
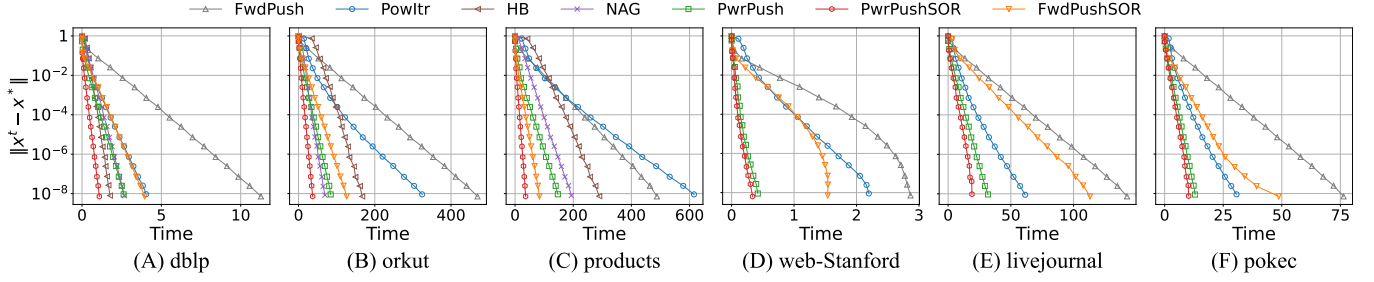
Figure 4: Estimation error v.s. run time (seconds), $\alpha = 0.15$.
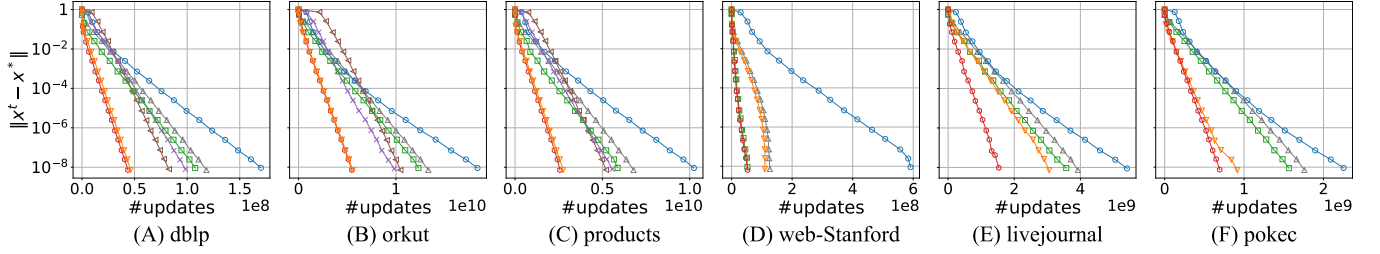
Figure 5: Estimation error v.s. #residue updates (total operations), $\alpha = 0.15$

the total operations. As shown in the appendix, we observed more significant speedup results when $\alpha = 0.05$. The speedup advantages still exist even when α is large, i.e., $\alpha = 0.2$ and 0.25.

Acceleration-based methods are faster and need fewer operations for undirected graphs. (A, B, C) of Fig. 4 and 5 present results on undirected graphs. Among all methods, HB and NAG are faster than these non-acceleration methods but also use fewer operations. When $\alpha = 0.05$ is a smaller value, the gap is more significant, as seen in Fig. 10 and 11. These results verify the $O(1/\sqrt{\alpha})$ times faster predicted by our theorem. However, compared with these local methods, the speedup of acceleration-based methods is insignificant. One may expect a local version of acceleration-based methods could further improve HB and NAG. PwrIter as a global method uses more operations than FwdPush but requires less run time, as shown in the results of directed graphs. This is, again, because PwrIter uses a continuous memory access strategy while the nodes in the queue of FwdPush are randomly ordered, thus slowing down the process.

Local linear convergence rate of FwdPush. To empirically answer Q1 asked in Sec. 1, we show that FwdPush has local linear convergence rate even when $\epsilon > (2m)^{-1}$. To do this, we simply set $\epsilon = 1/m$, randomly pick a node from three directed graphs, and then run FwdPush. The convergence rates are illustrated in Fig. 6. These linear decay rates of estimation error are consistent with Thm. 2. We found similar patterns on undirected graphs.

7 DISCUSSION

This paper examines the calculation of PPV for directed and undirected graphs. We show that the commonly used local method for undirected graphs, FwdPush, is a variation of Gauss-Seidel. To improve the efficiency of FwdPush, we propose to use the SOR
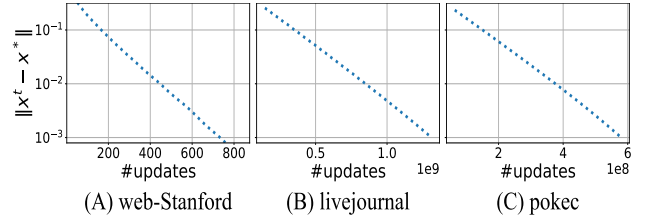
Figure 6: Locally linear convergence of FwdPush. For each directed graph, we randomly pick up a node and run FwdPush with $\epsilon = 1/m$.

technique. Our SOR-based methods can successfully speed up current local methods significantly. Our SOR-based and acceleration methods could help build large-scale graph neural networks. It is worth seeing whether the SOR technique can be applied to the dynamic graph computation of PPVs. Additionally, we demonstrate that momentum-based acceleration methods can be used to obtain the PPV calculation for undirected graphs, providing $O(1/\sqrt{\alpha})$ acceleration. Both acceleration methods are easy to implement and perform faster than other local methods when α is small. As a future work, it is interesting to see if it is possible to reduce $O(m)$ from the bound in Thm. 8 to a local quantity.

8 ACKNOWLEDGEMENT

REFERENCES

[1] Alekh Agarwal and Leon Bottou. 2015. A lower bound for the optimization of finite sums. In *International conference on machine learning*. PMLR, 78–86.

[2] Zeyuan Allen-Zhu and Lorenzo Orecchia. 2014. Linear coupling: An ultimate unification of gradient and mirror descent. *arXiv preprint arXiv:1407.1537* (2014).

[3] Zeyuan Allen-Zhu, Zheng Qu, Peter Richtárik, and Yang Yuan. 2016. Even faster accelerated coordinate descent using non-uniform sampling. In *International Conference on Machine Learning*. PMLR, 1110–1119.

[4] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng. 2007. Local computation of PageRank contributions. In *WAW*, Vol. 4863. Springer, 150–165.

[5] 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.

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

[7] William N Anderson Jr and Thomas D Morley. 1985. Eigenvalues of the Laplacian of a graph. *Linear and multilinear algebra* 18, 2 (1985), 141–145.

[8] Anton Anikin, Alexander Gasnikov, Alexander Gornov, Dmitry Kamzolov, Yury Maximov, and Yurii Nesterov. 2020. Efficient numerical methods to solve sparse linear equations with application to PageRank. *Optimization Methods and Software* (2020), 1–29.

[9] Arvind Arasu, Jasmine Novak, Andrew Tomkins, and John Tomlin. 2002. PageRank computation and the structure of the web: Experiments and algorithms. In *Proceedings of the eleventh international World Wide Web conference, poster track*. 107–117.

[10] Yossi Arjevani, Shai Shalev-Shwartz, and Ohad Shamir. 2016. On Lower and Upper Bounds in Smooth and Strongly Convex Optimization. *JMLR* 17, 1 (2016), 4303–4353.

[11] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. 2006. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 44–54.

[12] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast Incremental and Personalized PageRank. *Proceedings of the VLDB Endowment* 4, 3 (2010).

[13] Amir Beck and Marc Teboulle. 2009. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM Journal on Imaging Sciences* 2, 1 (2009), 183–202.

[14] Andras A Benczur, Karoly Csalogany, Tamas Sarlos, and Mate Uher. 2005. Spamrank–fully automatic link spam detection work in progress. In *Proceedings of the first international workshop on adversarial information retrieval on the web*. 1–14.

[15] Pavel Berkhin. 2006. Bookmark-coloring algorithm for personalized PageRank computing. *Internet Mathematics* 3, 1 (2006), 41–62.

[16] Abraham Berman and Robert J Plemmons. 1994. *Nonnegative matrices in the mathematical sciences*. SIAM.

[17] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Martin Blais, Amol Kapoor, Michal Lukasik, and Stephan Günnemann. 2019. Is PageRank all you need for scalable graph neural networks?. In *ACM KDD, MLG Workshop*.

[18] 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 & Data Mining*. 2464–2473.

[19] Paolo Boldi, Roberto Posenato, Massimo Santini, and Sebastiano Vigna. 2006. Traps and pitfalls of topic-biased PageRank. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 107–116.

[20] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. 2005. PageRank as a function of the damping factor. In *Proceedings of the 14th international conference on World Wide Web*. 557–566.

[21] Stephen Boyd. 2006. Convex optimization of graph Laplacian eigenvalues. In *Proceedings of the International Congress of Mathematicians*, Vol. 3. 1311–1319.

[22] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems* 30, 1-7 (1998), 107–117.

[23] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. 2011. Graph structure in the web. In *The Structure and Dynamics of Networks*. Princeton University Press, 183–194.

[24] Sébastien Bubeck et al. 2015. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning* 8, 3-4 (2015), 231–357.

[25] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and deep graph convolutional networks. In *International conference on machine learning*. PMLR, 1725–1735.

[26] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 257–266.

[27] Edmond Chow, Andreas Frommer, and Daniel B Szyld. 2021. Asynchronous Richardson iterations: theory and practice. *Numerical Algorithms* 87, 4 (2021), 1635–1651.

[28] Fan RK Chung and Fan Chung Graham. 1997. *Spectral graph theory*. Number 92. American Mathematical Soc.

[29] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM review* 51, 4 (2009), 661–703.

[30] Anna Concas, Lothar Reichel, Giuseppe Rodriguez, and Yunzi Zhang. 2021. Iterative Methods for the Computation of the Perron Vector of Adjacency Matrices. *Mathematics* 9, 13 (2021), 1522.

[31] Alessandro Epasto, Vahab Mirrokni, Bryan Perozzi, Anton Tsitsulin, and Peilin Zhong. 2022. Differentially Private Graph Learning via Sensitivity-Bounded Personalized PageRank. In *NeurIPS 2022 Workshop: New Frontiers in Graph Learning*. https://openreview.net/forum?id=dzVZGSe0NoJ

[32] 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.

[33] Kimon Fountoulakis, Farbod Roosta-Khorasani, Julian Shun, Xiang Cheng, and Michael W Mahoney. 2019. Variational perspective on local graph clustering. *Mathematical Programming* 174, 1 (2019), 553–573.

[34] Kimon Fountoulakis and Shenghao Yang. 2022. Open Problem: Running time complexity of accelerated ℓ_1-regularized PageRank. In *Conference on Learning Theory*. PMLR, 5630–5632.

[35] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations*.

[36] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. *Advances in neural information processing systems* 32 (2019).

[37] Claudio Gentile. 2003. The robustness of the p-norm algorithms. *Machine Learning* 53, 3 (2003), 265–299.

[38] David F Gleich. 2015. PageRank Beyond the Web. *SIAM Rev.* 57, 3 (2015), 321–363.

[39] David F Gleich, Andrew P Gray, Chen Greif, and Tracy Lau. 2010. An inner-outer iteration for computing PageRank. *SIAM Journal on Scientific Computing* 32, 1 (2010), 349–371.

[40] David F Gleich, Lek-Heng Lim, and Yongyang Yu. 2015. Multilinear PageRank. *SIAM J. Matrix Anal. Appl.* 36, 4 (2015), 1507–1541.

[41] Gene H Golub and Charles F Van Loan. 2013. *Matrix computations*. JHU press.

[42] Anne Greenbaum. 1997. *Iterative methods for solving linear systems*. SIAM.

[43] Vince Grolmusz. 2015. A note on the PageRank of undirected graphs. *Inform. Process. Lett.* 115, 6-8 (2015), 633–634.

[44] Xingzhi Guo, Baojian Zhou, and Steven Skiena. 2021. Subset Node Representation Learning over Large Dynamic Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 516–526.

[45] Wolfgang Hackbusch. 1994. *Iterative solution of large sparse systems of equations*. Vol. 95. Springer.

[46] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*. PMLR, 4116–4126.

[47] Taher Haveliwala and Sepandar Kamvar. 2003. *The second eigenvalue of the Google matrix*. Technical Report. Stanford.

[48] Taher H Haveliwala. 2003. Topic-sensitive PageRank: A context-sensitive ranking algorithm for web search. *IEEE transactions on knowledge and data engineering* 15, 4 (2003), 784–796.

[49] Magnus Rudolph Hestenes, Eduard Stiefel, et al. 1952. *Methods of conjugate gradients for solving linear systems*. Vol. 49. NBS Washington, DC.

[50] Sungpack Hong, Nicole C Rodia, and Kunle Olukotun. 2013. On fast parallel detection of strongly connected components (SCC) in small-world graphs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–11.

[51] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

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

[53] Jürgen Jost, Raffaella Mulas, and Florentin Münch. 2021. Spectral gap of the largest eigenvalue of the normalized graph Laplacian. *Communications in Mathematics and Statistics* (2021), 1–11.

[54] Jinhong Jung, Namyong Park, Sael Lee, and U Kang. 2017. Bepi: Fast and memory-efficient method for billion-scale random walk with restart. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 789–804.

[55] Sepandar D Kamvar, Taher H Haveliwala, Christopher D Manning, and Gene H Golub. 2003. Extrapolation methods for accelerating PageRank computations. In *Proceedings of the 12th international conference on World Wide Web*. 261–270.

[56] Sai Praneeth Karimireddy, Anastasia Koloskova, Sebastian U Stich, and Martin Jaggi. 2019. Efficient greedy coordinate descent for composite problems. In *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2887–2896.

[57] Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. 2013. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 911–920.

[58] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations (ICLR)*.

[59] Johannes Klicpera, Chandan Yeshwanth, and Stephan Günnemann. 2021. Directional Message Passing on Molecular Graphs via Synthetic Coordinates. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

[60] Amy N Langville and Carl D Meyer. 2011. *Google's PageRank and beyond*. Princeton university press.

[61] Chris Pan-Chi Lee, Gene H Golub, and Stefanos A Zenios. 2003. *A fast two-stage algorithm for computing PageRank and its extensions*. Technical Report. Citeseer.

[62] Yin Tat Lee and Aaron Sidford. 2013. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *2013 ieee 54th annual symposium on foundations of computer science*. IEEE, 147–156.

[63] Peter Lofgren. 2015. *Efficient algorithms for personalized PageRank*. Stanford University.

[64] 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.

[65] Haihao Lu, Robert Freund, and Vahab Mirrokni. 2018. Accelerating greedy coordinate descent methods. In *International Conference on Machine Learning*. PMLR, 3257–3266.

[66] DORIN Marghidanu. 2008. Generalizations and refinements for Bergstrom and Radon's inequalities. *Journal of Science and Arts* 8, 1 (2008), 57–62.

[67] Frank McSherry. 2005. A uniform approach to accelerated PageRank computation. In *Proceedings of the 14th international conference on World Wide Web*. 575–582.

[68] Robert Meusel, Sebastiano Vigna, Oliver Lehmberg, and Christian Bizer. 2014. Graph structure in the web—revisited: a trick of the heavy tail. In *Proceedings of the 23rd international conference on World Wide Web*. 427–432.

[69] Bojan Mohar. 1991. Eigenvalues, diameter, and mean distance in graphs. *Graphs and combinatorics* 7, 1 (1991), 53–64.

[70] Deanna Needell, Rachel Ward, and Nati Srebro. 2014. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. *Advances in neural information processing systems* 27 (2014), 1017–1025.

[71] Yurii Nesterov. 2003. *Introductory lectures on convex optimization: A basic course*. Vol. 87. Springer Science & Business Media.

[72] Yurii E Nesterov. 1983. A method for solving the convex programming problem with convergence rate O (1/k^ 2). In *Dokl. akad. nauk Sssr*, Vol. 269. 543–547.

[73] Julie Nutini, Mark Schmidt, Issam Laradji, Michael Friedlander, and Hoyt Koepke. 2015. Coordinate descent converges faster with the gauss-southwell rule than random selection. In *International Conference on Machine Learning*. PMLR, 1632–1641.

[74] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Previous number = SIDL-WP-1999-0120.

[75] Boris T Polyak. 1964. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics* 4, 5 (1964), 1–17.

[76] Ştefan Postăvaru, Anton Tsitsulin, Filipe Miguel Gonçalves de Almeida, Yingtao Tian, Silvio Lattanzi, and Bryan Perozzi. 2020. InstantEmbedding: Efficient Local Node Representations. *arXiv preprint arXiv:2010.06992* (2020).

[77] Samantha Samuelson, Hesameddin Mohammadi, and Mihailo R Jovanović. 2020. Transient growth of accelerated first-order methods. In *2020 American Control Conference (ACC)*. IEEE, 2858–2863.

[78] Damien Scieur, Alexandre d'Aspremont, and Francis Bach. 2020. Regularized nonlinear acceleration. *Mathematical Programming* 179, 1 (2020), 47–83.

[79] Jonathan Richard Shewchuk et al. 1994. An introduction to the conjugate gradient method without the agonizing pain.

[80] Daniel A Spielman and Shang-Hua Teng. 2014. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Anal. Appl.* 35, 3 (2014), 835–885.

[81] Nikita Spirin and Jiawei Han. 2012. Survey on web spam detection: principles and algorithms. *ACM SIGKDD explorations newsletter* 13, 2 (2012), 50–64.

[82] Thomas Strohmer and Roman Vershynin. 2009. A randomized Kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications* 15, 2 (2009), 262–278.

[83] Lubos Takac and Michal Zabovsky. 2012. Data analysis in public social networks. In *International scientific conference and international workshop present day trends of innovations*, Vol. 1. Present Day Trends of Innovations Lamza Poland.

[84] Zekun Tong, Yuxuan Liang, Henghui Ding, Yongxing Dai, Xinke Li, and Changhu Wang. 2021. Directed Graph Contrastive Learning. *Advances in Neural Information Processing Systems* 34 (2021).

[85] Zekun Tong, Yuxuan Liang, Changsheng Sun, Xinke Li, David Rosenblum, and Andrew Lim. 2020. Digraph inception convolutional networks. *Advances in neural information processing systems* 33 (2020), 17907–17918.

[86] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 world wide web conference*. 539–548.

[87] 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.

[88] Stephen J Wright. 2015. Coordinate descent algorithms. *Mathematical Programming* 151, 1 (2015), 3–34.

[89] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. In *Proceedings of the 2021 International Conference on Management of Data*. 1996–2008.

[90] Jaewon Yang and Jure Leskovec. 2012. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*. 1–8.

[91] David Young. 1954. Iterative methods for solving partial difference equations of elliptic type. *Trans. Amer. Math. Soc.* 76, 1 (1954), 92–111.

[92] 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.

[93] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems* 31 (2018), 5165–5175.

Zhen Chen, Xingzhi Guo, Baojian Zhou, Deqing Yang, and Steven Skiena

A STOCHASTIC MATRIX OF $\mathcal{G}$ AND DANGLING NODES

Given the directed graph $\mathcal{G}$, we present several standard ways to construct row stochastic matrix P. Recall D is the diagonal out-degree matrix of $\mathcal{G}$ and A is the associated adjacency matrix of $\mathcal{G}$. **Case 1.** If all nodes in $\mathcal{V}$ are not dangling nodes, that is, each node has at least one outgoing edge, then $P = D^{-1}A$; **Case 2.** If some nodes are dangling nodes, two popular ways to create P. Let $S = \{v : d_v = 0, v \in \mathcal{V}\}$ be the set of dangling nodes.

- For each dangling node, we create n edges pointing all nodes, and the augmented degree matrix is $D' = D + n \operatorname{diag}(1_S)$. So, $P = (D')^{-1}A$.
- We add a dummy node v and create $|S|$ edges pointing from S to v meanwhile adding self-loop for node v. Hence, $P = \left[D^{-1}A; [1_S^\top, 1] \right]$ where ; is the sign of row append.

For more options of creating P, one can refer to Section 3 of Gleich [38].

B STRONGLY-CONVEX AND SMOOTH OF f AND THE CONVERGENCE OF NAG METHOD

Given a convex function $f : \mathbb{R}^n \to \mathbb{R}$, we say f is μ-strongly convex if $\forall x, y \in \mathbb{R}^n$, we have

$$f(x) - f(y) \le \nabla f(x)^\top (x - y) - \frac{\mu}{2} \|x - y\|_2^2.$$

We say f is L-smooth, if for all $x, y \in R^n$, we have

$$f(x) - f(y) - \nabla f(y)^\top (x - y) \le \frac{L}{2} \|x - y\|_2^2.$$

Define Nesterov's Accelerated Gradient descent method (NAG) as the following iteration procedure

$$y_{t+1} = x_t - \frac{1}{\beta} \nabla f(x_t),$$

$$x_{t+1} = \left(1 + \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right) y_{t+1} - \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} y_t,$$

where $\kappa = L/\mu$ is the condition number of f.

Theorem 8 ([24]). *Let f be μ-strongly convex and L-smooth, then NAG method has the following convergence rate*

$$f(y_t) - f(x^*) \le \frac{\mu + L}{2} \|x_1 - x^*\|_2^2 \exp\left(-\frac{t-1}{\sqrt{\kappa}} \right). \tag{21}$$

C PROOF OF THEOREM 6

PROOF. Notice that f defined in (16) is α-strongly convex and $(2 - \alpha)$-strongly smooth. Hence letting $\mu = \alpha$ and $L = 2 - \alpha$, by applying Thm. 8, we have

$$f(y^t) - f(y^*) \le \frac{\mu + L}{2} \|x^0 - y^*\|_2^2 \exp\left(-\frac{t}{\sqrt{\kappa}} \right)$$

$$= \|x^0 - y^*\|_2^2 \exp\left(-\frac{t}{\sqrt{(2 - \alpha)/\alpha}} \right).$$

Note for any strongly convex function f, the optimization error can also be lower bounded by $\frac{\alpha}{2} \|y^t - y^*\|_2 \le f(y^t) - f(y^*)$, then we reach

$$\frac{\alpha}{2} \|y^t - y^*\|_2^2 \le \|x^0 - y^*\|_2^2 \exp\left(-\frac{t}{\sqrt{(2 - \alpha)/\alpha}} \right),$$

as $\|x^0 - y^*\|_2 = \|y^*\|_2 \le \|y^*\|_1 = \|D^{-1/2}x^*\|_1 \le \sqrt{1/d_{\min}} \le 1$. Notice $x^{t+1} = D^{1/2}y^{t+1}$, we then have

$$\|D^{-1/2}(x^{t+1} - x^*)\|_2 \le \sqrt{\frac{2}{\alpha}} \exp\left(-\frac{t}{2\sqrt{(2 - \alpha)/\alpha}} \right)$$

Note for any $x \in \mathbb{R}^n$, we have $\|x\|_1 \le \sqrt{n}\|x\|_2$. Then we have

$$\|x^{t+1} - x^*\|_1 \le d_{\max} \sqrt{\frac{2n}{\alpha}} \exp\left(-\frac{t}{2\sqrt{(2 - \alpha)/\alpha}} \right)$$

Use the fact $\|x^{t+1} - x^*\|_\infty \le \|x^{t+1} - x^*\|_1$ and apply per-iteration operations m of each iteration, we have the above operation complexity bound. □

D MORE EXPERIMENTAL RESULTS

D.1 Power law distribution of PPVs

As shown in Fig. 7, we present the magnitudes of x^* as a function of their ranks. We sort all magnitudes in descending order and label these magnitudes from rank 1 to rank n. Evidently, these magnitudes adhere to the power law distribution with a cutoff, as described by Clauset [29]. More specifically, let $p(x)$ represent a magnitude where x is the associated ranking ID; this yields the following relation

$$p(x) \propto L(x)x^{-a}, \text{ where } L(x) = e^{-bx}. \tag{22}$$

One can find suitable parameters a and b to fit these curves using (22).
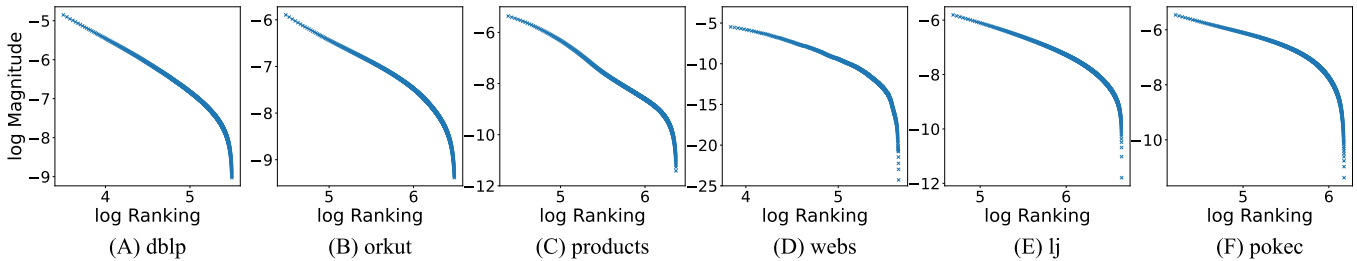
Figure 7: The power law distribution of magnitudes x^* of six graphs. We randomly pick one node from a graph and run the power iteration algorithm to obtain high precision x^*. It is important to note that in this particular setting, the power law distribution is characterized by a power law with a cutoff, as discussed in Clauset's work [29].

D.2 Comparison of empirical bounds

To further validate the effectiveness of our parameterized bound, we carry out a series of experiments on two additional graph datasets, specifically, *livejournal* and *pokec* shown in Fig. 8 and 9, respectively. When the value of α is relatively small, our bound is similar to B_1 for small values of ϵ, and it is empirically tighter than B_2, irrespective of whether ϵ is small or large. As α increases, the comparative tightness of our bound becomes markedly more pronounced.
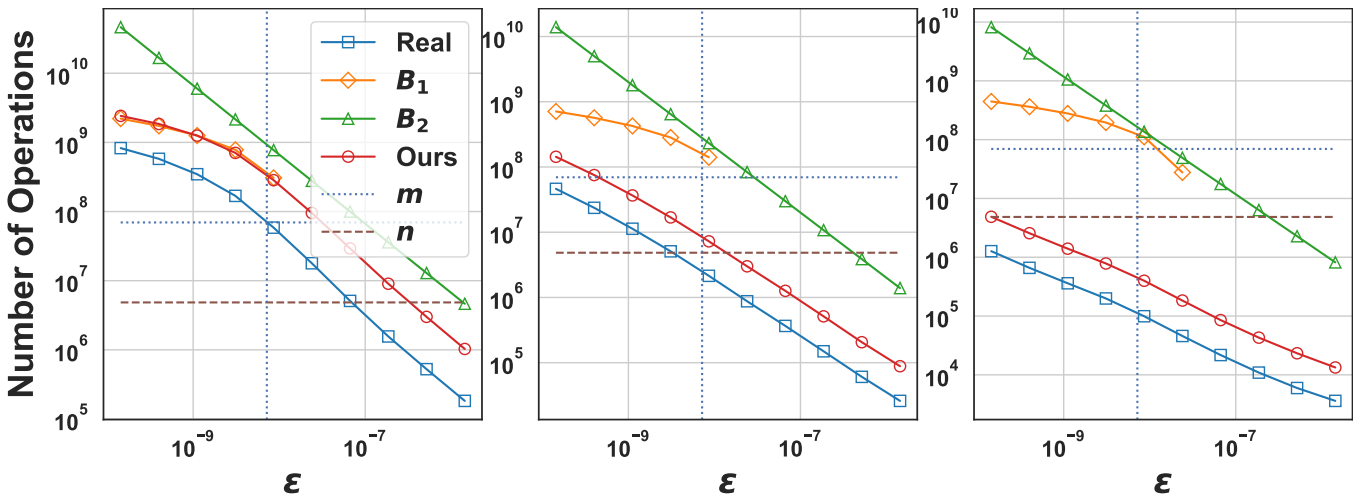

Figure 8: The bounds of *livejournal* dataset as a function of ϵ. The vertical line is where $\epsilon = (2m)^{-1}$. Left: $\alpha = 0.15$, Middle: $\alpha = 0.5$, and Right: $\alpha = 0.85$.
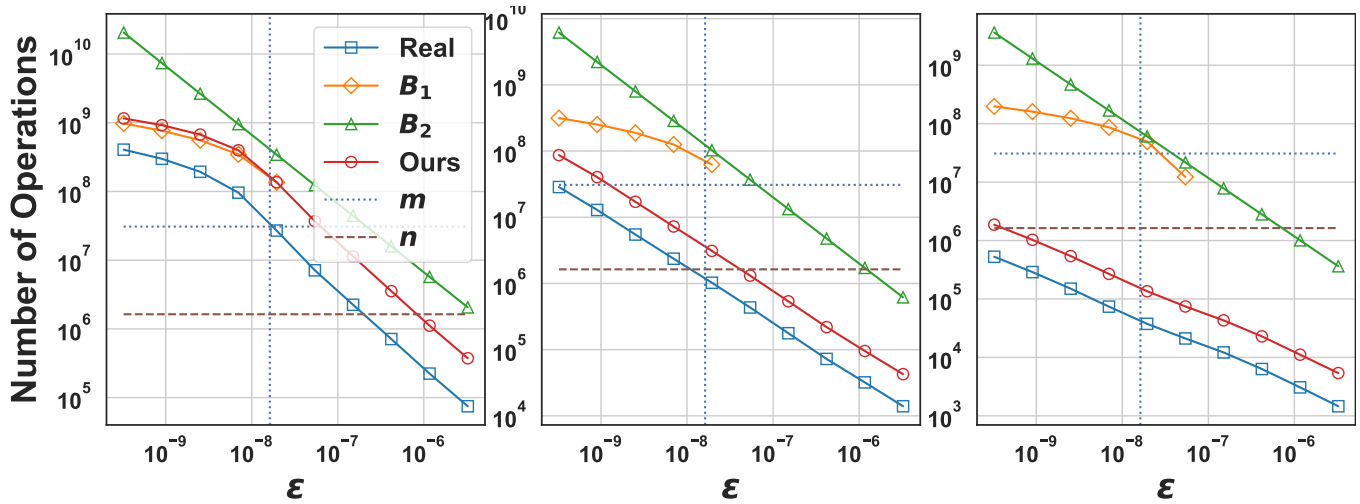
Figure 9: The bounds of *pokec* dataset as a function of ϵ.

D.3 More experiments on the run time and the number of operations comparison

A-F of Fig. 10, 11, 12 and 13 present the run time and the number of operations of PPV methods when $\alpha = 0.05$ and $\alpha = 0.2$, respectively. When compared to the setting where $\alpha = 0.15$, SOR-based methods exhibit a more significant improvement when $\alpha = 0.05$. For example, in the dataset of *products*, our PwrPushSOR is more than 5 times faster than FwdPush method.

Observation of superlinear behavior.   During the course of our experiments, we discerned that both PwrPushSOR and FwdPushSOR exhibited the potential for superlinear behavior during the final few iterations. For example, when setting α at 0.05, the runtime required by PwrPushSOR displayed superlinearity in relation to ℓ_1 error (see E of Fig. 10 and 11). Interestingly, this phenomenon mirrors the well-known superlinear convergence behavior observed when employing the conjugate gradient method to solve large symmetric systems of equations. This intriguing pattern certainly warrants further exploration and study.
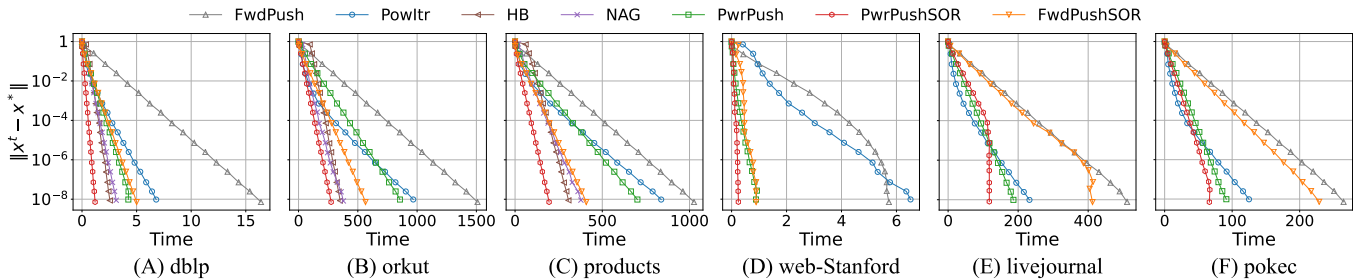

Figure 10: Actual l_1-error v.s. execution time (seconds), $\alpha = 0.05$.
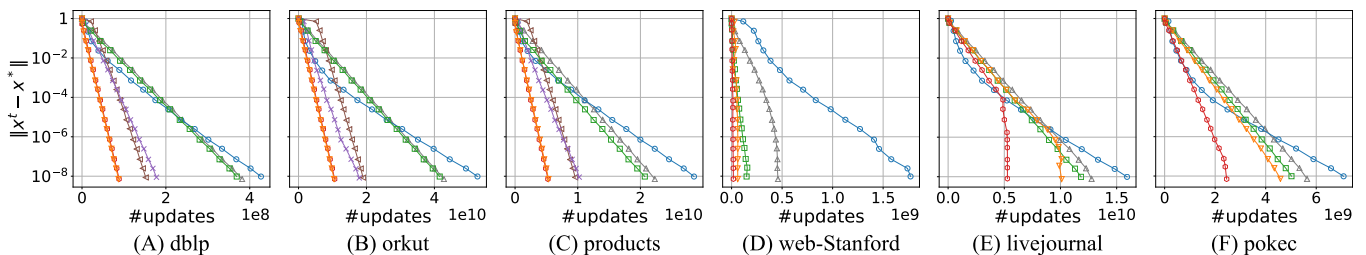

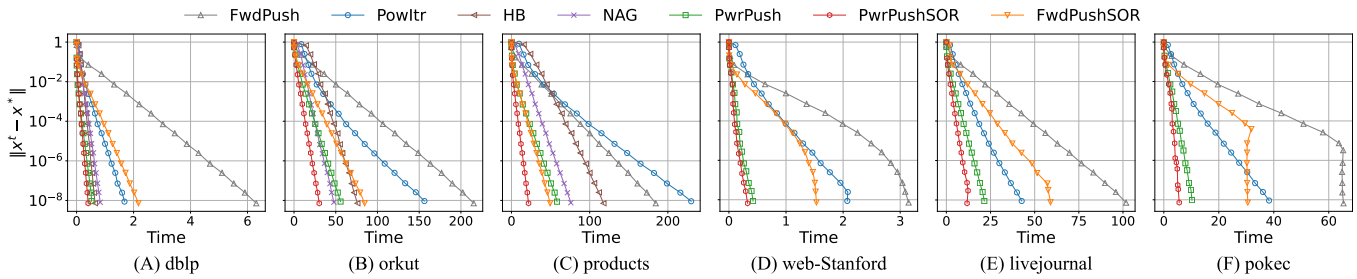Figure 11: Actual l_1-error v.s. #residue updates, $\alpha = 0.05$

Figure 12: Actual l_1-error v.s. execution time (seconds), $\alpha = 0.2$.
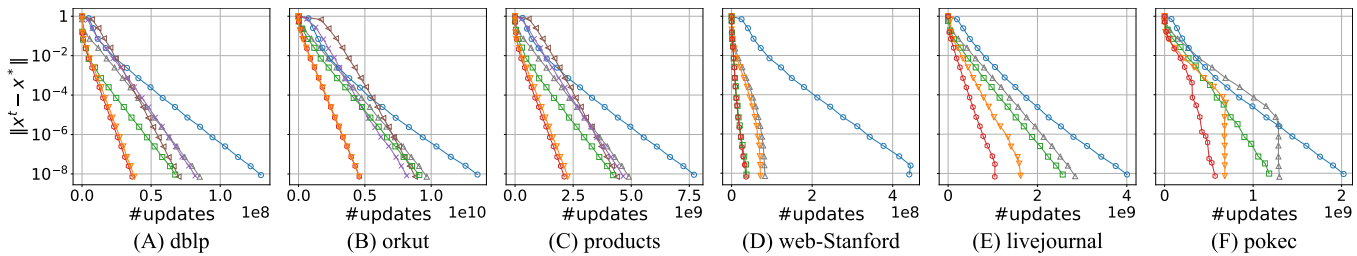

Figure 13: Actual l_1-error v.s. #residue updates, $\alpha = 0.2$

Significant speedup of local SOR methods even with large α. As depicted in Fig. 14 and Fig. 15, it is evident that the local SOR method still requires fewer runtime operations to reach equivalent approximate solutions, even when α is large. This consistently efficient performance clearly underlines the effectiveness and versatility of our method across a broad spectrum of settings.
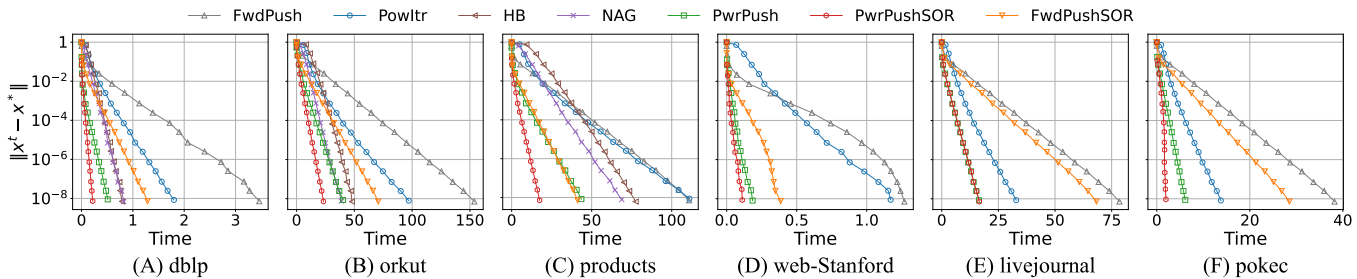

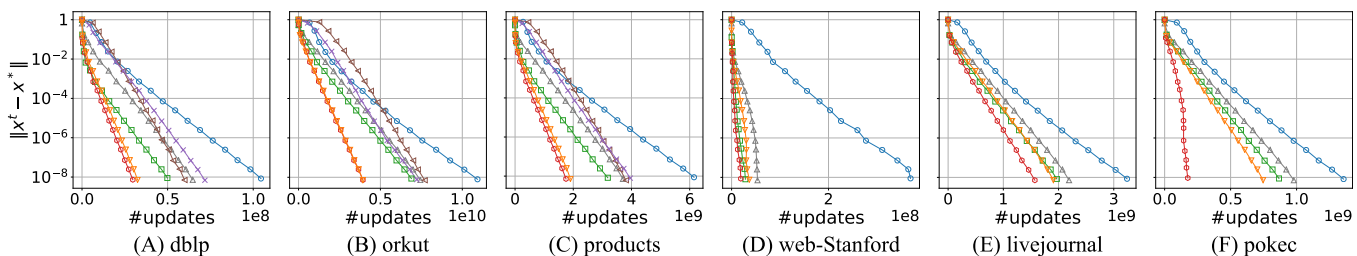Figure 14: Actual l_1-error v.s. execution time (seconds), $\alpha = 0.25$.

Figure 15: Actual l_1-error v.s. #residue updates, $\alpha = 0.25$

Zhen Chen, Xingzhi Guo, Baojian Zhou, Deqing Yang, and Steven Skiena

E FWDPUSH-MEAN ALGORITHM

We present FwdPush-Mean in Algo. 4. Compared with FwdPush, it computes the statistic of average residuals of active nodes $\bar{r}$ at Line 8. Residuals of nodes that are less than this average will postpone to the next epoch. Note the run time of computing $\bar{r}$ is not bigger than $|S_t|$; hence the total run time complexity will be the same as FwdPush. In practice, we found this strategy could help to reduce the total amount of push operations as shown in Fig. 3.

Algorithm 4 FwdPush-Mean$(\mathcal{G}, \epsilon, \alpha, s)$ with a dummy node

---

1: Initialization: $\boldsymbol{r} = \boldsymbol{e}_s, \boldsymbol{x} = \boldsymbol{0}$
2: $Q = [s, \ddagger]$ // Dummy node $\ddagger$ at the end of Q
3: $t = 0, t' = 0$
4: $\bar{r} = 0$
5: **while** Q.size() $\neq 1$ **do**
6:      $u = Q$.pop()
7:      **if** $u == \ddagger$ **then**
8:          $\bar{r} = \sum_{i \in \mathcal{S}^t} \frac{r_i/d_i}{|\mathcal{S}^t|}$
9:          Q.push(u)
10:          $t = t + 1$ // Next epoch time
11:          **continue**
12:      **if** $\frac{r_u}{d_u} < \bar{r}$ **then**
13:          Q.push(u) // Postpone current active node to next
14:          **continue**
15:      $x_u = x_u + \alpha \cdot r_u$
16:      **for** $v \in \text{Nei}(u)$ **do**
17:          $r_v = r_v + \frac{(1-\alpha)r_u}{d_u}$
18:          **if** $r_v \geq \epsilon d_v$ and $v \notin Q$ **then**
19:              Q.push(v)
20:      $r_u = 0$
21:      $t' = t' + 1$
22: **Return** $\boldsymbol{x}$