Personalized PageRank Estimation and Search: A Bidirectional Approach

Peter Lofgren
Department of CS
Stanford University
plofgren@cs.stanford.edu

Siddhartha Banerjee
School of ORIE
Cornell University
sbanerjee@cornell.edu

Ashish Goel
Department of MS&E
Stanford University
ashishg@stanford.edu

ABSTRACT

We present new algorithms for Personalized PageRank estimation and Personalized PageRank search. First, for the problem of estimating Personalized PageRank (PPR) from a source distribution to a target node, we present a new bidirectional estimator with simple yet strong guarantees on correctness and performance, and 3x to 8x speedup over existing estimators in experiments on a diverse set of networks. Moreover, it has a clean algebraic structure which enables it to be used as a primitive for the Personalized PageRank Search problem: Given a network like Facebook, a query like "people named John," and a searching user, return the top nodes in the network ranked by PPR from the perspective of the searching user. Previous solutions either score all nodes or score candidate nodes one at a time, which is prohibitively slow for large candidate sets. We develop a new algorithm based on our bidirectional PPR estimator which identifies the most relevant results by sampling candidates based on their PPR; this is the first solution to PPR search that can find the best results without iterating through the set of all candidate results. Finally, by combining PPR sampling with sequential PPR estimation and Monte Carlo, we develop practical algorithms for PPR search, and we show via experiments that our algorithms are efficient on networks with billions of edges.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process; G.2.2 [**Graph Theory**]: Graph Algorithms

Keywords

Personalized Search; Personalized PageRank; Social Network Analysis

1. INTRODUCTION

On social networks, personalization is necessary for returning relevant results for a query. For example, if a user searches for a common name like John on a social network like Facebook, the results should depend on who is doing the search and who their friends are. A good personalized model for measuring the importance of a node t to a searcher s is Personalized PageRank $\pi_s(t)$ [20, 13, 12] – this motivates a natural *Personalized PageRank Search Problem*: Given

- a network with nodes V (each associated with a set of keywords) and edges E (possibly weighted and directed),
- a keyword inducing a set of targets:
$$T = \{t \in V : t \text{ is relevant to the keyword}\}$$
- a searching user $s \in V$ (or more generally, a distribution over starting nodes),

return the top-k targets $t_1, \ldots, t_k \in T$ ranked by Personalized PageRank $\pi_s(t_i)$.

The importance of personalized search extends beyond social networks. For example, personalized PageRank can be used to rank items in a bi-partite user-item graph, in which there is an edge from a user to an item if the user has liked that item. This has proven useful on YouTube when recommending videos [5] and on Twitter for suggested users [3, 12]. On the web graph there is a large body of work on using Personalized PageRank to rank web pages (e.g. [14, 13]). The most clear-cut motivation for our work is for the social network name-search application discussed above, which we use as a running example in this paper.

The personalized search problem is difficult because every searching user has a different ranking on the target nodes. One naive solution would be to precompute the ranking for every searching user, but if our network has n users this requires $\Theta(n^2)$ storage, which is clearly infeasible. Another naive baseline would be to use power iteration [20] at query time, but that would take $\Theta(m)$ computation between the search query and response, where m is the number edges, which is also clearly infeasible. The challenge we face is to create a data structure much smaller than $O(n^2)$ which allows us to rank $|T|$ targets in response to a query in less than $O(|T|)$ time.

Previous work has considered the problem of personalized search on social networks. For example Vieira et. al. [24] consider this problem and provide excellent motivation for why results to a name-search query should be ranked based the friendships of the searching user and the candidate results. They and others (e.g. [4]) propose to rank results by shortest path length. However, this metric doesn't take into account the number of paths between two users: If the searcher and two results John A and John B are distance 3 apart, but the searcher and John A are connected by 100 length-3 paths while the searcher and John B are connected

WSDM'16, February 22–25, 2016, San Francisco, CA, USA.
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-3716-8/16/02…$15.00
DOI: http://dx.doi.org/10.1145/2835776.2835823

by a single length-3 path, than John A should be ranked above John B, yet the shortest distance can't distinguish the two. To the best of our knowledge, no prior work has solved the Personalized PageRank search problem using less than $O(n^2)$ storage and $O(|T|)$ query time. The reason we are able to solve this is by exploiting a new bidirectional method of PageRank, introduced in [19] and improved in this work.

Our search algorithm is based on two key ideas. The first is that we can find the top target nodes without having to consider each separately by *sampling a target $t_i \in T$ in proportion to its Personalized PageRank $\pi_s(t_i)$*. Because the top results typically have a much higher personalized PageRank than an average result, by sampling we can find the top results without iterating over all the results. The second idea is that the probability of a random walk exactly reaching an element in T is often very small, but by pre-computing an expanded set of nodes around each target, we can efficiently sample random walks until they get close to a target node, and then use the pre-computed data to sample targets t_i in proportion to $\pi_s(t_i)$.

There are currently two main limitations to our work. First, because we do pre-computation on the set of nodes relevant to a query, we need the set of queries to be known in advance, although in the case of name search we can simply let the space of queries be the set of all first or last names. Second, the pre-computed storage is significant; for name-search it is $O\left(n\sqrt{m}\right)$ to achieve query running time $O(\sqrt{m})$, where n is the number of nodes and m is the number of edges. However, large graphs tend to be sparse, so this is still much smaller than $O\left(n^2\right)$ and is less storage than any prior solution to the Personalized PageRank Search problem. Also, pre-computation doesn't need to be done for all queries: for queries with small or very large target sets we describe alternative algorithms which do not require pre-computation. These alternatives also overcome the limitation on queries being known in advance.

Contributions: To summarize, in this work we present:

- A new bidirectional PageRank estimator, `Bidirectional-PPR` (section 3), which has the following features:
  - *Simple analysis*: We combine a simple linear-algebraic invariant with standard concentration bounds. The new analysis also allows generalizations to arbitrary Markov Chains, as done in [6].
  - *Easy to implement*: The complete algorithm is only 18 lines of pseudo-code.
  - *Significant empirical speedup*: For a given accuracy, it executes 3x-8x faster than the fastest previous algorithm, `FAST-PPR` [19], on a diverse set of networks.
  - *Simple linear structure:* As shown in section 4.1, the estimates are a simple dot-product between a forward vector x^s and a reverse vector y^t – this enables the development of PPR samplers.
  - *Parallelizability:* Because the estimate is a dot-product, the precomputed vectors can be sharded across many servers, and the estimation algorithm can be naturally distributed, as shown in [11].
- Two new solutions to the Personalized PageRank Search problem – `BiPPR-Grouped` and `BiPPR-Sampling`. Given any set of targets T:
  - `BiPPR-Precomp-Grouped` precomputes and stores the reverse vectors $y^t, t \in T$ after grouping them by their coordinates. This exploits the natural sparsity of these vectors to speed-up the computation of the PPR estimates at runtime.
  - `BiPPR-Precomp-Sampling` samples nodes $t \in T$ proportional to their PPR $\pi_s(t)$. Now since PPR values are usually highly skewed, this serves as a good proxy for finding the top k search results.
- Extensive simulations on the Twitter-2010 network to test the scalability of our algorithms for PPR-search. Our experiments demonstrate the trade-off between storage and runtime, and suggest that we should use a combination of methods, depending on the size of the set of targets T induced by the keyword.

2. PRELIMINARIES

We are given a graph $G = (V, E)$ with n nodes and m edges. Define the out-neighbors of a node u by $\mathcal{N}^{out}(u) = \{v : (u, v) \in E\}$ and let $d^{out}(u) = \left|\mathcal{N}^{out}(u)\right|$; define $\mathcal{N}^{in}(u)$ and $d^{in}(u)$ similarly. Define the average degree of nodes $\bar{d} = \frac{m}{n}$. If the graph is weighted, for each $(u, v) \in E$ there is some positive weight $w_{u,v}$; otherwise we define $w_{u,v} = \frac{1}{d^{out}(u)}$ for all $(u, v) \in E$. For simplicity we assume the weights are normalized such that for all u, $\sum_v w_{u,v} = 1$.

The personalized PageRank from source distribution σ to target node t can be defined using linear algebra as the solution to the equation $\pi_\sigma = \pi_\sigma(\alpha\sigma + (1-\alpha)W)$, or equivalently defined using random walks

$$\pi_\sigma(t) = \Pr[\text{a random walk starting from } s \sim \sigma \\ \text{of length} \sim \text{geometric}(\alpha) \text{ stops at } t]$$

as shown in [2]. For concreteness, in this paper we often assume $\sigma = e_s$ for some single node s (meaning the random walks always start at a single node s), but all results extend in a straightforward manner to any starting distribution σ.

Personalized PageRank was first defined in the original PageRank paper [20]. For more on the motivation of Personalized PageRank, see [13] and the survey [10].

3. PAGERANK ESTIMATION

In this section, we present our new bidirectional algorithm for PageRank estimation. We first develop the basic algorithm along with its theoretical performance guarantees; next, we outline some extensions of the basic algorithm; finally, we conclude the section with simulations demonstrating the efficiency of our technique.

The `Bidirectional-PPR` Algorithm

At a high level, our algorithm estimates $\pi_s(t)$ by first working backwards from t to find a set of intermediate nodes 'near' t and then generating random walks forwards from s to detect this set.

The reverse work from t is done via the `Approx-Contributions` algorithm (see Algorithm 1) of Andersen et. al. [1], that, given a target t and a desired *additive error-bound* $r_{\max}$, produces estimates $p^t(s)$ of the PPR $\pi_s(t)$ for every start node s. More specifically, the `Approx-Contributions` algorithm produces two non-negative vectors $p^t \in \mathbb{R}^n$ and $r^t \in \mathbb{R}^n$ which satisfy the following invariant (Lemma 1 in [1])

$$\pi_s(t) = p^t(s) + \sum_{v \in V} \pi_s(v) r^t(v). \tag{1}$$

`Approx-Contributions` terminates once each residual value $r^t(v) < r_{\max}$; now, viewing $\sum_{v \in V} \pi_s(v) r^t(v)$ as an error term, Andersen et al. observe that $p^t(s)$ estimates $\pi_s(t)$ up to a maximum additive error of $r_{\max}$.

Our `Bidirectional-PPR` algorithm is based on the observation that in order to estimate $\pi_s(t)$ for a *particular* (s,t) pair, we can boost the accuracy by sampling and adding the residual values $r^t(v)$ from nodes v which are sampled from π_s. To see this, we first interpret Equation (1) as an expectation:

$$\pi_s(t) = p^t(s) + \mathbb{E}_{v\sim\pi_s}[r^t(v)].$$

Now, since $\max_v r^t(v) < r_{\max}$, the expectation $\mathbb{E}_{v\sim\pi_s(v)}[r^t(v)]$ can be efficiently estimated using Monte Carlo. To do so, we generate $w = c\frac{r_{\max}}{\delta}$ random walks of length $Geometric(\alpha)$ from start node s; here c is a parameter which depends on the desired accuracy, $r_{\max}$ is the maximum residual after running `Approx-Contributions`, and δ is the minimum PPR value we want to accurately estimate. Let V_i be the final node of the i^{th} random walk; note that $\Pr[V_i = v] = \pi_s(v)$. Let $X_i = r^t(V_i)$ denote the residual from the final node of the ith random walk, and $\bar{X} = \frac{1}{w}\sum_{i=1}^{w} X_i$. Then `Bidirectional-PPR` returns as an estimate of $\pi_s(t)$:

$$\widehat{\pi}_s(t) = p^t(s) + \bar{X}$$

The complete pseudocode is given in Algorithm 2.

Algorithm 1 `Approx-Contributions`$(G, \alpha, t, r_{\max})$ [1]

```
Inputs: graph $G$ with edge weights $w_{u,v}$, teleport probability $\alpha$, target node $t$, maximum residual $r_{\max}$
1: Initialize (sparse) estimate-vector $p_t = \vec{0}$ and (sparse) residual-vector $r_t = e_t$ (i.e. $r_t(v) = 1$ if $v = t$; else 0)
2: while $\exists v \in V\ s.t.\ r_t(v) > r_{\max}$ do
3:     for $u \in \mathcal{N}^{in}(v)$ do
4:         $r_t(u) \mathrel{+}= (1-\alpha) w_{u,v} r_t(v)$
5:     end for
6:     $p_t(v) \mathrel{+}= \alpha r_t(v)$
7:     $r_t(v) = 0$
8: end while
9: return $(p_t, r_t)$
```

Algorithm 2 `Bidirectional-PPR`(s, t, δ)

```
Inputs: graph $G$, teleport probability $\alpha$, start node $s$, target node $t$, minimum probability $\delta$, accuracy parameter $c$ (in our experiments we use $c = 7$)
1: Choose $r_{\max} = c_{\text{balance}}/\sqrt{m})$, where $c_{\text{balance}}$ is tuned to balance forward and reverse work. (For greater efficiency, use the balanced version described in Section 3.)

2: $(p_t, r_t)$ = Approx-Contributions$(t, r_{\max}, \alpha)$
3: Set number of walks $w = c r_{\max}/\delta$    (cf. Theorem 1)
4: for index $i \in [w]$ do
5:     Sample a random walk starting from $s$ (sampling a start from $s$ if $s$ is a distribution), stopping after each step with probability $\alpha$; let $v_i$ be the endpoint
6:     Set $X_i = r_t(v_i)$
7: end for
8: return $\widehat{\pi}_s(t) = p_t(s) + (1/w)\sum_{i\in[w]} X_i$
```

Accuracy Analysis

We first prove that `Bidirectional-PPR` returns an estimate with the desired accuracy with high probability:

THEOREM 1. *Given start node s (or source distribution σ), target t, minimum PPR δ, maximum residual $r_{max} > \frac{2e\delta}{\alpha\epsilon}$, relative error $\epsilon \le 1$, and failure probability p_{fail},* ***Bidirectional-PPR*** *outputs an estimate $\widehat{\pi}_s(t)$ such that with probability at least $1 - p_{fail}$ the following hold:*

- *If $\pi_s(t) \ge \delta$:* $|\pi_s(t) - \hat{\pi}_s(t)| \le \epsilon\pi_s(t)$.
- *If $\pi_s(t) \le \delta$:* $|\pi_s(t) - \hat{\pi}_s(t)| \le 2e\delta$.

The above result shows that the estimate $\hat{\pi}_s(t)$ can be used to distinguish between 'significant' and 'insignificant' PPR pairs: for pair (s,t), Theorem 1 guarantees that if $\pi_s(t) \ge \frac{(1+2e)\delta}{(1-\epsilon)}$, then the estimate is greater than $(1+2e)\delta$, whereas if $\pi_s(t) < \delta$, then the estimate is less than $(1+2e)\delta$. The assumption $r_{\max} > \frac{2e\delta}{\alpha\epsilon}$ is easily satisfied, as typically $\delta = O\left(\frac{1}{n}\right)$ and $r_{\max} = \Omega\left(\frac{1}{\sqrt{m}}\right)$. The proof is a straightforward application of Chernoff bounds and appears in the full version of this paper.

Running Time Analysis

The runtime of `Bidirectional-PPR` depends on the target t: if t has many in-neighbors and/or large global PageRank $\pi(t)$, then the running time will be slower than for a random t. Theorem 1 of [1] states that `Approx-Contributions` $(G, \alpha, t, r_{\max})$ performs $\frac{n\pi(t)}{\alpha r_{\max}}$ pushback operations, and the exact running time is proportional to the sum of the in-degrees of all the nodes where we pushback from. In the worst case, we might have $d^{in}(t) = \Theta(n)$ and `Bidirectional-PPR` takes $\Theta(n)$ time. However, for *a uniformly chosen target node*, we can prove the following:

THEOREM 2. *For any start node s (or source distribution σ), minimum PPR δ, maximum residual r_{max}, relative error ϵ, and failure probability p_{fail}, if the target t is chosen uniformly at random, then* `Bidirectional-PPR` *has expected running time*

$$O\left(\sqrt{\frac{\bar{d}}{\delta}}\frac{\sqrt{\log\left(1/p_{fail}\right)}}{\alpha\epsilon}\right).$$

In contrast, the running time for `Monte-Carlo` to achieve the same accuracy guarantee is $O\left(\frac{1}{\delta}\frac{\log(1/p_{\text{fail}})}{\alpha\epsilon^2}\right)$, and the running time for `Approx-Contributions` is $O\left(\frac{\bar{d}}{\delta\alpha}\right)$. The fastest previous algorithm for this problem, the `FAST-PPR` algorithm of [19], has an average running time bound of $O\left(\frac{1}{\alpha\epsilon^2}\sqrt{\frac{\bar{d}}{\delta}}\sqrt{\frac{\log(1/p_{\text{fail}})\log(1/\delta)}{\log(1/(1-\alpha))}}\right)$ for uniformly chosen targets. The running time bound of `Bidirectional-PPR` is thus asymptotically better than `FAST-PPR`, and in experiments the constants required for the same accuracy are smaller, making `Bidirectional-PPR` is 3 to 8 times faster on a diverse set of graphs.

PROOF. In [18], it is proven that for a uniform random t, Approx-Contributions runs in average time $\frac{\bar{d}}{\alpha r_{\max}}$ where $\bar{d}$ is the average degree of a node. On the other hand, from Theorem 1, we know that we need to generate $O\left(\frac{r_{\max}}{\delta\epsilon^2}\ln\left(1/p_{\text{fail}}\right)\right)$ random walks, each of which can be sampled in average time $1/\alpha$. Finally, we choose $r_{\max} = \frac{\epsilon}{\alpha}\sqrt{\frac{\bar{d}}{\ln(2/p_{\text{fail}})}}$ to minimize our running time bound and get the claimed result. □

Extensions Bidirectional-PageRank extends naturally to generalized PageRank using a source distribution σ rather than a single start node – we simply sample an independent starting node for each walk, and replace $p_t(s)$ with the expected value of $p_t(s)$ when s is sampled from the starting distribution.

The dynamic runtime-balancing method proposed in [19] can improve the running time of Bidirectional-PageRank in practice. In this technique, $r_{\max}$ is chosen dynamically in order to balance the amount of time spent by `Approx-Contributions` and the amount of time spent generating random walks. To implement this, we modify `Approx-Contributions` to use a priority queue in order to always push from the node v with the largest value of $r_t(v)$. We also change the while loop so that it terminates when the amount of time spent achieving the current value of $r_{\max}$ first exceeds the predicted amount of time required for sampling random walks, $c_{\text{walk}} \cdot c \cdot \frac{r_{\max}}{\delta}$, where c_{walk} is the average time it takes to sample a random walk. For full pseudocode, see [17].

Experimental Validation

We now compare `Bidirectional-PPR` to its predecessor algorithms (namely: `FAST-PPR` [18], `Monte Carlo` [2, 9] and `Approx-Contributions` [1]). The experimental setup is identical to that in [18]; for convenience, we describe it here in brief. We perform experiments on 6 diverse, real-world networks: two directed social networks (Pokec (31M edges) and Twitter-2010 (1.5 billion edges)), two undirected social network (Live-Journal (69M edges) and Orkut (117M edges)), a collaboration network (dblp (6.7M edges)), and a web-graph (UK-2007-05 (3.7 billion edges)). Since all algorithms have parameters that enable a trade-off between running time and accuracy, we first choose parameters such that the mean relative error of each algorithm is approximately 10%. For bidirectional-PPR, we find that setting $c = 7$ (i.e., generating $7 \cdot \frac{r_{\max}}{\delta}$ random walks) results in a mean relative error less than 8% on all graphs; for the other algorithms, we use the settings determined in [18]. We then repeatedly sample a uniformly-random start node $s \in V$, and a random target $t \in T$ sampled either uniformly or from PageRank (to emphasize more important targets). For both `Bidirectional-PPR` and `FAST-PPR`, we used the dynamic-balancing heuristic described above. The results are shown in Figure 1.

Note that `Bidirectional-PPR` is 3 to 8 times faster than `FAST-PPR` across all graphs. In particuar, `Bidirectional-PPR` only needs to sample $7\frac{r_{\max}}{\delta}$ random walks, while FAST-PPR needs $350\frac{r_{\max}}{\delta}$ walks to achieve the same mean relative error. This is because `Bidirectional-PPR` is unbiased, while `FAST-PPR` has a bias from `Approx-Contributions`.

4. PERSONALIZED PAGERANK SEARCH

We now turn from Personalized PageRank estimation to the Personalized PageRank search problem:

Given a start node s (or distribution σ) and a query q which filters the set of all targets to some list $T = \{t_i\} \subseteq V$, return the top-k targets ranked by $\pi_s[t_i]$.

We consider as baselines two algorithms which require no pre-computation. They are efficient for certain ranges of $|T|$, but our experiments show they are too slow for real-time search across most values of $|T|$:

- `Monte-Carlo` [2, 9]: Sample random walks from s, and filter out any walk whose endpoint is not in T. If we desire n_s samples, this takes time $O(n_s/\pi_s[T])$, where $\pi_s[T] := \sum_{t\in T} \pi_s[t]$ is the probability that a random walk terminates in T. This method works well if T is large, but in our experiments on Twitter-2010 it takes minutes per query for $|T| = 1000$ (and hours per query for $|T| = 10$).
- `Bidirectional-PPR`: On the other hand, we can estimate $\pi_s[t]$ to each $t \in T$ separately using `Bidirectional-PPR`. This has an average-case running time $O\left(|T|\sqrt{\bar{d}/\delta_k}\right)$ where δ_k is the PPR of the k^{th} best target. This method works well if T is small, but is too slow for large T; in our experiments, it takes on the order of seconds for $|T| \leq 100$, but more than a minute for $|T| = 1000$.

If we are allowed pre-computation, then we can improve upon `Bidirectional-PPR` by precomputing and storing a reverse vector from all target nodes. To this end, we first observe that the estimate $\hat{\pi}_s[t]$ can be written as a dot-product. Let $\tilde{\pi}_s$ be the empirical distribution over terminal nodes due to w random walks from s (with w chosen as in Theorem 1); we define the *forward vector* $x_s \in \mathbb{R}^{2n}$ to be the concatenation of the basis vector e_s and the random-walk terminal node distribution. On the other hand, we define the *reverse vector* $y^t \in \mathbb{R}^{2n}$, to be the concatenation of the estimates p^t and the residuals r^t. Formally, define

$$x_s = (e_s, \tilde{\pi}_s) \in \mathbb{R}^{2n}, \qquad y^t = (p^t, r^t) \in \mathbb{R}^{2n}. \tag{2}$$

Now, from Algorithm 2, we have

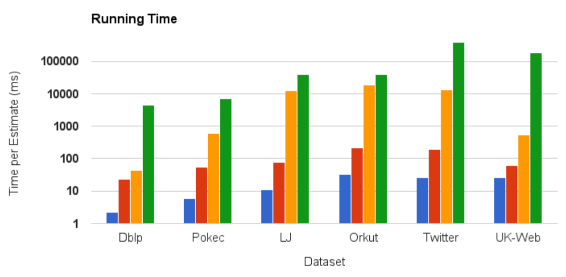
$$\hat{\pi}_s[t] = \langle x_s, y^t \rangle. \tag{3}$$

The above observation motivates the following algorithm:

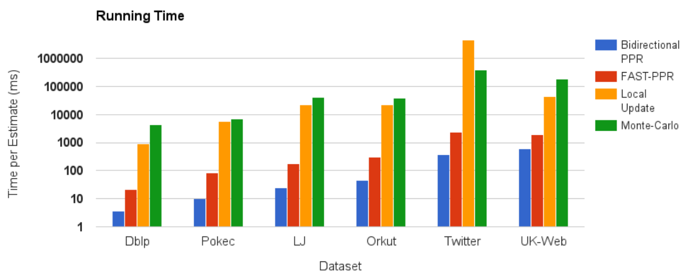
- `BiPPR-Precomp`: In this approach, we first use `Approx-Contributions` to pre-compute and store a reverse vector y^t for each $t \in V$. At query time, we generate random walks to form the forward vector x_s; now, given any set of targets T, we compute $|T|$ dot-products $\langle x_s, y^t\rangle$, and use these to rank the targets. This method now has an *worst-case* running time $O\left(|T|\sqrt{\bar{d}/\delta_k}\right)$. In practice, it works well if T is small, but is too slow for large T. In our experiments (doing 100,000 random walks at runtime) this approach takes around a second for $|T| \leq 30$, but this climbs to a minute for $|T| = 10,000$.

The `BiPPR-Precomp` approach is faster than `Bidirectional-PPR` (at the cost of additional precomputation and storage), and also faster than `Monte-Carlo` for small sets T, but it is still not efficient enough for real-time personalized search. This motivates us to find a more efficient algorithm that scales better than `Bidirectional-PPR` for large T, yet is fast for small $|T|$. In the following sections, we propose two different approaches for this – the first based on pre-grouping the precomputed reverse-vectors, and the second based on sampling target nodes from T according to PPR. For convenience, we first summarize the two approaches:

- `BiPPR-Precomp-Grouped`: Here, as in `BiPPR-Precomp`, we compute an estimate to each $t \in T$ using `Bidirectional-PPR`. However, we leverage the sparsity of the reverse vectors $y^t = (p^t, r^t)$ by first grouping them in a way we will describe. This makes the dot-product more efficient. This method has a *worst-case* running time of $O\left(|T|\sqrt{\bar{d}/\delta_k}\right)$, and in experiments we find it is much faster than `BiPPR-Precomp`. For our parameter choices its running time is less than 250ms across the range of $|T|$ we tried.
- `BiPPR-Precomp-Sampling`: We again first pre-compute the reverse vectors y^t. Next, for a given target t, we define the *expanded target-set* $T_t = \{v \in [2n] | y^t[v] \neq 0\}$, i.e., the set of nodes with non-zero reverse vectors from t. At



(a) Sampling targets uniformly



(b) Sampling targets from PageRank distribution

Figure 1: Average running-time (on log-scale) for different networks. We measure the time required for estimating PPR values $\pi_s(t)$ with threshold $\delta = \frac{4}{n}$ for 1000 (s,t) pairs. For each pair, the start node is sampled uniformly, while the target node is sampled uniformly in Figure 1(a), or from the global PageRank distribution in Figure 1(b). In this plot we use teleport probability $\alpha = 0.2$.

run-time, we now sample random walks forward from s to nodes in the expanded target sets. Using these, we create a sampler in average time $O(r_{\max}/\delta_k)$ (where as before δ_k is the k^{th} largest PPR value $\pi_s[t_k]$), which samples nodes $t \in T$ with probability proportional to the PPR $\pi_s[t]$. We describe this in detail in Section 4.2. Once the sampler has been created, it can be sampled in $O(1)$ time per sample. The algorithm works well for any size of T, and has the unique property that in can identify the top-k target nodes without computing a score for all $|T|$ of them. For our parameter choice its running time is less than 250ms across the range of $|T|$ we tried.

We note here that the case $k = 1$ (i.e., for finding the top PPR node) corresponds to solving a Maximum Inner Product Problem. In a recent line of work, Shrivastava and Li [21, 22] propose a sublinear time algorithm for this problem based on Locality Sensitive Hashing; however, their method assumes that there is some bound U on $\|y^t\|_2$ and that $\max_t\langle x_s, y^t\rangle$ is a large fraction of U. In personalized search, we usually encounter small values of $\max_t\langle x_s, y^t\rangle$ relative to $\max\|y^t\|_2$ – finding an LSH for Maximum Inner Product Search in this regime is an interesting open problem for future research. Our two approaches bypass this by exploiting particular structural features of the problem – `BiPPR-Precomp-Grouped` exploits the sparsity of the reverse vectors to speed up the dot-product, and `BiPPR-Precomp-Sampling` exploits the skewed distribution of PPR scores to find the top targets without even computing full dot-products.

4.1 Bidirectional-PPR with Grouping

In this method we improve the running-time of `BiPPR-Precomp` by pre-grouping the reverse vectors corresponding to each target set T. Recall that in `BiPPR-Precomp`, we first pre-compute reverse vectors $y^t = (p^t, r^t) \in \mathbb{R}^{2n}$ using `Approx-Contributions` for each t. At run-time, given s, we compute forward vector $x_s = (e_s, \tilde{\pi}_s)$ by generating sufficient random-walks, and then compute the scores $\langle x_s, y^t\rangle$ for $t \in T$. Our main observation is that *we can decrease the running time of the dot-products by pre-grouping the vectors y^t by coordinate.* The intuition behind this is that in each dot product $\sum_v x_s[v]y^t[v]$, the nodes v where $x_s[v] \neq 0$ often don't have $y^t[v] \neq 0$, and most of the product terms are 0. Hence, we can improve the running time by grouping the vectors y^t in advance by coordinate v. Now, at run-time, for each v such that $x_s[v] \neq 0$, we can efficiently iterate over the set of targets t such that $y^t[v] \neq 0$.

An alternative way to think about this is as a sparse matrix-vector multiplication $Y^T x_s$ after we form a matrix Y^T whose rows are y^t for $t \in T$. This optimization can then be seen as a sparse column representation of that matrix.

Algorithm 3 BiPPRGroupedPrecomputation$(T, r_{\max})$

```
Inputs: Graph $G$, teleport probability $\alpha$, target nodes $T$,
    maximum residual $r_{\max}$
 1: $z \leftarrow$ empty hash map of vectors such that for any $v$, $z[v]$
    defaults to an empty (sparse) vector in $\mathbb{R}^{2|V|}$
 2: for $t \in T$ do
 3:   Compute $y^t = (p_t, r_t) \in \mathbb{R}^{2|V|}$ via Approx-Contri-
      butions$(G, \alpha, t, r_{\max})$
 4:   for $v \in [2|V|]$ such that $y_t[v] > 0$ do
 5:     $z[v][t] = y^t[v]$
 6:   end for
 7: end for
 8: return $z$
```

Algorithm 4 BiPPRGroupedRankTargets$(s, r_{\max}, z)$

```
Inputs: Graph $G$, teleport probability $\alpha$, start node $s$, max-
    imum residual $r_{\max}$, $z$: hash map of reverse vectors
    grouped by coordinate
 1: Set number of walks $w = c\frac{r_{\max}}{\delta}$ (In experiments we
    found $c = 20$ achieved precision@3 above 90%.)
 2: Sample $w$ random-walks of length $Geometric(\alpha)$ from $s$;
    compute $\tilde{\pi}_s[v]$ = fraction of walks ending at node $v$
 3: Compute $x_s = (e_s, \tilde{\pi}_s) \in \mathbb{R}^{2|V|}$
 4: Initialize empty map score from $V$ to $\mathbb{R}$
 5: for $v$ such that $x_s[v] > 0$ do
 6:   for $t$ such that $z[v][t] > 0$ do
 7:     score$(t)$ += $x_s[v]z_v[t]$
 8:   end for
 9: end for
10: Return $T$ sorted in decreasing order of score
```

We refer to this method as `BiPPR-Precomp-Grouped`; the complete pseudo-code is given in Algorithm 4. The correctness of this method follows again from Theorem 1. In experiments, this method is efficient for T across all the sizes of T we tried, taking less than 250 ms even for $|T| = 10,000$. The improved running time of `BiPPR-Precomp-Grouped` comes at the cost of more storage compared to `BiPPR-Precomp`. In the case of name search, where each target typically only has a first and last name, each vector y^t only appears in two of these pre-grouped structures, so the storage is only twice the storage of `BiPPR-Precomp`. On the other hand if a target t contains many keywords, y^t will be included in many of these pre-grouped data structures, and storage cost will be significantly greater than for `BiPPR-Precomp`.

4.2 Sampling from Targets Matching a Query

The key idea behind this alternate method for PPR-search is that by sampling a target t in proportion to its PageRank we can quickly find the top targets without iterating over all of them. After drawing many samples, the targets can be ranked according to the number of times they are sampled. Alternatively a full `Bidirectional-PPR` query can be issued for some subset of the targets before ranking. This approach exploits the skewed distribution of PPR scores in order to find the top targets. In particular, prior empirical work has shown that on the Twitter graph, for each fixed s, the values $\pi_s[t]$ follow a power law [3].

We define the PPR-Search Sampling Problem as follows:

Given a source distribution s and a query q which filters the set of all targets to some list $T = \{t_i\} \subseteq V$, sample a target t_i with probability $p[t_i] = \frac{\pi_s[t_i]}{\sum_{t\in T}\pi_s[t]}$.

We develop two solutions to this sampling problem. The first, in $O(w) = O(r_{\max}/\delta_k)$ time, generates a data structure which can generate an arbitrary number of independent samples from a distribution which approximates the correct distribution. The second can generate samples from the exact distribution $\pi_s[t_i]$, and generates complete paths from s to some $t \in T$, but requires time $O(r_{\max}/\pi_s[T])$ per sample. Because the approximate sampler is more efficient, we present that here and defer the exact sampler to [17].

The `BiPPR-Precomp-Sampling` Algorithm

The high level idea behind our method is *hierarchical sampling*. Recall that the start node s has an associated forward vector $x_s = (e_s, \pi_s)$, and from each target t we have a reverse vector y^t; the PPR-estimate is given by $\pi_s[t] \approx \langle x_s, y^t\rangle$. Thus we want to sample $t \in T$ with probability:

$$p[t] = \frac{\langle x_s, y^t\rangle}{\sum_{j\in T}\langle x_s, y^j\rangle}.$$

We will sample t in two stages: first we sample an intermediate node $v \in V$ with probability:

$$p'_s[v] = \frac{x_s[v]\sum_{j\in T} y^j[v]}{\sum_{j\in T}\langle x_s, y^j\rangle}.$$

Following this, we sample $t \in T$ with probability:

$$p''_v[t] = \frac{y^t[v]}{\sum_{j\in T} y^j[v]}.$$

It is easy to verify that $p[t] = \sum_{v\in V} p'_s[v]p''_v[t]$. Figure 2 shows how the sampling algorithm works on an example graph. The pseudo-code is given in Algorithm 5 and Algorithm 6.

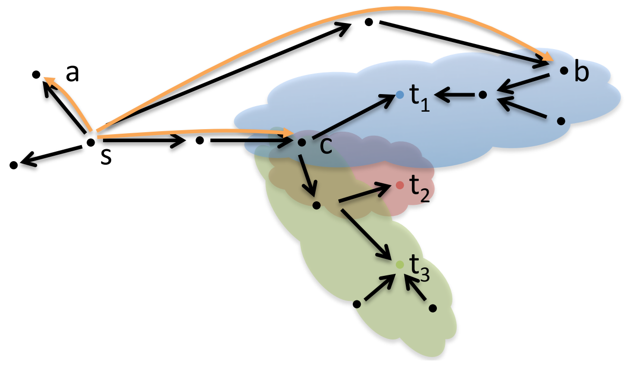


Figure 2: Search Example: Given target set $T = \{t_1, t_2, t_3\}$, for each target t_i we have drawn the expanded target-set, i.e., nodes v with positive residual $y^{t_i}[v]$. From source s, we sample three random walks, ending at nodes a, b, and c. Now suppose $y^{t_1}(b) = 0.64, y^{t_1}(c) = 0.4, y^{t_2}(c) = 0.16$, and $y^{t_3}(c) = 0.16$ – note that the remaining residuals are 0. Then we have $y^T(a) = 0, y^T(b) = 0.64$ and $y^T(c) = 0.72$, and consequently, the sampling weights of (a, b, c) are $(0, 0.213, 0.24)$. Now, to sample a target, we first sample from $\{a, b, c\}$ in proportion to its weight. Then if we sample b, we always return t_1; if we sample c, we sample (t_1, t_2, t_3) with probability $(5/9, 2/9, 2/9)$.

Note that we assume that some set of supported queries is known in advance, and we first pre-compute and store a separate data-structure for each query Q (i.e., for each target-set $T = \{t \in V : t \text{ is relevant to } Q\}$). In addition, we can optionally pre-compute w random walks from each start-node s, and store the forward vector x_s, or we can compute x_s at query time by sampling random walks.

Algorithm 5 SamplerPrecomputationForSet($T, r_{\max}$)

```
Inputs: Graph $G$, teleport probability $\alpha$, target-set $T$,
    maximum residual $r_{\max}$
1: for $t \in T$ do
2:     Compute $y^t = (p^t, r^t) \in \mathbb{R}^{2|V|}$ via Approx-Contri-
       butions($G, \alpha, t, r_{\max}$)
3: end for
4: Compute $y^T = \sum_{t\in T} y^t$
5: for $v \in V$ such that $y^T[v] > 0$ do
6:     Create sampler$_v$ which samples $t$ with probability
       $p''_v[t]$ (For example, using the alias sampling method
       [25], [15, section 3.4.1]).
7: end for
8: return ($y^T$, {sampler$_v$})
```

Running Time: For a small relative error for targets with $\pi_s[t] > \delta$, we use $w = cr_{\max}/\delta$ walks, where c is chosen as in Theorem 1. The support of our forward sampler is at most w so its construction time is $O(w)$ using the alias method of sampling from a discrete distribution [25], [15, section 3.4.1]. Once constructed, we can get independent samples in $O(1)$ time. Thus the query time to generate n_s samples is $O(cr_{\max}/\delta + n_s)$.

Accuracy: `BiPPR-Precomp-Sampling` does not sample exactly in proportion to π_s; instead, the sample probabilities

Algorithm 6 SampleAndRankTargets$(s, r_{\max}, y^T, \{\text{sampler}_v\})$

```
Inputs: Graph $G$, teleport probability $\alpha$, start node $s$, maximum residual $r_{\max}$, reverse vectors $y^T$, intermediate-node-to-target samplers $\{\text{sampler}_v\}$.
 1: Set number of walks $w = c\frac{r_{\max}}{\delta}$. In experiments we found $c = 20$ achieved precision@5 above 90% on Twitter-2010.
 2: Set number of samples $n_s$ (We use $n_s = w$)
 3: Sample $w$ random walks from $s$ and let $\tilde{\pi}_s$ be the empirical distribution of their endpoints; compute forward vector $x_s = (e_s, \tilde{\pi}_s) \in \mathbb{R}^{2|V|}$
 4: Create sampler$_s$ to sample $v \in [2|V|]$ with probability $p'_s[v]$, i.e., proportional to $x_s[v]y^T[v]$
 5: Initialize empty map score from $V$ to $\mathbb{N}$
 6: for $j \in [0, n_s - 1]$ do
 7:     Sample $v$ from sampler$_s$
 8:     Sample $t$ from sampler$_v$
 9:     Increment score$(t)$
10: end for
11: Return $T$ sorted in decreasing order of score
```

are proportional to a distribution $\hat{\pi}_s$ satisfying the guarantee of Theorem 1. In particular, for all targets t with $\pi_s[t] \geq \delta$, this will have a small relative error ϵ, while targets with $\pi_s[t] < \delta$ will likely be sampled rarely enough that they won't appear in the set of top-k most sampled nodes.

Storage Required: The storage requirements for `BiPPR-Precomp-Sampling` (and for `BiPPR-Precomp-Grouped`) depends on the distribution of keywords and how $r_{\max}$ is chosen for each target set. For simplicity, here we assume a single maximum residual $r_{\max}$ across all target sets, and assume each target is relevant to at most γ keywords. For example, in the case of name search, each user typically has a first and last name, so $\gamma = 2$.

THEOREM 3. *Let graph G, minimum-PPR value δ and time-space trade-off parameter r_{max} be given, and suppose every node contains at most γ keywords. Then the total storage needed for `BiPPR-Precomp-Sampling` to construct a sampler for any source node (or distribution) s and any set of targets T corresponding to a single keyword is $O\left(\frac{\gamma m}{\alpha r_{max}}\right)$.*

We can choose $r_{\max}$ to trade-off this storage requirement with the running time requirement of $O(cr_{\max}/\delta)$ – for example, we can set both the query running-time and per-node storage to $\sqrt{c\gamma\bar{d}/\delta}$ where $\bar{d} = m/n$ is the average degree. Now for name search $\gamma = 2$, and if we choose $\delta = \frac{1}{n}$ and $\alpha = \Theta(1)$, the per-query running time and per-node storage is $O(\sqrt{m})$.

PROOF. For each set T corresponding to a keyword, and each $t \in T$, we push from nodes v until for each v, $r^t[v] < r_{\max}$. Each time we push from a node v, we add an entry to the residual vector of each node $u \in \mathcal{N}^{in}(v)$, so the space cost is $d^{in}(v)$. Each time we push from a node v, we increase the estimate $p^t[v]$ by $\alpha r^t[v] \geq \alpha r_{\max}$, and $\sum_{t \in T} p^t[v] \leq \sum_{t \in T} \pi_v[t] = \pi_v[T]$ so v can be pushed from at most $\frac{\pi_v[t]}{\alpha r_{\max}}$ times. Thus the total storage required is

$$\sum_{v \in V} d^{in}(v)(\text{\# of times } v \text{ pushed}) \leq \sum_{v \in V} d^{in}(v) \frac{\pi_v[T]}{\alpha r_{\max}} \quad (4)$$

Let $\mathcal{T}$ be the set of all target sets (one target set per keyword). Then the total storage over all keywords is

$$\sum_{T \in \mathcal{T}} \sum_{t \in T} \sum_{v \in V} d^{in}(v) \frac{\pi_v[t]}{\alpha r_{\max}} \leq \gamma \sum_{v \in V} \sum_{t \in V} d^{in}(v) \frac{\pi_v[t]}{\alpha r_{\max}}$$
$$\leq \gamma \sum_{v \in V} d^{in}(v) \frac{1}{\alpha r_{\max}} \leq \gamma \frac{m}{\alpha r_{\max}}.$$

□

Adaptive Maximum Residual: One way to improve the storage requirement is by using larger values of $r_{\max}$ for target sets T with larger global PageRank. Intuitively, if T is large, then it's easier for random walks to get close to T, so we don't need to push back from T as much as we would for a small T. We now formalize this scheme, and outline the savings in storage via a heuristic analysis, based on a model of personalized PageRank values introduced by Bahmani et al. [3].

For a fixed s, we assume the values $\pi_s[v]$ for all $v \in V$ approximately follow a power law with exponent β. Empirically, this is known to be an accurate model for the Twitter graph – Bahmani et al. [3] find that the mean exponent for a user is $\beta = 0.77$ with standard deviation 0.08. To analyze our algorithm, we further assume that π_s restricted to T also follows a power law, i.e.:

$$\pi_s[t_i] = \frac{1-\beta}{|T|^{1-\beta}} i^{-\beta} \pi_s[T]. \quad (5)$$

Suppose we want an accurate estimate of $\pi_s[t_i]$ for the top-k results within T, so we set $\delta_k = \pi_s[t_k]$. From Theorem 1, the number of walks required is:

$$w = c\frac{r_{\max}}{\delta_k} = c_2 \frac{r_{\max}|T|^{1-\beta}}{\pi_s[T]}$$

where $c_2 = k^\beta c/(1-\beta)$. If we fix the number of walks as w, then we must set $r_{\max} = w\pi_s[T]/(c_2|T|^{1-\beta})$. Also, for a uniformly random start node s, we have $\mathbb{E}[\pi_s[T]] = \pi[T]$ (the global PageRank of T). This suggests we choose $r_{\max}(T)$ for set T as:

$$r_{\max}(T) = \frac{w\pi[T]}{c_2|T|^{1-\beta}} \quad (6)$$

Going back to equation (4), suppose for simplicity that the average $d^{in}(v)$ encountered is $\bar{d}$. Then the storage required for this keyword is bounded by:

$$\sum_{v \in V} d^{in}(v) \frac{\pi_v[T]}{r_{\max}} = \bar{d}\frac{n\pi[T]}{r_{\max}} = \frac{mc_2|T|^{1-\beta}}{w}.$$

Note that this is independent of $\pi[T]$. There is still a dependence on $|T|$, which is natural since for larger T there are more nodes which make it harder to find the top-k. For $\beta = 0.77$, the rate of growth, $|T|^{0.23}$ is fairly small, and in particular is sublinear in $|T|$.

Dynamic Graphs: So far we have assumed that the graph and keywords are static, but in practice they change over time. When a keyword is added to some node T, the node's reverse vector y^t needs to be added to the sampling data structure for that keyword. When an edge is added, the residual values need to be updated. We leave the extension to dynamic graphs to future work.

4.3 Experiments

We conduct experiments to test the efficiency of these personalized search algorithms as the size of the target set varies. We use one of the largest publicly available social networks, Twitter-2010 [16] with 40 million nodes and 1.5 billion edges. For various values of $|T|$, we select a target set T uniformly among all sets with that size, and compare the running times of the four algorithms we propose in this work, as well as the `Monte Carlo` algorithm. We repeat this using 10 random target sets and 10 random sources s per target set, and report the median running time for all algorithms. We use the same target sets and sources for all algorithms.

Parameter Choices: Because all five algorithms have parameters that trade-off running time and accuracy, we choose parameters such that the accuracy is comparable so we can compare running time on a level playing field. To choose a concrete benchmark, we chose parameters such that the precision@3 of the four algorithms we propose are consistently greater than 90% for the range of $|T|$ we used in experiment. We chose parameters for `Monte-Carlo` so that our algorithms are consistently more accurate than it, and its precision@3 is greater than 85%. In the full version we plot the precision@3 of the algorithms for the parameters we use when comparing running time.

We used $\delta = \pi_s(t_k)$ where $\pi_s(t_k)$ is estimated using Eqn. 5, using $k = 3$, power law exponent $\beta = 0.77$ (the mean value found empirically on Twitter), and assuming $\pi_s(T) = \frac{|T|}{n}$ (the expected value of $\pi_s(T)$ since T is chosen uniformly at random). Then we use Equation 6 to set $r_{\max}$, using $c = 20$ and two values of w, 10,000 and 100,000. We used the same value of $r_{\max}$ for `BiPPR-Precomp`, `BiPPR-Precomp-Grouped`, and `BiPPR-Precomp-Sampling`. For `Monte-Carlo`, we sampled $\frac{40}{\delta}$ walks[1].

Results: Figure 3 shows the running time of the five algorithms as $|T|$ varies for two different parameter settings in the trade-off between running time and precomputed storage requirement. Notice that `Monte-Carlo` is very slow on this large graph for small target set sizes, but gets faster as the size of the target set increases. For example when $|T| = 10$ Monte Carlo takes half an hour, and even for $|T| = 1000$ it takes more than a minute. `Bidirectional-PPR` is fast for small T, but slow for larger T, taking more than a second when $|T| \geq 100$. In contrast, `BiPPR-Precomp-Grouped` and `BiPPR-Precomp-Sampling` are both fast for all sizes of T, taking less than 250 ms when $w = 10,000$ and less than 25 ms when $w = 100,000$.

The improved running time of `BiPPR-Precomp-Grouped` and `BiPPR-Precomp-Sampling`, however, comes at the cost of pre-computation and storage. With these parameter choices, for $w = 10,000$ the pre-computation size per target set in our experiments ranged from 8 MB (for $|T| = 10$) to 200MB (for $|T| = 1000$) per keyword. For $w = 100,000$, the storage per keyword ranges from 3 MB (for $|T| = 10$) to 30MB (for $|T| = 10,000$).

[1]Note that `Monte-Carlo` was too slow to finish in a reasonable amount of time, so we measured the average time required to take 10 million walks, then multiplied by the number of walks needed. When measuring precision, we simulated the target weights `Monte-Carlo` would generate, by sampling t_i with probability $\pi_s(t_i)$; this produces exactly the same distribution of weights as `Monte-Carlo` would.

To get a larger range of $|T|$ relative to $|V|$, we also perform experiments on the Pokec graph [23] which has 1.6 million nodes and 30 million edges. Figure 4 shows the results on Pokec for $w = 100,000$. Here we clearly see the cross-over point where `Monte-Carlo` becomes more efficient than `Bidirectional-PPR`, while `BiPPR-Precomp-Grouped` and `BiPPR-Precomp-Sampling` consistently take less than 250 milliseconds. On Pokec, the storage used ranges from 800KB for $|T| = 10$ to 3MB for $|T| = 10,000$.

We implement our algorithms in Scala and report running times for Scala, but in preliminary experiments `BiPPR-Precomp-Grouped` is 3x faster when re-implemented in C++, we expect the running time would improve comparably for all five algorithms. Also, we ran each experiment on a single thread, but the algorithms parallelize naturally, so the latency could be improved by a multi-threaded implementation. We ran our experiments on a machine with a 3.33 GHz 12-core Intel Xeon X5680 processor, 12MB cache, and 192 GB of 1066 MHz Registered ECC DDR3 RAM. We measured the running time of the tread running each experiment to exclude garbage collector time. We loaded the graph used into memory and completed any pre-computation in RAM before measuring the running time of the algorithms.

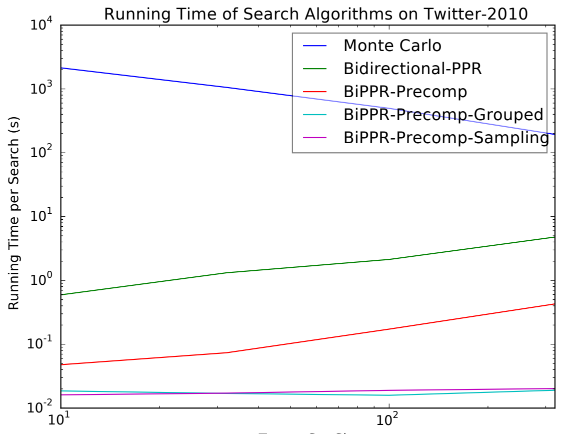
5. RELATED WORK

Prior work on PPR Estimation The `Bidirectional-PPR` algorithm introduced in the first half of this work builds on the `FAST-PPR` algorithm presented in [19] – for details of prior work on Personalized PageRank estimation, see the section on existing approaches in [19]. Although `FAST-PPR` was the first algorithm for PPR estimation with sublinear running-time guarantees, it has several drawbacks which are improved upon by our new `Bidirectional-PPR` algorithm:

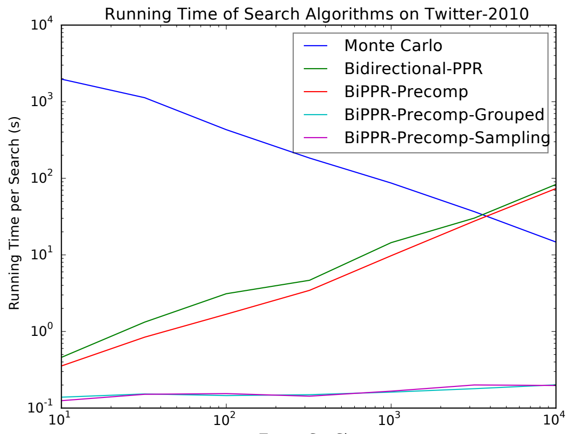
- `Bidirectional-PPR` has a simple linear structure which enables searching; Eqn. 3 shows that the estimates are a dot-produce between a forward vector x^s and a reverse vector y^t. In contrast, estimates in [19] require monitoring each walk to detect collisions with a "frontier" set.
- `Bidirectional-PPR` is 3x-8x faster than `FAST-PPR` for the same accuracy in experiments on diverse networks.
- `Bidirectional-PPR` is cleaner and more elegant, leading to simpler correctness proofs and performance analysis. This also makes it easier to generalize to arbitrary Markov Chains, as done in [6].

Comparison to Partitioned Multi-Indexing For personalized search, our indexing scheme is partially inspired by the Partitioned Multi-Indexing (PMI) scheme of Bahmani et al. [4]. Similar to our methods, PMI uses a bidirectional approach to rank search results according to shortest path distance from the searching user. Shortest path is easier to estimate than PPR, due to the fact that shortest path is a metric; moreover, shortest path is believed to be a less effective way of ranking search results than PPR. From a technical point of view, PMI is based on 'sweeping' from closer to more distant targets based on a distance oracle; in contrast, we use sampling to find the most relevant targets.

Prior work on Personalized PageRank Search In [7], Berkhin builds upon the previous work [14] and proposes efficient ways to compute the personalized PageRank vector π_s at runtime by combining pre-computed PPR vectors in a query-specific way. In particular, they identify "hub" nodes in advance, using heuristics such as global PageRank, and precompute approximate PPR vectors $\hat{\pi}_h$ for each hub



(a) Running Time, More Precomputation



(b) Running Time, Less Precomputation

Figure 3: Running time on Twitter-2010 (1.5 billion edges) on log-scale, with parameters chosen such that the Precision@3 of our algorithms exceeds 90% and exceeds the precision@3 of `Monte-Carlo`. The two plots demonstrate the storage-runtime tradeoff: Figure 3(a) (which performs $10K$ walks at runtime) uses more pre-computation and storage compared to Figure 3(b) (with $100K$ walks).

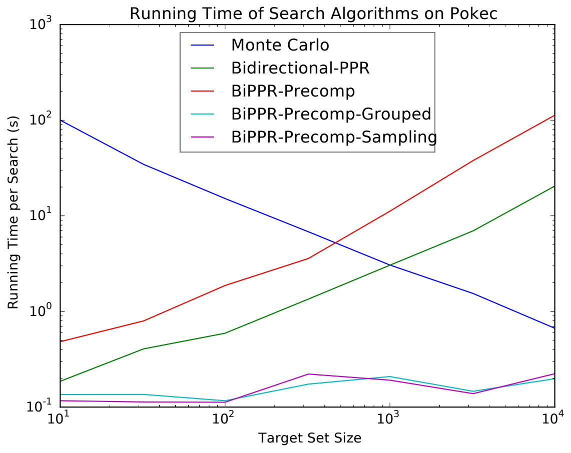


Figure 4: Running time on Pokec (30 million edges) performing 100K walks at runtime. Notice that `Monte-Carlo` is slow for small $|T|$, `Bidirectional-PPR` is slow for large $|T|$, and `BiPPR-Precomp-Grouped` and `BiPPR-Precomp-Sampling` are fast across the entire range of $|T|$.

node using a local forward-push algorithm called the Bookmark Coloring Algorithm (BCA). Chakrabarti [8] proposes a variant of this approach, where Monte-Carlo is used to pre-compute the hub vectors $\hat{\pi}_h$ rather than BCA.

Both approaches differ from our work in that they construct complete approximations to π_s, then pick out entries relevant to the query. This requires a high-accuracy estimate for π_s even though only a few entries are important. In contrast, our bidirectional approach allows us compute only the entries $\pi_s(t_i)$ relevant to the query.

6. ACKNOWLEDGMENTS

Research supported by the DARPA GRAPHS program via grant FA9550-12-1-0411, and by NSF grant 1447697. One author was supported by an NPSC fellowship.

7. REFERENCES

[1] R. Andersen, C. Borgs, J. Chayes, J. Hopcraft, V. S. Mirrokni, and S.-H. Teng. Local computation of pagerank contributions. In *Algorithms and Models for the Web-Graph*. Springer, 2007.

[2] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM Journal on Numerical Analysis*, 2007.

[3] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 2010.

[4] B. Bahmani and A. Goel. Partitioned multi-indexing: bringing order to social search. In *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012.

[5] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceedings of the*

17th international conference on World Wide Web, pages 895–904. ACM, 2008.

[6] S. Banerjee and P. Lofgren. Fast bidirectional probability estimation in markov models. In *Advances in Neural Information Processing Systems*, pages 1423–1431, 2015.

[7] P. Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics*, 3(1):41–62, 2006.

[8] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proceedings of the 16th international conference on World Wide Web*, pages 571–580. ACM, 2007.

[9] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2005.

[10] D. F. Gleich. PageRank beyond the web. *arXiv*, cs.SI:1407.5107, 2014. Accepted for publication in SIAM Review.

[11] A. Goel, P. Gupta, and P. Lofgren. In preparation: Cross partitioning: Realtime computation of cosine similarity, personalized pagerank, and more. Technical report, Stanford University, 2015. available at http://www.stanford.edu/~plofgren/.

[12] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2013.

[13] T. H. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, pages 517–526. ACM, 2002.

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

[15] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd Edition*. Reading, Mass.: Addison-Wesley, 1998.

[16] Laboratory for web algorithmics. http://law.di.unimi.it/datasets.php. Accessed: 2014-02-11.

[17] P. Lofgren. *Efficient Algorithms for Personalized PageRank*. PhD thesis, Stanford University, 2015. available at http://cs.stanford.edu/people/plofgren/.

[18] P. Lofgren and A. Goel. Personalized pagerank to a target node. *arXiv preprint arXiv:1304.4658*, 2013.

[19] P. A. Lofgren, S. Banerjee, A. Goel, and C. Seshadhri. Fast-ppr: Scaling personalized pagerank estimation for large graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1436–1445, New York, NY, USA, 2014. ACM.

[20] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[21] A. Shrivastava and P. Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pages 2321–2329, 2014.

[22] A. Shrivastava and P. Li. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). *stat*, 1050:13, 2014.

[23] Stanford network analysis platform (snap). http://http://snap.stanford.edu/. Accessed: 2014-02-11.

[24] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. ACM, 2007.

[25] A. J. Walker. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw.*, 3(3):253–256, Sept. 1977.