

Local Approximation of PageRank and Reverse PageRank *

Ziv Bar-Yossef
Department of Electrical Engineering
Technion, Haifa, Israel
and
Google Haifa Engineering Center, Haifa, Israel
zivby@ee.technion.ac.il

Li-Tal Mashiach
Department of Computer Science
Technion, Haifa, Israel
litalma@cs.technion.ac.il

ABSTRACT

We consider the problem of approximating the PageRank of a target node using only local information provided by a link server. This problem was originally studied by Chen, Gan, and Suel (CIKM 2004), who presented an algorithm for tackling it. We prove that local approximation of PageRank, even to within modest approximation factors, is infeasible in the worst-case, as it requires probing the link server for $\Omega(n)$ nodes, where n is the size of the graph. The difficulty emanates from nodes of high in-degree and/or from slow convergence of the PageRank random walk.

We show that when the graph has bounded in-degree and admits fast PageRank convergence, then local PageRank approximation can be done using a small number of queries. Unfortunately, natural graphs, such as the web graph, are abundant with high in-degree nodes, making this algorithm (or any other local approximation algorithm) too costly. On the other hand, *reverse* natural graphs tend to have low in-degree while maintaining fast PageRank convergence. It follows that calculating *Reverse PageRank* locally is frequently more feasible than computing PageRank locally.

We demonstrate that Reverse PageRank is useful for several applications, including computation of hub scores for web pages, finding influencers in social networks, obtaining good seeds for crawling, and measurement of semantic relatedness between concepts in a taxonomy.

1. INTRODUCTION

Over the past decade PageRank [28] has become one of the most popular methods for ranking nodes by their “prominence” in a network.¹ PageRank’s underlying idea is simple but powerful: a prominent node is one that is “supported” (linked to) by other prominent nodes. PageRank was originally introduced as means for rank-

* A short version of this paper is to appear as a poster at SIGIR 2008.

This work was supported by the European Commission Marie Curie International Re-integration Grant, by the Israel Science Foundation and by IBM faculty award.

¹According to Google Scholar (<http://scholar.google.com>), as of April 2008 the PageRank paper has 1,973 citations.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

ing web pages in search results. Since then it has found uses in many other domains, such as measuring centrality in social networks [19], evaluating the importance of scientific publications, prioritizing pages in a crawler’s frontier [9], personalizing search results [7], combating spam [16], measuring trust, selecting pages for indexing, and more. While the significance of PageRank in ranking search results seems to have diminished, due to the emergence of other effective alternatives (e.g., clickthrough-based measures), it is still an important tool in search infrastructure, social networks, and analysis of large graphs.

Local PageRank approximation. The vast majority of algorithms for computing PageRank, whether they are centralized, parallel, or decentralized, have focused on *global* computation of the PageRank vector. That is, PageRank scores for *all* the graph’s nodes are computed. While in many applications of PageRank a global computation is needed, there are situations in which one is interested in computing PageRank scores for just a small subset of the nodes.

Consider, for instance, a web site owner (e.g. a small or a large business), who would like to promote the web site in search engine rankings in order to attract traffic of potential clients. As PageRank is used by search engines to determine whether to crawl/index pages and to calculate their relevance scores, tracking the PageRank of the web site would enable the web site owner to better understand its position in search engine rankings and potentially take actions to improve the web site’s PageRank. In this case, the web site owner is interested only in the PageRank score of his own web site (and maybe also in the scores of his competitors’ web sites), but not in the PageRank scores of all other web pages.

Major search engines choose to keep the PageRank scores of web pages confidential, since there are many variations of the PageRank formula, and making the exact PageRank values public may enable spammers to promote illegitimate web sites. Some search engines publish crude PageRank values (e.g., through the Google Toolbar), but these are usually given in a 1 to 10 logarithmic scale. Users who wish to obtain more accurate PageRank scores for pages of their choice are left to compute them on their own. Global PageRank computation for the entire web graph is out of the question for most users, as it requires significant resources and knowhow. This brings up the following natural question: can one compute the PageRank score of a single web page using reasonable resources?

The same question arises in other contexts, where PageRank is used. For example, a Facebook² user may be interested in measuring her PageRank popularity by probing the friendship graph. Can this be done efficiently without traversing the whole network?

Chen *et al.* [8] were the first to introduce the problem of *local PageRank approximation*. Suppose we are given access to a large

²<http://www.facebook.com/>.

graph G through a *link server*, which for every given query node x , returns the edges incident to x (both incoming and outgoing).³ Can we use a small number of queries to the link server to approximate the PageRank score of a target node x to within high precision?

Chen *et al.* proposed an algorithm for solving this problem. Their algorithm crawls backwards a small subgraph around the target node, applies various heuristics to guess the PageRank scores of the nodes at the boundary of this subgraph, and then computes the PageRank of the target node within this subgraph. Chen *et al.* empirically showed this algorithm to provide good approximations on average. However, they noted that high in-degree nodes sometimes make the algorithm either very expensive or inaccurate.

Lower bounds. We study the limits of local PageRank approximation. We identify two factors that make local PageRank approximation hard on certain graphs: (1) the existence of high in-degree nodes; (2) slow convergence of the PageRank random walk.⁴

In order to demonstrate the effect of high in-degree nodes, we exhibit for every n a family of graphs of size n whose maximum in-degree is high ($\Omega(n)$) and on which any algorithm would need to send $\Omega(\sqrt{n})$ queries to the link server in order to obtain accurate PageRank approximations. For very large n , fetching \sqrt{n} pages from the network or sending \sqrt{n} queries to a search engine is very costly (for example, for the web graph $n \geq 10B$, and thus $\sqrt{n} \geq 128K$). The lower bound we prove applies to both randomized and deterministic algorithms. For deterministic algorithms, we are able to prove an even stronger (and optimal) $\Omega(n)$ lower bound.

Similarly, to demonstrate the effect of slow PageRank convergence, we present a family of graphs on which the PageRank random walk converges rather slowly (in $\Omega(\log n)$ steps) and on which every algorithm needs to submit $\Omega(n^{\frac{1}{2}-\epsilon})$ queries in order to obtain good PageRank approximations ($\epsilon > 0$ is a small constant that depends on the PageRank damping factor). Again, this lower bound holds for both randomized and deterministic algorithms. For deterministic algorithms, we show an optimal $\Omega(n)$ lower bound.

We note that the two lower bounds do not subsume each other, as the family of hard graphs constructed in the first bound has very fast PageRank convergence (2 iterations), while the family of hard graphs constructed in the second bound has bounded in-degree (2).

Sufficiency. Having proved that local PageRank approximation is hard for graphs that have high in-degree or do not admit quick PageRank convergence, it is natural to ask whether local PageRank approximation is feasible for graphs of bounded in-degree and on which PageRank converges quickly. We observe that a variation of the algorithm of Chen *et al.* works well for such graphs: if the PageRank random walk converges on the graph in r steps and if the maximum in-degree of the graph is d , then the algorithm crawls a subgraph of size at most d^r and thus requires at most this number of queries to the link server. When d and r are small, the algorithm is efficient. This demonstrates that the conditions we showed to be necessary for fast local PageRank approximation are also sufficient.

PageRank vs. Reverse PageRank. As natural graphs, like the web graph and social networks, are abundant with high in-degree nodes, our first lower bound suggests that local PageRank approximation is frequently infeasible on such graphs. We substantiate

³If G is the web graph, out-links can be extracted from the content of x itself and in-links can be retrieved from search engines using the `link:` query. As opposed to PageRank scores, in-links are information that search engines are willing to disclose.

⁴Note that the convergence rate of PageRank on a given graph is an intrinsic property of the graph, not of the particular algorithm used to compute PageRank. The convergence rate is governed by the difference between the first and the second eigenvalues of PageRank’s transition matrix. See [18] for more details.

this observation with an empirical analysis of a 280,000 crawl of the `www.stanford.edu` site. We show that locally approximating PageRank is especially difficult for the high PageRank nodes. These findings provide analytical and empirical explanations for the difficulties encountered by Chen *et al.*

We then demonstrate that *reverse* natural graphs (the graphs obtained by reversing the directions of all links) are more suitable for local PageRank approximation. By analyzing the `stanford.edu` crawl, we show that the reverse web graph, like the web graph, admits quick PageRank convergence (on 80% nodes of the reverse graph, PageRank converged within 20 iterations). We also show that the reverse graph has low in-degree (only 255 as opposed to 38,606 in the regular graph). These findings hint that local PageRank approximation should be feasible on the reverse graph.

To put this hypothesis to test, we measured the performance of our variation of the Chen *et al.* algorithm on a sample of nodes from the `stanford.edu` graph. We show that for highly ranked nodes the performance of the algorithm on the reverse graph is up to three times better than on the regular graph.

We conclude from the above that the reverse web graph is much more amenable to efficient local PageRank approximation than the regular web graph. Thus, computing *Reverse PageRank* (PageRank of the reverse graph; “RPR” in short) is more feasible to do locally than computing regular PageRank. Social networks and other natural graphs possess similar properties to the web graph (power law degrees, high in-degree vs. low out-degree) and are thus expected to exhibit similar behavior.

Applications of Reverse PageRank. While locally approximating RPR is easier than locally approximating PageRank, why would one want to compute PR in the first place? We observe that RPR has a multitude of applications: it has been used to select good seeds for the TrustRank measure [16], to detect highly influential nodes in social networks [19], and to find hubs in the web graph [14]. We present two additional novel applications of RPR: (1) finding good seeds for crawling; and (2) measuring the “semantic relatedness” of concepts in a taxonomy. Local approximation could be beneficial in 3 of the RPR applications: influencers detection, hub score computation, and measuring of semantic relatedness.

2. RELATED WORK

There is a large body of work on PageRank computation, varying from centralized algorithms (e.g., [21, 23, 4]), to parallel algorithms (e.g., [25, 24]), to decentralized P2P algorithms (e.g., [35, 29]). All of these are designed to compute the whole PageRank vector and are thus not directly applicable to our problem. See a survey by Berkhin [3] for an extensive review of PageRank computation techniques.

Apart from Chen *et al.*, also Davis and Dhillon [11] and Wu [36] consider computations of global PageRank values over subgraphs. These two works, however, do not rely on a link server and thus work in a different model than what we consider in this paper.

3. PRELIMINARIES

PageRank overview. Let $G = (V, E)$ be a directed graph on n nodes. Let M be the $n \times n$ probability transition matrix of the simple random walk on G :

$$M(u, v) = \begin{cases} \frac{1}{\text{outdeg}(u)}, & \text{if } u \rightarrow v \text{ is an edge,} \\ 0, & \text{otherwise.} \end{cases}$$

Let U be the probability transition matrix of the uniform random walk, in which at each step a node is chosen uniformly at random

independently of the history: $U(u, v) = \frac{1}{n}$.

Given a *damping factor* $0 \leq \alpha \leq 1$, PageRank [28] (denoted $\text{PR}^G(\cdot)$) is defined as the limit distribution of the random walk induced by the following convex combination of \mathbf{M} and \mathbf{U} :

$$\mathbf{P} = \alpha\mathbf{M} + (1 - \alpha)\mathbf{U}.$$

$\alpha = 0.85$ is a typical choice, and we use it in our experiments. *Personalized PageRank* [17, 20] is a popular generalization of PageRank, in which the uniform distribution in \mathbf{U} is replaced by a different, “personalized”, distribution. Everything we do in this paper can be rather easily generalized to the personalized case. For simplicity of exposition we choose to stick to the uniform case.

Local PageRank approximation. A local algorithm working on an input graph G is given access to G only through a “link server”. Given an id of a node $u \in V$, the server returns the IDs of u ’s neighbors in G (both in-neighbors and out-neighbors).

DEFINITION 1. An algorithm is said to locally approximate PageRank, if for any graph $G = (V, E)$, for which it has local access, any target node $u \in V$, and any error parameter $\epsilon > 0$, the algorithm outputs a value $\overline{\text{PR}}(u)$ satisfying:

$$(1 - \epsilon)\text{PR}^G(u) \leq \overline{\text{PR}}(u) \leq (1 + \epsilon)\text{PR}^G(u).$$

If the algorithm is randomized, it is required to output, for any inputs G, u, ϵ , a $1 \pm \epsilon$ approximation of $\text{PR}^G(u)$ with probability at least $1 - \delta$, where $0 < \delta < 1$ is the algorithm’s confidence parameter. The probability is over the algorithm’s internal coin tosses.

We measure the performance of such algorithms in terms of their *query cost*, which is the number of queries they send to the link server for the worst-case choice of graph G and target node u . Typically, the actual resources used by these algorithms (time, space, bandwidth) are proportional to their query cost. We will view polylogarithmic cost ($O(\log^{O(1)}(n))$) as feasible and polynomial cost ($\Omega(n^{1-\epsilon})$ for some $\epsilon > 0$) as infeasible.

PageRank and influence. Jeh and Widom [20] provide a useful characterization of PageRank in term of the notion of “influence”. We present a different variation of influence, which divides the influence of a node into layers. This will make the analysis easier.

The *influence* [8] of a node $v \in G$ on the PageRank of $u \in G$ is the fraction of the PageRank score of v that flows into u , excluding the effect of decay due to the damping factor α :

DEFINITION 2. For a path $p = (u_0, u_1, \dots, u_t)$, define

$$\text{weight}(p) = \prod_{i=0}^{t-1} \frac{1}{\text{outdeg}(u_i)}.$$

Let $\text{paths}_t(v, u)$ be the set of all paths of length t from v to u . The influence of v on u at radius t is:

$$\text{inf}_t(v, u) = \sum_{p \in \text{paths}_t(v, u)} \text{weight}(p).$$

(For $t = 0$, we define $\text{inf}_0(u, u) = 1$ and $\text{inf}_0(v, u) = 0$, for all $v \neq u$.) The total influence of v on u is:

$$\text{inf}(v, u) = \sum_{t=0}^{\infty} \text{inf}_t(v, u).$$

Note that the same node v may have influence on u at several different radii. Using influence, we define the *PageRank value of u at radius r* to be: $\text{PR}_r^G(u) = \frac{1-\alpha}{n} \sum_{t=0}^r \sum_{v \in G} \alpha^t \text{inf}_t(v, u)$. $\text{PR}_r^G(u)$

represents the cumulative PageRank score that flows into u from nodes at distance at most r from u . We show below a characterization of PageRank in terms of influence, which is similar to the one appearing in the work of Jeh and Widom [20].

THEOREM 3. For every node $u \in G$, $\text{PR}^G(u) = \lim_{r \rightarrow \infty} \text{PR}_r^G(u)$.

PROOF. First, let us express PR as a power series in terms of \mathbf{P} and \mathbf{M} :

LEMMA 4. For every $r \geq 1$,

$$\text{PR}_{r-1}^G(u) = \mathbf{P}^r(1, u) - \alpha^r \mathbf{M}^r(1, u).$$

The proof of the lemma can be found in the full version of the paper⁵. Assuming the correctness of the lemma:

$$\begin{aligned} \lim_{r \rightarrow \infty} \text{PR}_r^G(u) &= \lim_{r \rightarrow \infty} (\mathbf{P}^{r+1}(1, u) - \alpha^{r+1} \mathbf{M}^{r+1}(1, u)) \\ &= \lim_{r \rightarrow \infty} \mathbf{P}^{r+1}(1, u) - \lim_{r \rightarrow \infty} \alpha^{r+1} \mathbf{M}^{r+1}(1, u). \end{aligned}$$

As \mathbf{M}^{r+1} is a probability transition matrix, $\mathbf{M}^{r+1}(1, u) \leq 1$, and thus $\alpha^{r+1} \mathbf{M}^{r+1} < 1$. Therefore, as $r \rightarrow \infty$, $\alpha^{r+1} \mathbf{M}^{r+1}(1, u) \rightarrow 0$. This means that: $\lim_{r \rightarrow \infty} \text{PR}_r^G(u) = \lim_{r \rightarrow \infty} \mathbf{P}^{r+1}(1, u)$.

Now Consider the initial distribution $\mathbf{p}_0 = (1, 0, \dots, 0)$, and let $\mathbf{p}_r = \mathbf{p}_0 \mathbf{P}^r$. Recall that $\lim_{r \rightarrow \infty} (\mathbf{p}_r) = \text{PR}^G$. In particular, $\lim_{r \rightarrow \infty} (\mathbf{p}_r(u)) = \text{PR}^G(u)$. As $\mathbf{p}_r(u) = \mathbf{P}^r(1, u)$, we conclude that: $\lim_{r \rightarrow \infty} \text{PR}_r^G(u) = \text{PR}^G(u)$. \square

Note that $\text{PR}_r^G(u)$ approaches $\text{PR}^G(u)$ from below. Throughout this paper, we will use the following notion of influence convergence rate, which is reminiscent of the standard mixing time [33] of Markov Chains:

DEFINITION 5. For a graph G , a target node u , and an approximation parameter $\epsilon > 0$, define the pointwise influence mixing time as:

$$T_\epsilon(G, u) = \min\{r \geq 0 \mid \frac{\text{PR}_r^G(u) - \text{PR}^G(u)}{\text{PR}^G(u)} < \epsilon\}.$$

The standard convergence rate of PageRank is defined as the rate at which the rows of the matrix \mathbf{P}^r approach the PageRank vector as $r \rightarrow \infty$. Lemma 4 implies that the difference from the above notion of mixing time is at most $O(\log(1/\epsilon))$ and thus the two notions are essentially equivalent. Therefore, for the rest of the paper when we say that “the PageRank random walk converges in r iterations on a node u ”, we will actually mean that $T_\epsilon(G, u) \leq r$.

4. LOWER BOUNDS

In this section we present four lower bounds on the query complexity of local PageRank approximation, which demonstrate the two major sources of hardness for this problem: high in-degrees and slow PageRank convergence. The first two lower bounds (one for randomized and another for deterministic algorithms) address high in-degrees, while the other two address slow PageRank convergence. For lack of space, we provide a proof of only the first lower bound. The other proofs appear in the full draft of this paper.

High in degree. The first two lower bounds demonstrate that graphs with high in-degree can be hard for local PageRank approximation. The hard instances constructed in the proofs are 3-level “tree-like” graphs with very high branching factors.⁶ Thus, PageRank converges very quickly on these graphs (in merely 2 iterations),

⁵Available at <http://www.ee.technion.ac.il/people/zivby/>.

⁶Strictly speaking, each graph we create is a union of a 3-level tree with a bunch of singleton nodes with self loops.

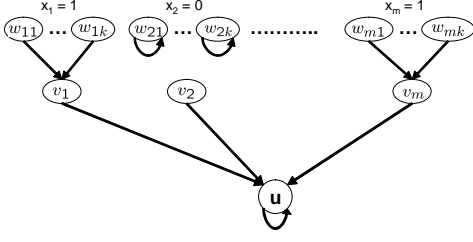


Figure 1: Hard graph (Theorem 6).

yet local PageRank approximation requires lots of queries, due to the high degrees. The first lower bound ($\Omega(\sqrt{n})$) holds for any algorithm, even randomized ones, and the second lower bound (an optimal $\Omega(n)$) holds for deterministic algorithms only.

THEOREM 6. Fix any $\alpha \in (0, 1)$, $\delta \in (0, 1)$, and $\epsilon \in (0, \frac{1}{2})$. Let A be an algorithm that locally approximates PageRank to within relative error ϵ and confidence $1 - \delta$. Then, for every sufficiently large n , there exists a graph G on at most n nodes and a node $u \in G$ on which A uses $\Omega(\sqrt{n})$ queries. Furthermore, the maximum in-degree of G is $\Omega(\sqrt{n})$, while PageRank converges in merely 2 iterations on G .

PROOF. We prove the lower bound by a reduction from the OR problem. In the OR problem, an algorithm is given a vector \mathbf{x} of m bits (x_1, \dots, x_m) , and is required to output the OR $x_1 \vee x_2 \vee \dots \vee x_m$. The algorithm has only “local access” to \mathbf{x} , meaning that in order to recover any bit x_i , the algorithm must send a query to an external server. The goal is to compute the OR with as few queries to the server as possible. A simple sensitivity argument (cf. [5, 1]) shows that $m(1 - 2\delta)$ queries are needed for computing OR to within confidence $1 - \delta$.

We reduce the OR problem to local PageRank approximation as follows. We assume $n \geq \max\{(\frac{1}{\alpha} + 1)^2 \cdot (\frac{4}{\alpha} + 1) + 1, \frac{36}{\alpha} + 10\}$. Define $m = \lfloor \sqrt{\frac{n-1}{\frac{4}{\alpha} + 1}} \rfloor$ and $k = \lfloor \frac{n-1}{m} \rfloor - 1$. Note that by the choice of n , $m \geq 1$, $k \geq 1$. Furthermore, $m \geq \Omega(\sqrt{n})$. Let S be the maximum number of queries A uses on graphs of size $\leq n$. We will use A to construct an algorithm B that computes the OR function on input vectors of length m using at most S queries. That would imply $S \geq m(1 - 2\delta) = \Omega(\sqrt{n})$.

We map each input vector $\mathbf{x} = (x_1, \dots, x_m)$ into a graph $G_{\mathbf{x}}$ on $n' = m(k+1) + 1$ nodes (see Figure 1). Note that $n' \leq n$ and therefore A uses at most S queries on $G_{\mathbf{x}}$. $G_{\mathbf{x}}$ contains a tree of depth 2, whose root is u . The tree has one node at level 0 (namely, u), m nodes at level 1 (v_1, \dots, v_m), and mk nodes at level 2 ($w_{11}, \dots, w_{1k}, \dots, w_{m1}, \dots, w_{mk}$). All the nodes at level 1 link to u . For each $i = 1, \dots, m$, the k nodes w_{i1}, \dots, w_{ik} either all link to v_i (if $x_i = 1$) or all link to themselves (if $x_i = 0$). Finally, u links to itself. Note that $G_{\mathbf{x}}$ is sink-free and has a maximum in-degree $\geq m \geq \Omega(\sqrt{n})$. Furthermore, since the longest path in $G_{\mathbf{x}}$ is of length 2 (excluding self loops), PageRank converges in merely 2 steps on any node in G .

For each node y , we denote by $\text{PR}^{G_{\mathbf{x}}}(y)$ the PageRank of y in the graph $G_{\mathbf{x}}$. The following claim shows that $\text{PR}^{G_{\mathbf{x}}}(u)$ is determined by the number of 1’s in \mathbf{x} :

CLAIM 7. Let $|\mathbf{x}|$ be the number of 1’s in \mathbf{x} . Then,

$$\text{PR}^{G_{\mathbf{x}}}(u) = \frac{1 - \alpha}{n'} (1 + \alpha m + \alpha^2 k |\mathbf{x}|).$$

PROOF. Using the influence characterization of PageRank (The-

orem 3),

$$\text{PR}^{G_{\mathbf{x}}}(u) = \frac{1 - \alpha}{n'} \sum_{t=0}^{\infty} \sum_{v \in G_{\mathbf{x}}} \alpha^t \text{inf}_t(v, u). \quad (1)$$

In $G_{\mathbf{x}}$ every node $v \in G_{\mathbf{x}}$ has at most one path to u . Furthermore, all the nodes along this path are of out-degree 1. Therefore, $\text{inf}_t(v, u) = 1$, if the path from v to u is of length t , and $\text{inf}_t(v, u) = 0$, otherwise. There is one node (u) whose path to u is of length 0, m nodes (v_1, \dots, v_m) whose path to u is of length 1, and $k|\mathbf{x}|$ nodes (nodes w_{ij} for i ’s s.t. $x_i = 1$ and $j = 1, \dots, k$) whose path to u is of length 2. We can now rewrite Equation 1 as follows: $\text{PR}^{G_{\mathbf{x}}}(u) = \frac{1 - \alpha}{n'} (1 + \alpha m + \alpha^2 k |\mathbf{x}|)$. \square

Note that $\text{PR}^{G_{\mathbf{x}}}(u)$ is the same for all \mathbf{x} that have the same number of 1’s. Furthermore, it is monotonically increasing with $|\mathbf{x}|$. Let

$$p_0 = \frac{1 - \alpha}{n'} (1 + \alpha m)$$

$$p_1 = \frac{1 - \alpha}{n'} (1 + \alpha m + \alpha^2 k).$$

The algorithm B now works as follows. Given an input \mathbf{x} , B simulates A on the graph $G_{\mathbf{x}}$ and on the target node u . In order to simulate the link server for $G_{\mathbf{x}}$, B may resort to queries to its own external server (which returns bits of \mathbf{x}): (a) If A probes the link server for u , B returns u, v_1, \dots, v_m as the in-neighbors and u as the single out-neighbor. In this case, B ’s simulation of the link server is independent of \mathbf{x} , so there is no need to probe the external server. (b) If A probes a node v_i , for $i = 1, \dots, m$, B sends i to its own server; if the answer is $x_i = 1$, B returns w_{i1}, \dots, w_{ik} as the in-neighbors and u as the out-neighbor; if the answer is $x_i = 0$, B returns only u as the out-neighbor. (c) If A probes a node w_{ij} , B sends i to the external server; if $x_i = 1$, B returns v_i as the out-neighbor; if $x_i = 0$, B returns w_{ij} as the out-neighbor and in-neighbor. After the simulation of A ends, B declares the OR to be 1, if A ’s estimation of $\text{PR}^{G_{\mathbf{x}}}(u)$ is at least $p_1(1 - \epsilon)$, and 0 otherwise.

Note that each query A sends to the link server incurs at most one query to B ’s server. So B uses a total of at most S queries. To prove that B is always correct, we analyze two cases.

Case 1: $\bigvee_{i=1}^m x_i = 1$. In this case $|\mathbf{x}| \geq 1$. Therefore, by Claim 7, $\text{PR}^{G_{\mathbf{x}}}(u) \geq p_1$. This means that A ’s output will satisfy $\overline{\text{PR}}_{\mathbf{x}}(u) \geq p_1(1 - \epsilon)$ with probability $\geq 1 - \delta$. In this case B outputs 1, as needed.

Case 2: $\bigvee_{i=1}^m x_i = 0$. In this case $|\mathbf{x}| = 0$. Therefore, by Claim 7, $\text{PR}^{G_{\mathbf{x}}}(u) = p_0$. Hence, A ’s output will satisfy $\overline{\text{PR}}_{\mathbf{x}}(u) \leq p_0(1 + \epsilon)$ with probability $\geq 1 - \delta$. The following claim shows that this value is less than $p_1(1 - \epsilon)$, and thus B outputs 0, as needed.

CLAIM 8. $p_0(1 + \epsilon) < p_1(1 - \epsilon)$.

PROOF. To prove the claim, it suffices to show that $\frac{p_1 - p_0}{p_1 + p_0} > \epsilon$. Expanding the LHS, we have: $\frac{p_1 - p_0}{p_1 + p_0} = \frac{\alpha^2 k}{2 + 2\alpha m + \alpha^2 k}$. By the choice of n ,

$$m = \lfloor \sqrt{\frac{n-1}{\frac{4}{\alpha} + 1}} \rfloor \geq \sqrt{\frac{(\frac{1}{\alpha} + 1)^2 (\frac{4}{\alpha} + 1)}{\frac{4}{\alpha} + 1}} - 1 = (\frac{1}{\alpha} + 1) - 1 = \frac{1}{\alpha}.$$

Therefore, $2 + 2\alpha m \leq 4\alpha m$, and thus

$$\frac{\alpha^2 k}{2 + 2\alpha m + \alpha^2 k} \geq \frac{\alpha^2 k}{4\alpha m + \alpha^2 k} = \frac{1}{\frac{4m}{\alpha k} + 1}.$$

From k 's definition,

$$\begin{aligned} \frac{k}{m} &= \frac{\lfloor \frac{n-1}{m} \rfloor - 1}{m} \geq \frac{\frac{n-1}{m} - 2}{m} = \frac{n-1}{(\lfloor \sqrt{\frac{n-1}{4/\alpha+1}} \rfloor)^2} - \frac{2}{\lfloor \sqrt{\frac{n-1}{4/\alpha+1}} \rfloor} \\ &\geq \frac{n-1}{4/\alpha+1} - \frac{2}{\sqrt{\frac{n-1}{4/\alpha+1}} - 1} \geq 4/\alpha + 1 - \frac{2}{\sqrt{\frac{36/\alpha+9}{4/\alpha+1}} - 1} = \\ &4/\alpha + 1 - \frac{2}{\sqrt{9} - 1} = 4/\alpha. \end{aligned}$$

Since $\epsilon < \frac{1}{2}$, $\frac{\epsilon}{1-\epsilon} > 1$. Therefore, $\frac{k}{m} \geq \frac{4}{\alpha} > \frac{4}{\alpha} \cdot \frac{\epsilon}{1-\epsilon}$. Hence, $\frac{1}{\frac{4m}{\alpha k} + 1} > \epsilon$. \square

For deterministic algorithms, we are able to strengthen the lower bound to the optimum $\Omega(n)$:

THEOREM 9. Fix any $\alpha \in (\frac{1}{2}, 1)$ and $\epsilon \in (0, \frac{2}{4+\alpha})$. Let A be a deterministic algorithm that locally approximates PageRank to within a factor of $1 \pm \epsilon$. Then, for every $n > 4$, there exists a graph G on at most n nodes and a node $u \in G$ on which A uses $\Omega(n)$ queries. Furthermore, the maximum in-degree of G is $\Omega(n)$ and PageRank converges in merely 2 iterations on G .

The proof uses a reduction from the ‘‘majority-by-a-margin’’ problem (determine whether a sequence of m bits has at least $(\frac{1}{2} + \epsilon)m$ 1’s or $(\frac{1}{2} + \epsilon)m$ 0’s). As majority-by-a-margin has a $\Omega(\frac{1}{\epsilon^2})$ lower bound for randomized algorithms [6, 1], when the approximation factor ϵ is small ($\epsilon \leq \frac{1}{\sqrt{n}}$), we obtain an $\Omega(n)$ lower bound also for randomized algorithms. It remains open to determine whether an $\Omega(n)$ lower bound holds for randomized algorithms when the approximation factor is large.

Slow PageRank convergence. The next two lower bounds demonstrate that slow PageRank convergence is another reason for the intractability of local PageRank approximation. We show an $\Omega(n^\gamma)$ lower bound for randomized algorithms (where $\gamma < \frac{1}{2}$ depends on α) and an $\Omega(n)$ lower bound for deterministic algorithms. The hard instances constructed in the proofs are deep binary trees. So, the maximum in-degree in these graphs is 2, and the high query costs are incurred by the slow convergence ($O(\log n)$ iterations). The proofs of these two lower bounds are similar to the proofs of Theorems 6 and 9. They essentially trade fast convergence for bounded in-degree, by transforming the hard input graphs from shallow trees of large in-degree into deep trees of bounded in-degree.

THEOREM 10. Fix any $\alpha \in (\frac{1}{2}, 1)$, $\epsilon \in (0, 1)$, and $\delta \in (0, \frac{1}{2})$. Let A be an algorithm that locally approximates PageRank to within relative error ϵ and confidence $1 - \delta$. Then, for every sufficiently large n , there exists a graph G on at most n nodes and a node $u \in G$ on which A uses $\Omega(n^{\frac{1+\log \alpha}{2+\log \alpha}})$ queries. Furthermore, the maximum in-degree of G is 2 and PageRank converges in $\Omega(\log n)$ iterations on u .

Note that the lower bound depends on α . The closer α is to 1, the closer is the lower bound to $\Omega(\sqrt{n})$.

THEOREM 11. Fix any $\alpha \in (\frac{1}{2}, 1)$ and $\epsilon \in (0, \frac{2\alpha-1}{2\alpha+1})$. Let A be a deterministic algorithm that locally approximates PageRank to within relative error ϵ . Then, for every $n > 4$, there exists a graph G on at most n nodes and a node $u \in G$ on which A uses $\Omega(n)$ queries. Furthermore, the maximum in-degree of G is 2 and PageRank converges in $\Omega(\log n)$ iterations on u .

As before, when the approximation factor is small ($\epsilon \leq \frac{1}{\sqrt{n}}$), the proof of this theorem gives also an $\Omega(n)$ lower bound for randomized algorithms. It remains open to determine whether an $\Omega(n)$ lower bound holds for randomized algorithms when the approximation factor is large.

5. UPPER BOUNDS

The above lower bounds imply that high in-degrees and slow PageRank convergence make local PageRank approximation difficult. We next show that local PageRank can be approximated efficiently on graphs that have low in-degrees and that admit fast PageRank convergence. In fact, a variant of the algorithm proposed by Chen *et al.* [8] is already sufficient for this purpose. In the following we present this novel variant. We also explain the difference between the variant and the original algorithm below.

The algorithm. The algorithm performs a brute force computation of $\text{PR}_r^G(u)$ (see Figure 2). Recall that

$$\text{PR}_r^G(u) = \frac{1-\alpha}{n} \sum_{t=0}^r \sum_{v \in G} \alpha^t \text{inf}_t(v, u).$$

The algorithm crawls the subgraph of radius r around u ‘‘backwards’’ (i.e., it fetches all nodes that have a path of length $\leq r$ to u). The crawling is done in BFS order. For each node v at layer t , the algorithm calculates the influence of v on u at radius t . It sums up the influence values, weighted by the factor $\frac{1-\alpha}{n} \alpha^t$. In order to compute the influence values, the algorithm uses the following recursive property of influence:

$$\text{inf}_t(v, u) = \frac{1}{\text{outdeg}(v)} \sum_{w: v \rightarrow w} \text{inf}_{t-1}(w, u). \quad (2)$$

That is, the influence of v on u at radius t equals the average influence of the out-neighbors of v on u at radius $t - 1$. Thus, the influence values at layer t can be computed from the influence values at layer $t - 1$. Note that for nodes w that do not have a path of length $t - 1$ to u , $\text{inf}_{t-1}(w, u) = 0$. Therefore, in the expression 2, we can sum only over out-neighbors w of v that have a path of length $t - 1$ to u . In the pseudo-code below, layer_t consists of all nodes that have a path of length t to u .

procedure LocalPRAAlgorithm(u)

```

1:  $\text{PR}_0^G(u) := \frac{1-\alpha}{n}$ 
2:  $\text{layer}_0 := \{u\}$ 
3:  $\text{inf}_0(u, u) := 1$ 
4: for  $t = 1, \dots, r$  do
5:    $\text{layer}_t :=$  Get all in-neighbors of nodes in  $\text{layer}_{t-1}$ 
6:   for each  $v \in \text{layer}_t$  do
7:      $\text{inf}_t(v, u) := \frac{1}{\text{outdeg}(v)} \sum_{w \in \text{layer}_{t-1}, v \rightarrow w} \text{inf}_{t-1}(w, u)$ 
8:   end for
9:    $\text{PR}_t^G(u) := \text{PR}_{t-1}^G(u) + \frac{1-\alpha}{n} \sum_{v \in \text{layer}_t} \alpha^t \text{inf}_t(v, u)$ 
10: end for
11: return  $\text{PR}_r^G(u)$ 

```

Figure 2: The local PR approximation algorithm.

Recall that $\text{PR}_r^G(u)$ converges to $\text{PR}^G(u)$ as $r \rightarrow \infty$ (Theorem 3). So, ideally, we would like to choose $r = T_\epsilon(G, u)$. Since the algorithm computes $\text{PR}_r^G(u)$, it is immediate from the definition of $T_\epsilon(G, u)$ that if the algorithm runs with $r = T_\epsilon(G, u)$, it is guaranteed to output a value which is in the interval $[(1 - \epsilon)\text{PR}^G(u), \text{PR}^G(u)]$. In practice, calculating $T_\epsilon(G, u)$ may be hard. So, we can do one of two things: (1) run the algorithm with r ,

which is guaranteed to be an upper bound on $T_\epsilon(G, u)$ (see below for details); or (2) run the algorithm without knowing r a priori, and stop the algorithm whenever we notice that the value of $\text{PR}_r^G(u)$ does not change by much. This latter approach is not guaranteed to provide a good approximation but it works well in practice.

Difference from the algorithm of Chen *et al.* Also the algorithm of Chen *et al.* constructs a subgraph by crawling the graph backwards from the target node. There are two major differences between our variant and their algorithm, though. First, the algorithm of Chen *et al.* attempts to estimate the influence of the "boundary" of the graph that was not crawled, while our algorithm refers only to the impact of the crawled subgraph. Thus, while our algorithm always provides an under-estimate of the true PageRank value, their algorithm can also over-estimate it. Second, our algorithm iteratively computes the "influence at radius r " on the target node, while their algorithm applies the standard iterative PageRank computation. The advantage in our approach is that one can bound the gap between the produced approximation and the real PageRank value in terms of the PageRank convergence rate. On the other hand, the heuristic estimation Chen *et al.* provide for the boundary influence may sometimes lead to large approximation errors.

Complexity analysis. The following notion will be used to quantify the number of nodes the local PR algorithm needs to crawl:

DEFINITION 12. For a graph G , a target node u , and $r \geq 0$, the crawl size at radius r is:

$$C_r(G, u) = |\{v \in G \mid \exists \text{ a path from } v \text{ to } u \text{ whose length} \leq r\}|.$$

PROPOSITION 13. If the local PR algorithm runs for r iterations, then its cost is $C_r(G, u)$.

The proof is immediate from the definition of crawl size. The following is a trivial bound on the crawl size. It shows that if both r and the graph's maximum in-degree are low, the brute force algorithm is efficient:

PROPOSITION 14. Let d be the maximum in-degree of G . Then,

$$C_r(G, u) \leq d^r.$$

Finally, we provide a bound on the number of iterations that the local PR algorithm needs to run. It shows that $r = O(\log(1/\text{PR}_r^G(u)))$ is always sufficient (in practice much lower r may be enough). Note that the minimum PageRank value of any node is at least $\frac{1-\alpha}{|G|}$, and thus $O(\log(1/\text{PR}_r^G(u))) = O(\log(|G|))$.

THEOREM 15. Let G be any directed graph and let $u \in G$. Then, for any $\epsilon > 0$,

$$T_\epsilon(G, u) \leq \left\lceil \frac{1}{1-\alpha} \left(\ln \frac{1}{\text{PR}_r^G(u)} + \ln \frac{2}{\epsilon} \right) \right\rceil - 1.$$

PROOF. Let $r = \left\lceil \frac{1}{1-\alpha} \left(\ln \frac{1}{\text{PR}_r^G(u)} + \ln \frac{2}{\epsilon} \right) \right\rceil - 1$. We will show:

$$\frac{\text{PR}^G(u) - \text{PR}_r^G(u)}{\text{PR}^G(u)} < \epsilon.$$

It would follow from Definition 5 that $r \geq T_\epsilon(G, u)$.

Let us denote by \mathbf{P} the PageRank transition matrix.

$$\begin{aligned} & \frac{\text{PR}^G(u) - \text{PR}_r^G(u)}{\text{PR}^G(u)} \\ &= \frac{\text{PR}^G(u) - \mathbf{P}^{r+1}(1, u) + \mathbf{P}^{r+1}(1, u) - \text{PR}_r^G(u)}{\text{PR}^G(u)} \\ &\leq \frac{|\text{PR}^G(u) - \mathbf{P}^{r+1}(1, u)|}{\text{PR}^G(u)} + \frac{|\mathbf{P}^{r+1}(1, u) - \text{PR}_r^G(u)|}{\text{PR}^G(u)}. \end{aligned} \quad (3)$$

We will show that each of the above two terms is at most $\epsilon/2$. We start with the first term. According to Sinclair's bound on the pointwise mixing time [32] (see Proposition 2.1, pages 47–48),

$$\frac{|\text{PR}^G(u) - \mathbf{P}^{r+1}(1, u)|}{\text{PR}^G(u)} \leq \frac{\lambda_{\max}^{r+1}}{\text{PR}^G(u)},$$

where $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_{|G|}|$ are the eigenvalues of \mathbf{P} ordered by absolute values and $\lambda_{\max} = \max\{|\lambda_i| : 2 \leq i \leq |G|\}$ is the second largest eigenvalue. Haveliwala and Kamvar showed in [18] that for the PageRank matrix, $|\lambda_{\max}| \leq \alpha$, and therefore,

$$\frac{\lambda_{\max}^{r+1}}{\text{PR}^G(u)} \leq \frac{\alpha^{r+1}}{\text{PR}^G(u)}.$$

To bound the latter, we use the following calculation:

CLAIM 16. If $r \geq \frac{1}{1-\alpha} (\ln \frac{1}{\text{PR}_r^G(u)} + \ln \frac{2}{\epsilon}) - 1$, then

$$\frac{\alpha^{r+1}}{\text{PR}^G(u)} \leq \frac{\epsilon}{2}.$$

The proof of claim 16 can be found in the full version of this paper.

This shows that the first term in Equation 3 is at most $\epsilon/2$. We now bound the second term. By Lemma 4,

$$|\mathbf{P}^{r+1}(1, u) - \text{PR}_r^G(u)| = \alpha^{r+1} \mathbf{M}^{r+1}(1, u).$$

Since $\mathbf{M}^{r+1}(1, u) \leq 1$, $|\alpha^{r+1} \mathbf{M}^{r+1}(1, u)| \leq \alpha^{r+1}$. Therefore,

$$\frac{|\mathbf{P}^{r+1}(1, u) - \text{PR}_r^G(u)|}{\text{PR}^G(u)} \leq \frac{\alpha^{r+1}}{\text{PR}^G(u)}.$$

Claim 16 shows that the latter is at most $\epsilon/2$. The theorem follows. \square

Optimizing by pruning. To lower the cost of the local PR algorithm in practice, we follow Chen *et al.* and apply a pruning heuristic. The idea is simple: if the influence of a node v on u at radius r is small, then only a small fraction of its score eventually propagates to $\text{PR}_r^G(u)$ and thus omitting v from the computation of $\text{PR}_r^G(u)$ should not do much harm. Furthermore, nodes whose paths to u pass only through low influence nodes are likely to have low influence on u as well, and thus pruning the crawl at low influence nodes is unlikely to neglect high influence nodes from the crawl.

The pruning heuristic is implemented by calling the procedure depicted in Figure 3. The procedure removes all nodes whose influence is below some threshold value T from layer r . Thus, these nodes will not be expanded in the next iteration.

procedure Prune(r)

- 1: **for** each $v \in \text{layer}_r$ **do**
- 2: **if** $\alpha^r \text{inf}_r(v) < T$ **then**
- 3: remove v from layer_r ,
- 4: **end if**
- 5: **end for**

Figure 3: The pruning procedure.

The problem of the pruning heuristic is that stopping the crawl whenever the PageRank value does not change much does not guarantee an approximation. In the full version of the paper we give an example for that.

6. PAGERANK VS. REVERSE PAGERANK

In the previous sections we established that there are two necessary and sufficient conditions for a graph to admit efficient local PageRank approximation: (1) quick PageRank convergence; and (2) bounded in-degree. In this section we compare two graphs in light of these criteria: the web graph and the reverse web graph. We demonstrate that while both graphs admit fast PageRank convergence, the reverse web graph has bounded in-degree and is therefore more suitable for local PageRank approximation. We also show empirically that the algorithm of Chen *et al.* performs better on the reverse web graph rather than on the web graph.

Experimental setup. We base our empirical analysis on a 280,000 page crawl of the `www.stanford.edu` domain performed in September 2002 by the WebBase project⁷. We built the adjacency matrices of these graphs, which enabled us to calculate their true PageRank and Reverse PageRank as well as to simulate link servers for the algorithm of Chen *et al.* In the PR and RPR iterative computations we used the uniform distribution as the initial distribution.

The same `stanford.edu` crawl has been previously used by Kamvar *et al.* [22] to analyze the convergence rate of PageRank on the web graph. Kamvar *et al.* also showed that the convergence rate of PageRank on a much larger crawl of about 80M pages is almost the same as the one on the `stanford.edu` crawl. In addition, Dill *et al.* [13] showed that the structure of the web is “fractal-like”, i.e., cohesive sub-regions exhibit the same characteristics as the web at large. These observations hint that the results of our experiments on the relatively small 280,000 page crawl are applicable also to larger connected components of the web graph.

Convergence rate. We start by analyzing the PageRank convergence rate. Kamvar *et al.* [21] already observed that PageRank converges very quickly on most nodes of the web graph (it converges in less than 15 iterations on most nodes, while requiring about 50 iterations to converge globally). In Figure 4, we show that a similar phenomenon holds also for the reverse web graph. The two histograms specify for each integer t , the number of pages in the `stanford.edu` graph on which PageRank and Reverse PageRank converge in t iterations. We determine that PageRank converges on a page u in t steps, if $\frac{|\text{PR}_t^G(u) - \text{PR}_{t-1}^G(u)|}{\text{PR}_{t-1}^G(u)} < 10^{-3}$. As can be seen from the results, Reverse PageRank converges only slightly slower than PageRank: on about 80% of the nodes it converges in less than 20 iterations.

Crawl growth rate. Previous studies [31] have already shown that the maximum out-degree of the web graph is much lower than its maximum in-degree. The same holds in the `stanford.edu` graph, whose maximum in-degree is 38,606, while its maximum out-degree is only 255. We show a more refined analysis, which demonstrates that the average growth rate of backward BFS crawls around target nodes with high PageRank is much slower in the reverse web graph than in the web graph.

In Figure 5, we plot the average size of a backward BFS crawl as a function of the crawl depth for the `stanford.edu` graph and for its reverse. To create the plot for the regular graph, we selected random nodes from the graph as follows. We ordered all the nodes in the graph by their PageRank, from highest to lowest. We divided the nodes into buckets of exponentially increasing sizes (the first bucket had the top 12 nodes, the second one had the next 24 nodes, and so on). We picked from each bucket 100 random nodes (if the bucket was smaller we took all its nodes), and performed a backward BFS crawl from each sample node up to depth 9. For each bucket and for each $t = 1, \dots, 9$, we calculated the average num-

⁷Available at vlado.fmf.uni-lj.si/pub/networks/data/mix/mixed.htm.

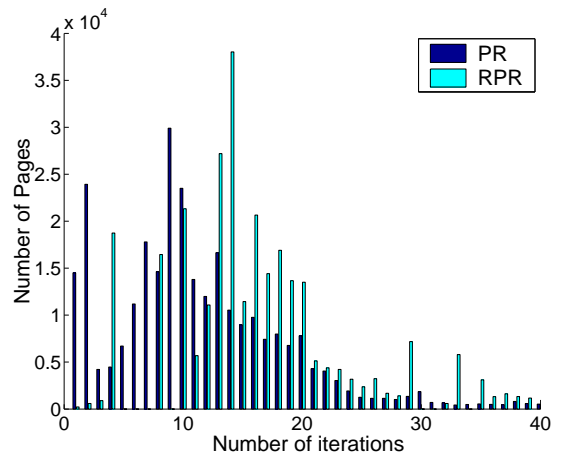


Figure 4: Convergence times for PageRank and Reverse PageRank on the `stanford.edu` graph.

ber of nodes crawled up to depth t when starting the crawl from a node in the bucket. The plot for the reverse graph was constructed analogously. We present in Figure 5 the results for the top bucket (12 pages with highest PageRank/Reverse PageRank), the middle bucket (768 pages with intermediate PR/RPR) and the last bucket (85,000 pages with lowest PR/RPR).

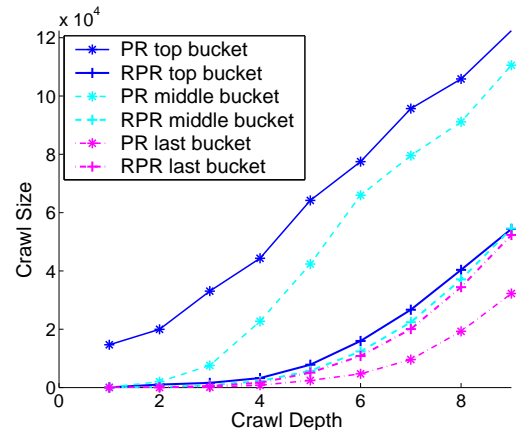


Figure 5: Average growth rates of backward BFS crawls at the `stanford.edu` graph and its reverse.

The graph clearly indicates that the growth rate of the backward BFS crawl in the reverse web graph is slower than in the regular graph for pages with high PR/RPR. For example, the average crawl size at depth 6 in the top bucket on the regular graph was 77,480, while the average crawl size at depth 6 in the top bucket on the reverse graph was 15,980 (a gap of 80%). The situation was opposite for the low ranked nodes. For example, the average crawl size at depth 6 in the last bucket on the reverse graph was 10,835, while the average crawl size at depth 6 in the last bucket on the regular graph was 4,701 (a gap of 57%). As we show below, the decreased crawl growth rate for the highly ranked nodes well pays off the increase in crawl growth rate for the low ranked nodes.

Algorithm’s performance. We made a direct empirical comparison of the performance of the algorithm of Chen *et al.* on the web graph vs. the reverse web graph. To do the comparison, we used

the same buckets and samples as the ones used for evaluating the crawl growth rate. We then calculated, for each bucket, the average cost (number of queries to the link server) of the runs on samples from that bucket. The results are plotted in Figure 6.

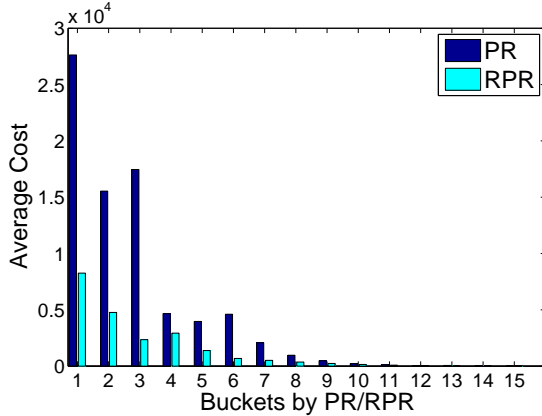


Figure 6: Average cost of the Chen *et al.* algorithm by PageRank/Reverse PageRank values. Results of runs on the `stanford.edu` graph and its reverse.

The graph shows that the cost of the algorithm on the reverse graph is significantly lower than on the regular graph, especially for highly ranked nodes. For example, the average cost of the algorithm on the first bucket of PageRank was three times higher than the cost of the algorithm on the first bucket of Reverse PageRank. On the other hand, for the low ranked nodes, the increased crawl growth rate on the reverse graph and the regular graph are almost the same. For example, the average cost of the algorithm on the last bucket of PageRank was 13 and for Reverse PageRank it was 14.

7. APPLICATIONS OF REVERSE PAGERANK

RPR has already been used in the past to select good seeds for the TrustRank measure [16], to detect highly influential nodes in social networks [19], and to find hubs in the web graph [14]. In this section we present two additional novel applications: (1) finding good seeds for crawling; and (2) measuring the “semantic relatedness” of concepts in a taxonomy.

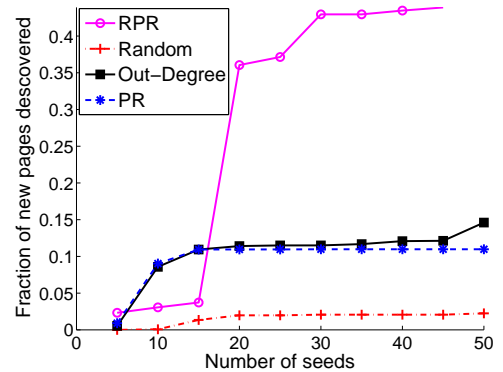
We note that local RPR approximation is potentially useful in several of these applications. For example, to estimate the influence score of a given node in a social network, the hub score of a given page on the web, or the semantic relatedness of two given concepts in a taxonomy. Social networks exhibit similar properties to the web graph [27], such as the power law degree distribution and the gap between in- and out- degrees. As shown below, the same holds for the taxonomy graph extracted from the Open Directory Project.

7.1 Finding crawl seeds

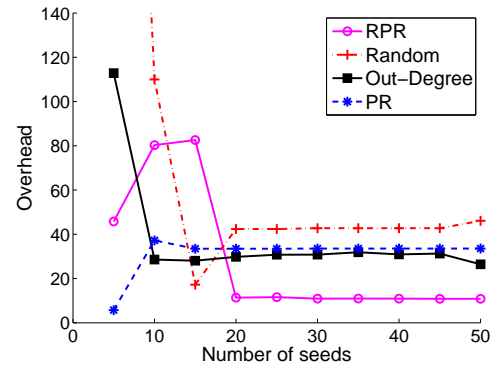
Discoverability of the web. Motivated by the fast pace of web growth and the need of crawlers to discover new content quickly, Dasgupta *et al.* [10] have recently posed the following question: how can a crawler discover as much new content as possible, while incurring as little “overhead” as possible? Dasgupta *et al.* define the overhead of a crawler to be the average number of old pages it needs to refetch per new discovered pages. Formally, if the crawler

refreshes a set S of “seed” pages previously crawled, resulting in a set $N(S)$ of new pages being discovered, then the overhead is $|S|/|N(S)|$. Dasgupta *et al.* present crawling algorithms that ensure low overhead and analyze them theoretically and empirically.

Selecting seeds using Reverse PageRank. We show that RPR is an effective strategy for finding good seeds. Our algorithm simply chooses the nodes with highest Reverse PageRank values to be the seed set. The intuition behind this is the following. A page p has high RPR if many pages are reachable from p by short paths, and moreover these pages are not reachable from many other pages. Thus, by selecting p as a seed, we benefit from discovering many new pages without doing too many fetches (because the paths leading to them are short) and furthermore these new pages are not “covered” by other potential seeds. Assuming the web graph does not change drastically between two crawls, we can predict the Reverse PageRank of the nodes in the new graph by calculating RPR on the already known sub-graph.



(a) Fraction of the new pages discovered by the crawler versus the number of seeds.



(b) Overhead versus the number of seeds.

Figure 7: 4-level BFS crawl.

Experimental results. To evaluate this seed selection strategy, we used two 1 million page Stanford WebBase crawls⁸. The two crawls are of the same sites and were conducted one week apart in May 2006. The later crawl consists of 132,000 new pages. We compared four seed selection strategies: the k pages with highest

⁸<http://www-diglib.stanford.edu/~testbed/doc2/WebBase>.

RPR scores, the k pages with highest PR scores, the k pages with largest out-degree, and k random pages. We chose the seeds from the nodes of the first crawl and performed a BFS crawl for t levels starting from these seeds on the second crawl. Figure 7 shows the results for $t = 4$. We can see that RPR performs significantly better than the rest of the strategies, discovering more than twice new content with less overhead compared to any other strategy.

7.2 Measuring semantic relatedness

Semantic relatedness indicates how much two concepts are related to each other. Semantic relatedness is used in many applications in natural language processing, such as word sense disambiguation, information retrieval, interpretation of noun compounds, and spelling correction (cf. [34]).

In the experiments below, we focus on measuring semantic relatedness between concepts represented as nodes in the Open Directory Project⁹ (ODP) taxonomy. Given two nodes in ODP, we wish to find the relatedness between the concepts corresponding to these nodes. Note that the ODP is a directed graph, whose links represent an *is-a* relation between concepts. Thus two concepts should be related if the sets of nodes that are reachable from them are “similar”.

Previously, Strube and Ponzetto [34] used Wikipedia for computing semantic relatedness. Given a pair of words w_1 and w_2 , their method, called WikiRelate!, searches for Wikipedia articles, p_1 and p_2 that respectively contain w_1 and w_2 in their titles. Semantic relatedness is then computed using various distance measures between p_1 and p_2 . Also the ODP was previously used to measure semantic relatedness by Gabrilovich and Markovitch in [15]. The authors used machine learning techniques to explicitly represent the meaning of any text as a weighted vector of concepts. We show that (personalized) Reverse PageRank can be also used to measure semantic relatedness. Note that to this end we do not use any textual analysis of the taxonomy, only its graph structure.

Given two nodes x, y in the ODP graph, we compute two personalized Reverse PageRank vectors RPR_x and RPR_y . RPR_x is the personalized Reverse PageRank vector of the ODP graph corresponding to a personalization vector that has 1 in the position corresponding to x and 0 everywhere else. Note that for a node a , a high value of $RPR_x(a)$ implies there are many short paths from x to a . This implies a is a prominent sub-concept of x and it is not a prominent sub-concept for (many) other nodes. Thus, the vector RPR_x represents x by the weighted union of its sub-concepts.

We evaluate two alternative techniques for using these vectors in measuring semantic relatedness: (1) Reverse PageRank: the measure of y as a sub-concept of x is the score $RPR_x(y)$ and the measure of x as a sub-concept of y is the score $RPR_y(x)$; (2) Reverse PageRank similarity: two concepts will be similar in case they have significant overlap between their Reverse PageRank vectors. Therefore, the similarity between x and y is the cosine similarity between the vectors RPR_x and RPR_y . At first glance, RPR similarity seems more accurate than RPR, but RPR has a computational advantage since we can calculate $RPR_x(y)$ by using the local approximation algorithm. In our experiments we compare the quality of these two measures.

An alternative graph-based approach for finding related nodes in a graph is the *cocitation algorithm* [12]. Two nodes are *cocited* if they share a common parent. The number of common parents of two nodes is their *degree of cocitation*. This measure is not suitable for us, since parent sharing is quite rare in the ODP. Another measure of semantic relatedness is the *path-based measure* [30], which defines the semantic distance (inverse of relatedness) between two

nodes as the length of the shortest path between them in the graph.

Experimental results. We base our experiment on a 110,000 page crawl of the ODP. First, we verified that the reverse ODP graph admits the two conditions of efficient local PageRank approximation. We analyzed the Reverse PageRank convergence rate and saw that more than 90% of the nodes converged in less than 20 iterations. The maximum out-degree of the graph was 2745.

To evaluate the semantic relatedness measures, we chose a collection of concepts (“main concepts”) from the ODP and ranked another collection of concepts (“test concepts”) according to their relatedness to the main concepts. We used three methods for measuring relatedness: Reverse PageRank, Reverse PageRank similarity, and inverse path-based. Table 8(a) shows the ordering of the test concepts by relatedness to the main concept “Einstein” using each one of the techniques. Table 8(b) shows a similar comparison for the main concept “ice climbing”.

RPR	RPR similarity	Path-based
Einstein, Albert	Einstein, Albert	Einstein, Albert
Physics Prize	Newton, Isaac	Agriculture
Physics	Physics Prize	Internet
Newton, Isaac	Physics	Nuclear
Nuclear	Nuclear	Physics Prize
Agriculture	World War II	Pizza
United States	Agriculture	United States
World War II	Helicopter	Physics
Internet	Ronald Reagan	Newton, Isaac
Helicopter	Italy	Italy
Ronald Reagan	Internet	World War II
Italy	United States	Ronald Reagan
Pizza	Sudoku	Helicopter
Sudoku	Pizza	Sudoku

(a) Relatedness to “Einstein”.

RPR	RPR similarity	Path-based
Ice climbing	Ice climbing	Ice climbing
Climbing	Rock Climbing	Climbing
Mountaineering	Mountaineering	Camping
Rock Climbing	Climbing	Rock climbing
Hiking	Hiking	Baseball
Hunting	Hunting	Fishing
Fishing	Fishing	Gardening
Baseball	Camping	Card games
Camping	Dogs	Dogs
Gardening	Baseball	Hunting
Dogs	Yoga	Mountaineering
Ireland	Gardening	Yoga
Yoga	Card games	Ireland
Card games	Ireland	Hiking

(b) Relatedness to “Ice climbing”.

Figure 8: Test concepts ordered by their relatedness to a main concept.

As can be seen from the results, the Reverse PageRank-based rankings were much better than the path-based ranking: while the path-based measure ranked “Agriculture” and “Internet” as very related concepts to “Einstein”, both our measures ranked “physics prize” and “Newton, Isaac” on the top of the list. For the “ice climbing” concept, the path-based measure ranked “Basketball” and “Card game” before “Mountaineering” and “Hiking”, while both of them were ranked high by the RPR measures. We can also see from the experiment that the quality of RPR measure is almost the same as RPR similarity measure, which means we can use the

⁹<http://www.dmoz.org>.

local approximation algorithm to find semantic relatedness.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have studied the limitations of local PageRank approximation. We have shown that in the worst-case $\Omega(\sqrt{n})$ queries to the link server are needed in order to obtain a good PageRank approximation. For deterministic algorithms, a stronger (and optimal) $\Omega(n)$ lower bound was shown. For future work, it will be interesting to determine whether an $\Omega(n)$ lower bound holds for randomized algorithms as well.

We have identified two graph properties that make local PageRank approximation hard: abundance of high in-degree nodes and slow convergence of the PageRank random walk. We have shown that graphs that do not have these properties do admit efficient local PageRank approximation. A future direction can be to explore whether local algorithms could be used to estimate the relative order of the PageRank values.

As the web graph has many high in-degree nodes, we conclude that it is not suitable for local PageRank approximation. We have validated this conclusion by empirical analysis over a large crawl. We then have shown that the reverse web graph is amenable to efficient local PageRank approximation, as it has bounded in-degree and it admits quick PageRank convergence. We have demonstrated empirically that the algorithm of Chen *et al.* [8] indeed performs much better on the reverse web graph than on the web graph. We leave for a future work to evaluate the property of the crawl growth rate for certain models of the Web graphs, such as preferential attachment [2], the copying model [26], etc.

Finally, we have presented two novel applications of Reverse PageRank. The first application is detecting good seeds for crawling. In our experiments we have compared the Reverse PageRank to three other methods for seeds choice. As part of future work it would be interesting to compare our method to additional known methods in different crawler models, for example, the ones that were presented by Cho *et al.* in [9]. The second novel application is measuring semantic relatedness between concepts in a taxonomy. The experimental study we have conducted on the ODP taxonomy shows promising directions. In the future it will be interesting to evaluate the Reverse PageRank measures on the more complex *wikipedia*¹⁰ graph.

9. REFERENCES

- [1] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Sampling algorithms: Lower bounds and applications. In *STOC*, pages 266–275, 2001.
- [2] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
- [3] P. Berkhin. A survey on PageRank computing. *Internet Mathematics*, 2(1):73–120, 2005.
- [4] A. Z. Broder, R. Lempel, F. Maghoul, and J. O. Pedersen. Efficient PageRank approximation via graph aggregation. *Inf. Retr.*, 9(2):123–138, 2006.
- [5] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Comp. Sc.*, 288(1):21–43, 2002.
- [6] R. Canetti, G. Even, and O. Goldreich. Lower bounds for sampling algorithms for estimating the average. *Information Processing Letters*, 53:17–25, 1995.
- [7] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks*, (11–16):1623–1640, 1999.
- [8] Y. Chen, Q. Gan, and T. Suel. Local methods for estimating PageRank values. In *Proc. CIKM*, pages 381–389, 2004.
- [9] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.
- [10] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins. The discoverability of the web. In *Proc. 16th WWW*, pages 421–430, 2007.
- [11] J. V. Davis and I. S. Dhillon. Estimating the global PageRank of Web communities. In *Proc. 12th SIGKDD*, pages 116–125, 2006.
- [12] J. Dean and M. R. Henzinger. Finding related pages in the World Wide Web. *Computer Networks*, 31(11–16):1467–1479, 1999.
- [13] S. Dill, R. Kumar, K. Mccurley, S. Rajagopalan, D. Sivakumar, and A. Tomkins. Self-similarity in the web. *ACM Trans. Internet Techn.*, 2(3):205–223, 2002.
- [14] D. Fogaras. Where to start browsing the Web? In *IICS*, pages 65–79, 2003.
- [15] E. Gabrilovich and S. Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proc. 20th IJCAI*, pages 250–257, 2007.
- [16] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating Web Spam with TrustRank. In *VLDB*, pages 576–587, 2004.
- [17] T. H. Haveliwala. Topic-sensitive PageRank: a context-sensitive ranking algorithm for web search. *IEEE Trans. on Knowledge and Data Engineering*, 15(4):784–796, 2003.
- [18] T. H. Haveliwala and S. D. Kamvar. The second eigenvalue of the Google matrix. Technical report, Stanford University, 2003.
- [19] A. Java, P. Kolar, T. Finin, and T. Oates. Modeling the spread of influence on the Blogosphere. Technical report, University of Maryland, Baltimore County, 2006.
- [20] G. Jeh and J. Widom. Scaling personalized Web search. In *Proc. 12th WWW*, pages 271–279, 2003.
- [21] S. Kamvar, H. Haveliwala, and G. Golub. Adaptive methods for the computation of PageRank. *Linear Algebra and its Applications*, 386:51–65, 2004.
- [22] S. Kamvar, T. Haveliwala, and G. Golub. Adaptive methods for the computation of pagerank, 2003.
- [23] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Exploiting the block structure of the Web for computing PageRank. Technical report, Stanford University, 2003.
- [24] C. Kohlschütter, P. A. Chirita, and W. Nejdl. Efficient parallel computation of PageRank. In *Proc. 28th ECIR*, pages 241–252, 2006.
- [25] G. Kollias and E. Gallopoulos. Asynchronous computation of PageRank computation in an interactive multithreading environment. In *Web Information Retrieval and Linear Algebra Algorithms*, 2007.
- [26] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *41th IEEE Symposium On Foundations of Computer Science (FOCS)*, pages 57–65, 2000.
- [27] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The web and social networks. *Computer*, 35(11):32–36, 2002.
- [28] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.
- [29] J. X. Parreira, D. Donato, S. Michel, and G. Weikum. Efficient and decentralized PageRank approximation in a Peer-to-Peer Web search network. In *Proc. 32nd VLDB*, pages 415–426, 2006.
- [30] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Trans. on Systems, Man and Cybernetics*, 19(1):17–30, 1989.
- [31] M. A. Serrano, A. G. Maguitman, M. Boguñá, S. Fortunato, and A. Vespignani. Decoding the structure of the WWW: A comparative analysis of Web crawls. *TWEB*, 1(2), 2007.
- [32] A. Sinclair. *Algorithms for Random Generation and Counting: a Markov Chain Approach*. Birkhauser Verlag, Basel, Switz., 1993.
- [33] A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.
- [34] M. Strube and S. P. Ponzetto. WikiRelate! computing semantic relatedness using Wikipedia. In *AAAI 2006*, pages 1419–1424, 2006.
- [35] Y. Wang and D. J. DeWitt. Computing PageRank in a distributed Internet search engine system. In *VLDB*, pages 420–431, 2004.
- [36] Y. Wu. Subgraphrank: PageRank approximation for a subgraph or in a decentralized system. VLDB PhD workshop, 2007.

¹⁰<http://www.wikipedia.org/>.