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`

May 5, 2008

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 [42] 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 ranking web pages in search results. Since then it has found uses in many other domains, such measuring centrality in social networks [26], evaluating the importance of scientific publications, prioritizing pages in a crawler's frontier [12], personalizing search results [10], combating spam [22], measuring trust, selecting pages for indexing, and

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

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

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 natural contexts, where PageRank is used. For example, a Facebook[2] 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, Gan, and Suel [11] were the first to introduce the problem of *local PageRank approximation*. Suppose we are given access to a large 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 then 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. In this work 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]

[2]http://www.facebook.com/.

[3]If 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.

[4]Note 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 [24] for

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 lower bound has very fast PageRank convergence (2 iterations), while the family of hard graphs constructed in the second lower bound has bounded in-degree (2).

Sufficiency. Having proved that local PageRank approximation is hard for graphs that either 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 two 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 to do on such graphs. We substantiate 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, requiring thousands of queries to the link server. 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 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

more details.

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 RPR in the first place? We observe that RPR has a multitude of applications. It has been used before to select good seeds for the TrustRank measure [22], to detect highly influential nodes in social networks [26], and to find hubs in the web graph [19]. 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 in3 of the RPR applications: influencers detection, hub score computation, and measuring of semantic relatedness.

2 Related Work

PageRank was first introduced in [42] by Page *et al.* as a global ranking of web pages, independent of their textual content, solely based on the hyperlink structure of the web. It was shown in [42] that the PageRank vector can be obtained by calculating a principal eigenvector of a linear system, which can be found via the power method [21]. Due to the immense size of the Web graph this computation demands enormous computational resources and can take a significant amount of time. This brought many researchers to look for other possible solution methods.

There is a large body of work on accelerating PageRank computation, varying from centralized algorithms (e.g. [28, 30, 8, 1, 39, 17]), to parallel algorithms (e.g., [35, 34]) to decentralized P2P algorithms (e.g., [52, 44]). In a recent work [40] McSherry claims that with some clever engineering and exploitation of the unique properties of the Web graph, computing PageRank is much less daunting than is typically perceived. All of these are designed for computing the whole PageRank vector and are therefore not directly applicable to our problem. See a survey by Berkhin [5] for an extensive review of global PageRank computation techniques.

The idea of calculating PageRank locally was studied in previous works. The most related one is the work of Chen, Gan, and Suel [11], which uses the set of influential nodes to calculate the PageRank of a single page. The algorithm is based on the availability of a link server that can supply the set of in-links to a given page. They experimentally show that a reasonable estimate of the node's PageRank can be obtained by visiting at most a few hundred nodes. Chen *et al.* noticed that their results have a significant estimation error from time to time. Our work gives explanation to this phenomenon and provides the theoretical analysis of the lower bound for these family of algorithms. In addition, we discuss the specific case of using the approximate algorithm on bounded in-degree graphs, and in particular, for calculating Reverse PageRank.

Lempel and Moran [38] used proof techniques that are similar to ours in the context of rank-stability and rank-similarity. They constructed combinatorial graph structures to prove worst-case notions on PageRank, HITS and SALSA algorithms.

In the second part of our work we discuss applications of Reverse PageRank. One of the previously known applications of Reverse PageRank is in the context of TrustRank, first introduced by Gyöngyi *et al.* in [22]. The idea is to choose pages with high Reverse PageRank as seed nodes and to give preference to pages from which many other pages could be reached. Fogaras [19] suggested Reverse PageRank as a measure of *good hubs*. He noticed that in ranking keyword queries Reverse PageRank gives high credit to archives or large collections of databases within a web site. In [26] Java *et al.* suggested calculating Reverse PageRank on a social networks of bloggers to detect influential bloggers.

The above works did not consider local approximation of Reverse PageRank, which appears to be very suitable for finding hub web pages and influencers in social networks.

3 Preliminaries

3.1 PageRank overview

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

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

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 [42] 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 as well. One problem with using solely the web's hyperlink structure to build the transition matrix is the fact that some rows of the matrix might contain all zeroes. Thus $\mathbf{M}$ is not stochastic. This occurs whenever a node contains no out-links. Such nodes are called dangling nodes. A simple trick to eliminate dangling nodes consists of introducing a dummy node, which has links to all nodes and is pointed by every dangling node. We have chosen to use this solution in our experiments.

Personalized PageRank [23, 27] 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. Nevertheless, we choose to stick to the uniform case, for simplicity of exposition.

For more information about PageRank, refer to recent surveys on the topic [6, 7].

3.2 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 $\overline{PR}(u)$ satisfying:*

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

If the algorithm is randomized, it is required to output, for any inputs G, u, ϵ, a $1 \pm \epsilon$ approximation of $PR^G(u)$ 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 poly-logarithmic cost ($O(\log^{O(1)}(n))$ as feasible and polynomial cost ($\Omega(n^{1-\epsilon})$ for some $\epsilon > 0$) as infeasible.

3.3 PageRank and influence

Jeh and Widom [27] and Chen *et al.* [11] 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. Later, this will make the analysis easier.

The *influence* [11] 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 $p = (u_0, u_1, \ldots, u_t)$, 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 $\text{inf}_0(1, u) = 1$ *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:

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

$\mathtt{PR}_r^G(u)$ 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 [27].

Theorem 3. *For every node $u \in G$,*

$$\lim_{r \to \infty} \mathit{PR}_r^G(u) = \mathit{PR}^G(u).$$

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

Lemma 4. *For every $r \geq 1$,*

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

Before we prove the lemma, let us show how to use it to complete the proof of the theorem:

$$\lim_{r \to \infty} \mathtt{PR}_r^G(u) = \lim_{r \to \infty} (\mathbf{P}^{r+1}(1, u) - \alpha^{r+1} M^{r+1}(1, u))$$

$$= \lim_{r\to\infty} \mathbf{P}^{r+1}(1,u) - \lim_{r\to\infty} \alpha^{r+1}\mathbf{M}^{r+1}(1,u).$$

As $\mathbf{M}^{r+1}$ is a probability transition matrix, $\mathbf{M}^{r+1}(1,u) \leq 1$, and thus as $t \to \infty$, $\alpha^{r+1}\mathbf{M}^{r+1}(1,u) \to 0$. Which means that:

$$\lim_{r\to\infty} \mathtt{PR}_r^G(u) = \lim_{r\to\infty} \mathbf{P}^{r+1}(1,u).$$

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

$$\lim_{r\to\infty} \mathtt{PR}_r^G(u) = \mathtt{PR}^G(u).$$

Next we prove Lemma 4. To this end, we show the following characterization of powers of the PageRank matrix:

Claim 5. *For every $r \geq 1$,*

$$\mathbf{P}^r = \alpha^r\mathbf{M}^r + (1-\alpha)\cdot\sum_{i=0}^{r-1}\alpha^i\mathbf{U}\mathbf{M}^i.$$

Proof.

$$\begin{aligned}\mathbf{P}^t &= (\alpha\mathbf{M} + (1-\alpha)\mathbf{U})^t \\ &= \sum_{S\subseteq\{1,2,\dots,t\}} \alpha^{|S|}(1-\alpha)^{t-|S|}\cdot\pi_S(\mathbf{M},\mathbf{U}).\end{aligned}$$

Here, $\pi_S(\mathbf{M},\mathbf{U})$ is a product of t matrices, where $\mathbf{M}$ occurs in the positions corresponding to S and $\mathbf{U}$ occurs in all other positions. For example, for $t=4$ and $S=\{1,3\}$, $\pi_S(\mathbf{M},\mathbf{U}) = \mathbf{MUMU}$.

Note that if $\mathbf{T}_1$ and $\mathbf{T}_2$ are transition matrices of two random walks on the same graph, then $\mathbf{T}_1\mathbf{T}_2$ is the transition matrix of a composite random walk, in which each step consists of first taking a step according to $\mathbf{T}_1$ and then a step according to $\mathbf{T}_2$. Now, as $\mathbf{U}$ represents a random walk in which each step is done independently of the currently visited node, then when we compose $\mathbf{U}$ with any other random walk, we get $\mathbf{U}$ itself. That is, for every $\mathbf{T}$, $\mathbf{TU} = \mathbf{U}$. It follows that $\pi_S(\mathbf{M},\mathbf{U})$ always has the form $\mathbf{UM}^k$ for some k. Formally, $\pi_S(\mathbf{M},\mathbf{U}) = \mathbf{UM}^{t-\max(S^c)}$, where $S^c = \{1,\dots,t\}\setminus S$ is the complement of S (in the edge case $S^c = \emptyset$, $\pi_S(\mathbf{M},\mathbf{U}) = \mathbf{M}^t$). As a result, we rewrite the expression for $\mathbf{P}^t$ as follows:

$$\mathbf{P}^t = \alpha^t\mathbf{M}^t + \mathbf{U}\cdot\sum_{S\subsetneq\{1,2,\dots,t\}}\alpha^{|S|}(1-\alpha)^{t-|S|}\cdot\mathbf{M}^{t-\max(S^c)}.$$

We now rewrite the sum on the RHS:

$$\begin{aligned}
&\sum_{S\subsetneq\{1,2,\ldots,t\}} \alpha^{|S|}(1-\alpha)^{t-|S|}\cdot \mathbf{M}^{t-\max(S^c)}\\
&= \sum_{i=1}^{t}\mathbf{M}^{t-i}\cdot \sum_{S\subsetneq\{1,\ldots,t\},\max(S^c)=i}\alpha^{|S|}(1-\alpha)^{t-|S|}\\
&= \sum_{i=1}^{t}\mathbf{M}^{t-i}\cdot\alpha^{t-i}(1-\alpha)\cdot\sum_{T\subseteq\{1,\ldots,i-1\}}\alpha^{|T|}(1-\alpha)^{i-1-|T|}\\
&= \sum_{i=1}^{t}\mathbf{M}^{t-i}\cdot\alpha^{t-i}(1-\alpha)\cdot(\alpha+(1-\alpha))^{i-1}\\
&= (1-\alpha)\cdot\sum_{i=1}^{t}\alpha^{t-i}\mathbf{M}^{t-i} \;=\; (1-\alpha)\cdot\sum_{i=0}^{t-1}\alpha^{i}\mathbf{M}^{i}.
\end{aligned}$$

□

Using Claim 5, for every $t \geq 1$,

$$\begin{aligned}
\mathbf{P}^r(1,u)-\alpha^r\mathbf{M}^r(1,u) &= (1-\alpha)\cdot\sum_{t=0}^{r-1}\alpha^t(\mathbf{U}\mathbf{M}^t)(1,u)\\
&= (1-\alpha)\cdot\sum_{t=0}^{r-1}\alpha^t\sum_{v\in G}\mathbf{U}(v)\mathbf{M}^t(v,u)\\
&= \frac{1-\alpha}{n}\sum_{t=0}^{r}\sum_{v\in G}\alpha^t\mathbf{M}^t(v,u).
\end{aligned}$$

We complete the proof of Claim 5, by noting the following identity between $\mathbf{M}^t(v,u)$ and $\inf_t(v,u)$:

Claim 6. *For all u, v, t, $\inf_t(v,u) = \mathbf{M}^t(v,u)$.*

Proof. We prove the claim by induction on t. For the base case, $t = 0$, note that $\mathbf{M}^0 = \mathbf{I}$, the identity matrix. Thus, $\mathbf{M}^0(1,u) = 1 = \inf_0(1,u)$ and $\mathbf{M}^0(v,u) = 0 = \inf_0(v,u)$, for $v \neq u$. For $t = 1$, there are two sub-cases: either there is a link from v to u or not. In the former case, there is exactly one length-1 path from v to u, and its weight is $1/\operatorname{outdeg}(v)$. This weight is exactly the same as the entry $\mathbf{M}(v,u)$. Hence, $\inf_1(v,u) = 1/\operatorname{outdeg}(v) = \mathbf{M}(v,u)$. In the latter case, there is no length-1 path from v to u, and thus $\inf_1(v,u) = 0 = \mathbf{M}(u,v)$.

Let $t \geq 2$. Assume that for all u, v and all $t' < t$, $\inf_{t'}(v,u) = \mathbf{M}^{t'}(v,u)$. Note that every path p of length t from v to u is the concatenation of a path p_1 of length $t-1$ from v to some node w and a path p_2 of length 1 from w to u. Furthermore, $\text{weight}(p) = \text{weight}(p_1)\cdot\text{weight}(p_2)$. Conversely, the concatenation of every path p_1 of length $t-1$ from v to some node w and every path p_2 of length 1 from w to u yields a path p of length t from v to u s.t. $\text{weight}(p) = \text{weight}(p_1)\cdot\text{weight}(p_2)$.

We can therefore rewrite $\inf_t(v,u)$ as follows:

$$\begin{aligned}
\inf_t(v,u) &= \sum_{p \in \text{paths}_t(v,u)} \text{weight}(p) \\
&= \sum_{w \in V} \sum_{p_1 \in \text{paths}_{t-1}(v,w)} \sum_{p_2 \in \text{paths}_1(w,u)} \text{weight}(p_1) \cdot \text{weight}(p_2) \\
&= \sum_{w \in V} \left(\sum_{p_1 \in \text{paths}_{t-1}(v,w)} \text{weight}(p_1) \right) \cdot \\
&\quad \left(\sum_{p_2 \in \text{paths}_1(w,u)} \cdot \text{weight}(p_2) \right) \\
&= \sum_{w \in V} \inf_{t-1}(v,w) \cdot \inf_1(w,u).
\end{aligned}$$

Using now the induction hypothesis, we have:

$$\begin{aligned}
&\sum_{w \in V} \inf_{t-1}(v,w) \cdot \inf_1(w,u) \\
&\quad = \sum_{w \in V} \mathbf{M}^{t-1}(v,w) \cdot \mathbf{M}(w,u) \;=\; \mathbf{M}^t(v,u).
\end{aligned}$$

□

□

Note that $\texttt{PR}_r^G(u)$ approaches $\texttt{PR}^G(u)$ from below. Throughout this paper, we will use the following notion of influence convergence rate, which is reminiscent of the standard mixing time [50] of Markov Chains:

Definition 7. *For a graph G, a target node u, and an approximation parameter $\epsilon > 0$, define the* pointwise influence mixing time *as:*

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

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 $r \to \infty$. 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 algorithms and another for deterministic algorithms) address high in-degrees, while the other two lower bounds address slow PageRank convergence.

4.1 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.[5] Thus, PageRank converges very quickly on these graphs (in merely 2 iterations), 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 8. *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 $(x_1, \ldots, x_m)$, and is required to output the OR $x_1 \vee x_2 \vee \cdots \vee x_m$. 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. [9, 3]) 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 $\mathbf{x} = (x_1, \ldots, x_m)$ 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 $(v_1, \ldots, v_m)$, and mk nodes at level 2 $(w_{11}, \ldots, w_{1k}, \ldots, w_{m1}, \ldots, w_{mk})$. All the nodes at level 1 link to u. For each $i = 1, \ldots, m$, the k nodes $w_{i1}, \ldots, w_{ik}$ 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 $\mathtt{PR}^G_{\mathbf{x}}(y)$ the PageRank of y in the graph $G_{\mathbf{x}}$. The following claim shows that $\mathtt{PR}^G_{\mathbf{x}}(u)$ is determined by the number of 1's in $\mathbf{x}$:

Claim 9. *Let $|\mathbf{x}|$ be the number of 1's in $\mathbf{x}$. Then,*

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

Proof. Using the influence characterization of PageRank (Theorem 3),

$$\mathtt{PR}^G_{\mathbf{x}}(u) \;=\; \frac{1-\alpha}{n'} \sum_{t=0}^{\infty} \sum_{v \in G_{\mathbf{x}}} \alpha^t \inf_t(v,u). \tag{1}$$

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, $\inf_t(v,u) = 1$, if the path from v to u is of length t,

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

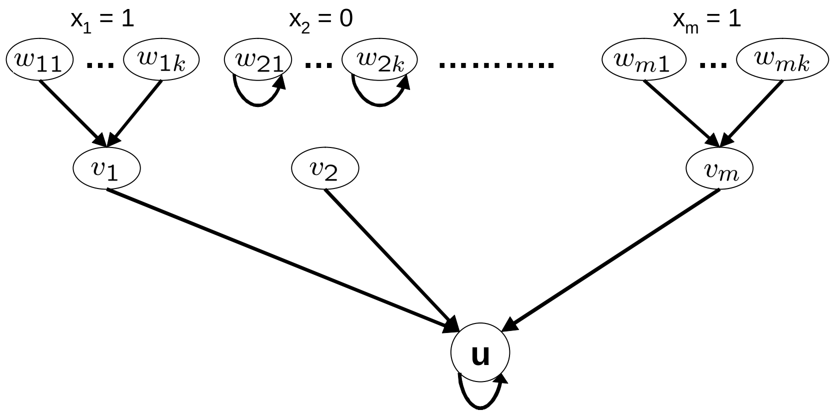


Figure 1: Hard graph (Theorem 8).

and $\inf_t(v, u) = 0$, otherwise. There is one node (u) whose path to u is of length 0, m nodes $(v_1, \ldots, v_m)$ whose path to u is of length 1, and $k|\mathbf{x}|$ nodes (nodes w_{ij} for i's s.t. $\mathbf{x}_i = 1$ and $j = 1, \ldots, k$) whose path to u is of length 2. We can now rewrite Equation 1 as follows:

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

□

Note that $\mathtt{PR}^G_{\mathbf{x}}(u)$ 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,$v_1, \ldots, v_m$ 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 $i = 1, \ldots, m$, B sends i to its own server; if the answer is $x_i = 1$, B returns $w_{i1}, \ldots, w_{ik}$ 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 $\mathtt{PR}^G_{\mathbf{x}}(u)$ 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 9, $\mathtt{PR}^G_{\mathbf{x}}(u) \geq p_1$. This means that A's output will satisfy $\overline{\mathtt{PR}}_{\mathbf{x}}(u) \geq p_1(1-\epsilon)$ 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 9, $\mathtt{PR}^G_{\mathbf{x}}(u) = p_0$. Hence, A's output will satisfy $\overline{\mathtt{PR}}_{\mathbf{x}}(u) \leq p_0(1+\epsilon)$ 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 10. $p_0(1+\epsilon) < p_1(1-\epsilon)$.

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,

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

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,

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

$$\geq \frac{n-1}{\frac{n-1}{\frac{4}{\alpha}+1}} - \frac{2}{\sqrt{\frac{n-1}{\frac{4}{\alpha}+1}} - 1} \geq \frac{4}{\alpha} + 1 - \frac{2}{\sqrt{\frac{\frac{36}{\alpha}+9}{\frac{4}{\alpha}+1}} - 1} = \frac{4}{\alpha} + 1 - \frac{2}{\sqrt{9}-1} = \frac{4}{\alpha}.$$

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, $\frac{1}{\frac{4m}{\alpha k}+1} > \epsilon$. □

□

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

Theorem 11. *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.*

Proof. We prove the lower bound by a reduction from a variant of the majority problem. In majority-by-a-margin, we are given a vector $\mathbf{x}$ of m bits $x_1, \ldots, x_m$, which is guaranteed to contain either at least $(\frac{1}{2}+\beta)m$ 0's or at least $(\frac{1}{2}+\beta)m$ 1's.

We first use a simple adversarial argument to show that at least $(1-2\beta)m$ queries to $\mathbf{x}$ are needed for computing majority-by-a-margin deterministically.

Proposition 12. *Given a vector x of m bits $x_1, ..., x_m$, which is guaranteed to contain either at least $(\frac{1}{2}+\beta)m$ 0's or at least $(\frac{1}{2}+\beta)m$ 1's, at least $(1-2\beta)m$ queries to $\mathbf{x}$ are needed for determining (deterministically) which of the two cases holds.*

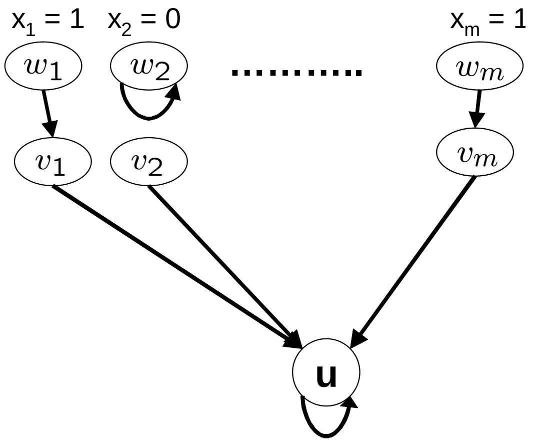


Figure 2: Hard graph (Theorem 11).

Proof. We will prove the proposition using an adversary argument. Suppose, to reach a contradiction, that there exists an algorithm A that solves majority-by-a-margin on inputs of length m with $t < (1-2\beta)m$ queries to the input server. Without loss of generality, we assume A never queries the same bit twice.

The adversary simulates the input server as follows. The adversary provides the algorithm with 0's and 1's alternately. For the first bit queried by A (regardless of which bit position that was actually queried), the adversary returns a 0; for the second bit, the adversary returns a 1; and so on. Now, since the algorithm performs at most $t < (1-2\beta)m$ queries, it must have seen at most $(\frac{1}{2}-\beta)m$ 0's and at most $(\frac{1}{2}-\beta)m$ 1's. Now, if the algorithm outputs "0", the adversary sets the rest of the $m-t$ input bits to be 1, while if the algorithm outputs "1", the adversary sets the rest of the input bits to be 0. In both cases the input created by the adversary is legal (has at least $(\frac{1}{2}+\beta)m$ 0's or at least $(\frac{1}{2}+\beta)m$ 1's). In the former case, the output should be 1, but the algorithm outputs 0, while in the latter case, the output should be 0, but the algorithm outputs 1. This is a contradiction to our assumption that A correctly solves the majority-by-a-margin problem. □

Let $m = \lfloor (n-1)/2 \rfloor$. Note that $n-1 \le 2m+1 \le n$. We reduce majority-by-a-margin to local PageRank approximation as follows. Let S be the maximum number of queries A uses on graphs of size $\le n$. We will use A to construct an algorithm B that computes majority-by-a-margin on inputs of size m and margin $\beta = \epsilon(4+\alpha)/(2\alpha)$ using at most S queries. It would then follow that $S \ge m(1-2\beta) \ge \Omega(n)$.

We map each input vector $\mathbf{x}$ of majority-by-a-margin to an input graph $G_{\mathbf{x}}$ on $n' = 2m+1$ nodes (see Figure 2). $G_{\mathbf{x}}$ contains a tree of depth 2, whose root is u. Level 0 consists of u, level 1 of the m nodes $v_1, \ldots, v_m$, and level 2 of the m nodes $w_1, \ldots, w_m$. $v_1, \ldots, v_m$ and u link to u. For each $i = 1, \ldots, m$, w_i links to v_i, if $x_i = 1$, and links to itself, if $x_i = 0$. Note that $G_{\mathbf{x}}$ is sink-free and that the inverse P-distance converges on it in 2 steps.

For each node y, we denote by $\mathtt{PR}^G_{\mathbf{x}}(y)$ the PageRank of y in the graph $G_{\mathbf{x}}$. We characterize $\mathtt{PR}^G_{\mathbf{x}}(u)$ as follows:

Claim 13.
$$PR^G_{\mathbf{x}}(u) = \frac{1-\alpha}{n'}(1 + \alpha m + \alpha^2|\mathbf{x}|).$$

Proof. Since the out-degree of $G_{\mathbf{x}}$ is 1, all the nodes that have a path to u have influence 1 on u. There is one node (u) of distance 0 from u, m nodes of distance 1, and $|\mathbf{x}|$ nodes of distance

2. Therefore, using Theorem 3,

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

□

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

$$p_0 = \frac{1-\alpha}{n'}\left(1+\alpha m+\alpha^2(\frac{1}{2}-\beta)m\right)$$

and let

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

Given an input $\mathbf{x}$, B simulates A on the graph $G_{\mathbf{x}}$ and on the target node u. The simulation of the link server works as follows: (a) If A probes the link server for u, B returns u,$v_1,\ldots,v_m$ as the in-neighbors and u as the single out-neighbor. (b) If A probes v_i, for $i \in \{1,\ldots,m\}$, B queries x_i; if $x_i = 1$, B returns w_i as the single in-neighbor and u as the single out-neighbor; if $x_i = 0$, B returns only u as the single out-neighbor. (c) If A probes w_i, for $i \in \{1,\ldots,m\}$, B queries x_i; if $x_i = 1$, B returns v_i as the single out-neighbor; if $x_i = 0$, B returns w_i as both the single out-neighbor and single in-neighbor. As each query A sends to the link server requires at most one query to the external server in the simulation, B uses at most S queries. To prove that B is always correct, we analyze two cases.

Case 1: The majority bit of $\mathbf{x}$ is 1. In this case $|\mathbf{x}| \geq (\frac{1}{2}+\beta)m$. Therefore, by Claim 13, $\mathtt{PR}^G_{\mathbf{x}}(u) \geq p_1$. This means that A's output will satisfy $\overline{\mathtt{PR}}_{\mathbf{x}}(u) \geq p_1(1-\epsilon)$. In this case B outputs 1, as needed.

Case 2: The majority bit of $\mathbf{x}$ is 0. In this case $|\mathbf{x}| \leq (\frac{1}{2}-\beta)m$. Therefore, by Claim 13, $\mathtt{PR}^G_{\mathbf{x}}(u) \leq p_0$. Hence, A's output will satisfy $\overline{\mathtt{PR}}_{\mathbf{x}}(u) \leq p_0(1+\epsilon)$. The following claim shows that this value is less than $p_1(1-\epsilon)$, and thus B outputs 0, as needed.

Claim 14. $p_0(1+\epsilon) < p_1(1-\epsilon)$.

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:

$$\frac{p_1-p_0}{p_1+p_0} \;=\; \frac{2\beta\alpha^2 m}{2+2\alpha m+\alpha^2 m}.$$

As $\alpha > 1/2$, $m \geq (n-1)/2$, and $n > 4$, we have $m > 1/\alpha$, and thus $2+2\alpha m < 4\alpha m$. Hence,

$$\frac{2\beta\alpha^2 m}{2+2\alpha m+\alpha^2 m} > \frac{2\beta\alpha^2 m}{4\alpha m+\alpha^2 m} = \frac{2\beta\alpha}{4+\alpha}.$$

By our choice of β ($\beta = \epsilon(4+\alpha)/(2\alpha)$), the RHS is exactly ϵ. □

□

It remains open to determine whether an $\Omega(n)$ lower bound holds for randomized algorithms when the approximation factor is large.

4.2 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 (essentially) 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 8 and 11. 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 15. *Fix any $\alpha \in (\frac{1}{2}, 1)$, $\epsilon \in (0, \frac{1}{2})$, 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 $\Omega(n^{\frac{1+\log\alpha}{2}})$ queries. Furthermore, the maximum in-degree of G is 2 and PageRank converges in $\Omega(\log n)$ iterations on u.*

Remark: The logs in the theorem's proof, as of the rest of the logs in the paper, are of the base of two.

Proof. We prove the lower bound by a reduction from the OR problem. We assume $n > 8 \cdot (\frac{1-\epsilon}{\epsilon} \cdot \frac{1}{\alpha})^{\frac{1}{1+\log\alpha}}$. Let m be the largest power of two which is at most $(\frac{n}{8})^{\frac{1+\log\alpha}{2}} \cdot \sqrt{\frac{2\alpha(1-\epsilon)}{8\epsilon}}$. Let k be the smallest power of two which greater than $(\frac{1}{\alpha} \cdot \frac{4m\epsilon}{1-\epsilon})^{\frac{1}{1+\log\alpha}}$. Let $l_m = \log m$ and let $l_k = \log k$. 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 $S \geq m(1-2\delta) = \Omega(n^{\frac{1+\log\alpha}{2}})$.

We map each input vector $\mathbf{x} = (x_1, \ldots, x_m)$ into a graph $G_{\mathbf{x}}$ on $n' = m(2k+1) - 1$ nodes (see Figure 3). $G_{\mathbf{x}}$ consists of $m+1$ binary trees $T, S_1, \ldots, S_m$. The root of T is u and its leaves are $v_1, \ldots, v_m$. Thus, T has $2m-1$ nodes and its depth is ℓ_m. For each $i = 1, \ldots, m$, the leaves of S_i are $w_{i1}, \ldots, w_{ik}$. Thus, S_i has $2k-1$ nodes and its depth is ℓ_k. If $x_i = 1$, the root of S_i links to v_i, and otherwise it links to itself. Finally, u links to itself. Note that $G_{\mathbf{x}}$ is sink-free and has a maximum in-degree of 2.

For each node y, we denote by $\mathtt{PR}^G_{\mathbf{x}}(y)$ the PageRank of y in the graph $G_{\mathbf{x}}$.

Claim 16.
$$PR^G_{\mathbf{x}}(u) = \frac{1-\alpha}{n'(2\alpha-1)}((2\alpha)^{\ell_m+1} - 1 + |\mathbf{x}|\alpha^{\ell_m}((2\alpha)^{\ell_k+1} - 1)).$$

Proof. As the maximum out-degree of $G_{\mathbf{x}}$ is 1, each node that has a path to u has influence 1 on u. For $t = 0, \ldots, \ell_m$, there are 2^t nodes whose path to u is of length t (the nodes at layer t of T). For each $i = 1, \ldots, m$ and $t = 0, \ldots, \ell_k$, if $x_i = 1$, then there are 2^t nodes at layer t of S_i that have a path of length $\ell_m + t$ to u. Therefore, the total number of nodes that have a path of length $\ell_m + t$ to u is $|\mathbf{x}| \cdot 2^t$. We can now use Theorem 3 to write $\mathtt{PR}^G_{\mathbf{x}}(u)$ as follows:

$$\begin{aligned}
\mathtt{PR}^G_{\mathbf{x}}(u) &= \\
&= \frac{1-\alpha}{n'}\left(\sum_{t=0}^{\ell_m} 2^t \cdot \alpha^t + |\mathbf{x}| \cdot \sum_{t=0}^{\ell_k} 2^t \cdot \alpha^{\ell_m+t}\right) \\
&= \frac{1-\alpha}{n'}\left(\frac{(2\alpha)^{\ell_m+1}-1}{2\alpha-1} + |\mathbf{x}|\alpha^{\ell_m} \cdot \frac{(2\alpha)^{\ell_k+1}-1}{2\alpha-1}\right) \\
&= \frac{1-\alpha}{n'(2\alpha-1)}((2\alpha)^{\ell_m+1} - 1 + |\mathbf{x}|\alpha^{\ell_m}((2\alpha)^{\ell_k+1}-1)).
\end{aligned}$$

□

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

$$p_0 = \frac{1-\alpha}{n'(2\alpha-1)}((2\alpha)^{\ell_m+1}-1)$$

and

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

Given an input $\mathbf{x}$, B simulates A on the graph $G_{\mathbf{x}}$ and on the target node u. The simulation of the link server works as follows: (a) If A probes the link server for any node except for $v_1, \ldots, v_m$ and root$(S_1), \ldots,$ root(S_m), B simply returns the in-neighbors and out-neighbors of that node in its corresponding tree (T or one of the S_i's). (b) If A probes v_i, then B returns its single out-neighbor in T. For figuring out whether root(S_i) is an in-neighbor, B probes its own server for x_i. If $x_i = 1$, root(S_i) is an in-neighbor of v_i, and otherwise it is not. (c) If A probes root(S_i), B returns its two in-neighbors in S_i as in-neighbors. For figuring out whether v_i is an out-neighbor, B queries x_i. After the simulation of A ends, B declares the majority bit to be 1, if A's estimation of $\mathtt{PR}^G_{\mathbf{x}}(u)$ 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 16, $\mathtt{PR}^G_{\mathbf{x}}(u) \geq p_1$. This means that A's output will satisfy $\overline{\mathtt{PR}}_{\mathbf{x}}(u) \geq p_1(1-\epsilon)$ 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 16, $\mathtt{PR}^G_{\mathbf{x}}(u) = p_0$. Hence, A's output will satisfy $\overline{\mathtt{PR}}_{\mathbf{x}}(u) \leq p_0(1+\epsilon)$ 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 17. *$p_0(1+\epsilon) < p_1(1-\epsilon)$.*

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:

$$\begin{aligned}
\frac{p_1-p_0}{p_1+p_0} &= \frac{\alpha^{\ell_m+1}((2\alpha)^{\ell_k+1}-1)}{2\cdot((2\alpha)^{\ell_m+1}-1) + \alpha^{\ell_m+1}((2\alpha)^{\ell_k+1}-1)} \\
&> \frac{\alpha^{\ell_m+1}((2\alpha)^{\ell_k+1}-1)}{2\cdot(2\alpha)^{\ell_m+1} + \alpha^{\ell_m+1}((2\alpha)^{\ell_k+1}-1)} \\
&= \frac{(2\alpha)^{\ell_k+1}-1}{2^{\ell_m+2} + (2\alpha)^{\ell_k+1}-1}.
\end{aligned}$$

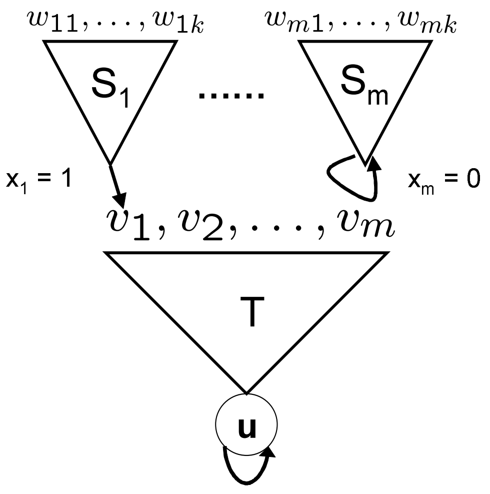


Figure 3: Hard graph (Theorem 15).

Recall that $k = 2^{\ell_k}$ and $m = 2^{\ell_m}$. Therefore,

$$\frac{(2\alpha)^{\ell_k+1} - 1}{2^{\ell_m+2} + (2\alpha)^{\ell_k+1} - 1} = \frac{2k\alpha \cdot \alpha^{\ell_k} - 1}{4m + 2k\alpha \cdot \alpha^{\ell_k} - 1} = \frac{1}{\frac{4m}{2k\alpha \cdot \alpha^{\ell_k} - 1} + 1} = \frac{1}{\frac{4m}{2\alpha \cdot k^{1+\ell_\alpha} - 1} + 1}$$

By the definition of k,

$$\frac{1}{\frac{4m}{2\alpha \cdot k^{1+\log(\alpha)} - 1} + 1} \geq \frac{1}{\frac{4m}{2\alpha(\frac{1}{\alpha} \cdot \frac{4m\epsilon}{1-\epsilon})^{\frac{1}{1+\log(\alpha)} \cdot (1+\log(\alpha))} - 1} + 1} = \frac{1}{\frac{4m}{2\alpha \frac{1}{\alpha} \cdot \frac{4m\epsilon}{1-\epsilon} - 1} + 1} = \frac{1}{\frac{4m}{\frac{8m\epsilon}{1-\epsilon} - 1} + 1}.$$

By the definition of m and the choice of n,

$$\frac{4m\epsilon}{1-\epsilon} \geq \frac{4\epsilon}{1-\epsilon} \cdot \frac{1}{2} \cdot (\frac{n}{8})^{\frac{1+\log(\alpha)}{2}} \sqrt{\frac{2\alpha(1-\epsilon)}{8\epsilon}} > \frac{2\epsilon}{1-\epsilon} (\frac{8(\frac{1-\epsilon}{\epsilon} \cdot \frac{1}{\alpha})^{\frac{1}{1+\log(\alpha)}}}{8})^{\frac{1+\log(\alpha)}{2}} \sqrt{\frac{2\alpha(1-\epsilon)}{8\epsilon}}$$

$$= \frac{2\epsilon}{1-\epsilon} (\frac{1-\epsilon}{\epsilon} \cdot \frac{1}{\alpha})^{\frac{1}{2}} (\frac{2\alpha(1-\epsilon)}{8\epsilon})^{\frac{1}{2}} = \frac{2\epsilon}{1-\epsilon} \cdot \frac{1}{2} \cdot \frac{1-\epsilon}{\epsilon} = 1$$

Thus,

$$\frac{1}{\frac{4m}{\frac{8m\epsilon}{1-\epsilon} - 1} + 1} > \frac{1}{\frac{4m}{\frac{8m\epsilon}{1-\epsilon} - \frac{4m\epsilon}{1-\epsilon}} + 1} = \frac{1}{\frac{4m}{\frac{4m\epsilon}{1-\epsilon}} + 1} = \frac{1}{\frac{1-\epsilon}{\epsilon} + 1} = \frac{\epsilon}{1-\epsilon+\epsilon} = \epsilon.$$

□

□

Also in this case we are able to show an optimal $\Omega(n)$ lower bound for deterministic algorithms:

Theorem 18. *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.*

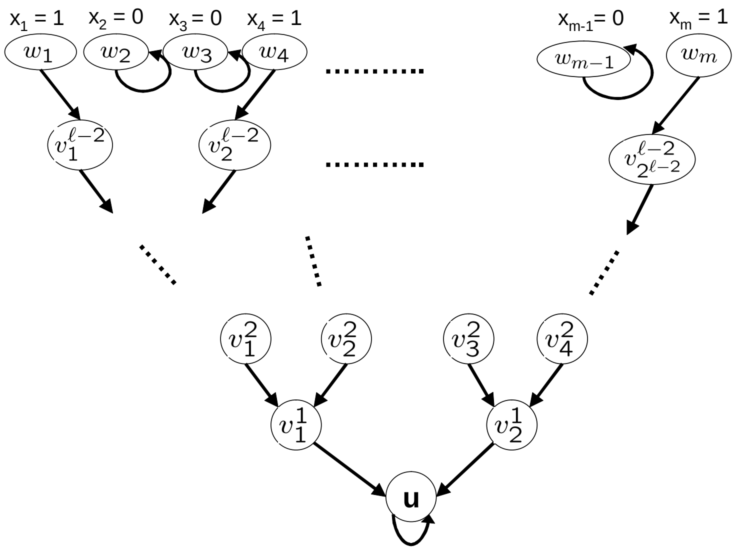


Figure 4: Hard graph (Theorem 18).

Proof. We prove the lower bound by a reduction from majority-by-a-margin. Let $M = 2^\ell$ be the largest power of 2 that is at most n. Note that $n/2 \le M \le n$. Let S be the maximum number of queries A uses on graphs of size $\le n$. We will use A to construct an algorithm B that computes majority-by-a-margin on inputs of size $m = M/2$ and margin $\beta = (\epsilon/2) \cdot (2\alpha+1)/(2\alpha-1)$ using at most $2S$ queries. It would then follow that $S \ge \frac{1}{2}(1 - \epsilon \cdot (2\alpha+1)/(2\alpha-1)) \cdot m = \Omega(n)$.

We map each input vector $\mathbf{x}$ of majority-by-a-margin to an input graph $G_{\mathbf{x}}$ on $n' = M-1 \le n$ nodes (see Figure 4). $G_{\mathbf{x}}$ contains a binary tree of depth $\ell-1$, whose root is u. We denote the nodes at level t of the tree ($t = 0, \ldots, \ell-2$) by $v_1^t, \ldots, v_{2^t}^t$. Note that $u = v_1^0$. We denote the nodes at level $\ell-1$ by $w_1, \ldots, w_m$. The tree is complete, except for nodes at level $\ell-1$. For each $i = 1, \ldots, m$, w_i links to $v_{\lceil i/2 \rceil}^{\ell-2}$, if $x_i = 1$, and links to itself, if $x_i = 0$. Finally, u links to itself. Note that $G_{\mathbf{x}}$ is sink-free and has a maximum in-degree of 2.

For each node y, we denote by $\mathtt{PR}_{\mathbf{x}}^G(y)$ the PageRank of y in the graph $G_{\mathbf{x}}$.

Claim 19.

$$\mathit{PR}_{\mathbf{x}}^G(u) = \frac{1-\alpha}{n'}\left(\frac{(2\alpha)^{\ell-1}-1}{2\alpha-1} + |\mathbf{x}| \cdot \alpha^{\ell-1}\right).$$

As $G_{\mathbf{x}}$ is of out-degree 1, every node that has a path to u has influence 1 on u. For $t = 0, \ldots, \ell-2$, there are 2^t nodes whose path to u is of length t. For $t = \ell-1$, the number of nodes at layer $\ell-1$ that have a path to u is exactly $|\mathbf{x}|$. Hence, the number of nodes whose path to u is of length $\ell-1$ is $|\mathbf{x}|$. We can now use Theorem 3 to write $\mathtt{PR}_{\mathbf{x}}^G(u)$ as follows:

$$\begin{aligned}\mathtt{PR}_{\mathbf{x}}^G(u) &= \frac{1-\alpha}{n'}\left(\sum_{t=0}^{\ell-2} 2^t \cdot \alpha^t + |\mathbf{x}| \cdot \alpha^{\ell-1}\right) \\ &= \frac{1-\alpha}{n'}\left(\frac{(2\alpha)^{\ell-1}-1}{2\alpha-1} + |\mathbf{x}| \cdot \alpha^{\ell-1}\right).\end{aligned}$$

□

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

$$p_0 = \frac{1-\alpha}{n'}\left(\frac{(2\alpha)^{\ell-1}-1}{2\alpha-1} + (\frac{1}{2} - \beta)2^{\ell-1} \cdot \alpha^{\ell-1}\right)$$

and let

$$p_1 = \frac{1-\alpha}{n'}\left(\frac{(2\alpha)^{\ell-1}-1}{2\alpha-1} + (\frac{1}{2}+\beta)2^{\ell-1}\cdot\alpha^{\ell-1}\right).$$

Given an input $\mathbf{x}$, B simulates A on the graph $G_{\mathbf{x}}$ and on the target node u. The simulation of the link server works as follows: (a) If A probes the link server for u, B returns u,v_1^1,v_2^1 as the in-neighbors and u as the single out-neighbor. (b) If A probes a node v_i^t, for $t = 1, \ldots, \ell-3$, B returns v_{2i-1}^{t+1},v_{2i}^{t+1} as the in-neighbors and $v_{\lceil i/2\rceil}^{t-1}$ as the single out-neighbor. (c) If A probes a node $v_i^{\ell-2}$ at layer $\ell-2$, B sends $2i-1$ and $2i$ to its own server; if $x_{2i-1} = 1$, B returns w_{2i-1} as an in-neighbor of $v_i^{\ell-2}$; similarly, if $x_{2i} = 1$, B returns w_{2i} as an in-neighbor of $v_i^{\ell-2}$; in any case B returns $v_{\lceil i/2\rceil}^{\ell-3}$ as the single out-neighbor. (d) If A probes a node w_i, B sends i to its own server; if $x_i = 1$, B returns $v_{\lceil i/2\rceil}^{\ell-2}$ as the single out-neighbor; if the answer is $x_i = 0$, B returns w_i as both the single out-neighbor and single in-neighbor. After the simulation of A ends, B declares the majority bit to be 1, if A's estimation of $\mathtt{PR}_{\mathbf{x}}^G(u)$ is at least $p_1(1-\epsilon)$, and 0 otherwise.

Note that each query A sends to the link server incurs at most two queries to B's server. So B uses a total of at most $2S$ queries. To prove that B is always correct, we analyze two cases.

Case 1: The majority bit of $\mathbf{x}$ is 1. In this case $|\mathbf{x}| \geq (\frac{1}{2}+\beta)2^{\ell-1}$. Therefore, by Claim 19, $\mathtt{PR}_{\mathbf{x}}^G(u) \geq p_1$. This means that A's output will satisfy $\overline{\mathtt{PR}}_{\mathbf{x}}(u) \geq p_1(1-\epsilon)$. In this case B outputs 1, as needed.

Case 2: The majority bit of $\mathbf{x}$ is 0. In this case $|\mathbf{x}| \leq (\frac{1}{2}-\beta)2^{\ell-1}$. Therefore, by Claim 19, $\mathtt{PR}_{\mathbf{x}}^G(u) \leq p_0$. Hence, A's output will satisfy $\overline{\mathtt{PR}}_{\mathbf{x}}(u) \leq p_0(1+\epsilon)$. The following claim shows that this value is less than $p_1(1-\epsilon)$, and thus B outputs 0, as needed.

Claim 20. $p_0(1+\epsilon) < p_1(1-\epsilon)$.

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:

$$\begin{aligned}
\frac{p_1 - p_0}{p_1 + p_0} &= \frac{2\beta(2\alpha)^{\ell-1}}{2\cdot\frac{(2\alpha)^{\ell-1}-1}{2\alpha-1} + (2\alpha)^{\ell-1}} \\
&> \frac{2\beta(2\alpha)^{\ell-1}}{2\cdot\frac{(2\alpha)^{\ell-1}}{2\alpha-1} + (2\alpha)^{\ell-1}} \\
&= \frac{2\beta}{\frac{2}{2\alpha-1}+1} \\
&= 2\beta\cdot\frac{2\alpha-1}{2\alpha+1}.
\end{aligned}$$

By our choice of β, the RHS is exactly ϵ, as needed. □

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, the algorithm proposed by Chen *et al.* [11] is already sufficient for this purpose. In the following we present a novel variant of this algorithm. We explain the difference between the variant and the original algorithm below.

5.1 The algorithm

The algorithm performs a brute force computation of $\mathtt{PR}_r^G(u)$ (see Figure 5). Recall that

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

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:

$$\operatorname{inf}_t(v,u) = \frac{1}{\operatorname{outdeg}(v)}\sum_{w:v\to w}\operatorname{inf}_{t-1}(w,u).$$

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 $t-1$.

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

Figure 5: The local PR approximation algorithm.

Recall that $\mathtt{PR}_r^G(u)$ converges to $\mathtt{PR}^G(u)$ as $r\to\infty$ (Theorem 3). So, ideally, we would like to choose $r = T_\epsilon(G,u)$. Since the algorithm computes $\mathtt{PR}_r^G(u)$, 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 $[(1-\epsilon)\mathtt{PR}^G(u), \mathtt{PR}^G(u)]$. 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 $\mathtt{PR}_t^G(u)$ does not change by much. This latter approach is not guaranteed to provide a good approximation but it works well in practice.

5.2 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 lead to large approximation errors.

5.3 Complexity analysis

Next, we show an upper bound on $T_\epsilon(G, u)$. This bound indicates that we never need to run the brute force algorithm with r bigger than $O(\log(1/\mathtt{PR}^G(u)))$. Note that the minimum PageRank value of any node is at least $\frac{1-\alpha}{|G|}$, and thus $O(\log(1/\mathtt{PR}^G(u))) = O(\log(|G|)$. Hence, if the crawl growth rate is low (e.g., when the graph's maximum in-degree is low), then the query cost of this algorithm is low.

The following notion will be used to quantify the number of nodes the brute force algorithm needs to crawl:

Definition 21. *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 22. *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 23. *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 $r = O(\log \frac{1}{\mathtt{PR}^G(u)})$ is always sufficient (in practice much lower r may be enough as well).

Theorem 24. *Let G be any directed graph and let $u \in G$. Then, for any $\epsilon > 0$,*

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

Proof. Let $r = \lceil \frac{1}{1-\alpha}(\ln \frac{1}{\mathtt{PR}^G(u)} + \ln \frac{2}{\epsilon}) \rceil - 1$. We will show that

$$\frac{\mathtt{PR}^G(u) - \mathtt{PR}^G_t(u)}{\mathtt{PR}^G(u)} < \epsilon.$$

It would follows from Definition 7 that $t \geq T_\epsilon(G, u)$.

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

$$\begin{aligned}
&\frac{\mathtt{PR}^G(u) - \mathtt{PR}^G_t(u)}{\mathtt{PR}^G(u)} \\
&= \frac{\mathtt{PR}^G(u) - \mathbf{P}^{t+1}(1,u) + \mathbf{P}^{t+1}(1,u) - \mathtt{PR}^G_t(u)}{\mathtt{PR}^G(u)} \\
&\leq \frac{|\mathtt{PR}^G(u) - \mathbf{P}^{t+1}(1,u)|}{\mathtt{PR}^G(u)} + \frac{|\mathbf{P}^{t+1}(1,u) - \mathtt{PR}^G_t(u)|}{\mathtt{PR}^G(u)}. \qquad (2)
\end{aligned}$$

We will show that each of the two terms is at most $\epsilon/2$. We start with the first term. According to Sinclair's bound on the pointwise mixing time [49] (see Proposition 2.1, pages 47–48),

$$\frac{|\mathtt{PR}^G(u) - \mathbf{P}^{t+1}(1,u)|}{\mathtt{PR}^G(u)} \leq \frac{\lambda^{t+1}_{\max}}{\mathtt{PR}^G(u)},$$

where $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_{|G|}|$ 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 [24] that for the PageRank matrix, $|\lambda_{\max}| \leq \alpha$, and therefore,

$$\frac{\lambda^{t+1}_{\max}}{\mathtt{PR}^G(u)} \leq \frac{\alpha^{t+1}}{\mathtt{PR}^G(u)}.$$

To bound the latter, we use the following calculation:

Claim 25. *If* $t \geq \frac{1}{1-\alpha}(\ln \frac{1}{PR^G(u)} + \ln \frac{2}{\epsilon}) - 1$, *then*

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

Proof. Since $\alpha \leq 1$, then by the choice of t,

$$\alpha^{t+1} \leq \alpha^{\frac{1}{1-\alpha}(\ln \frac{1}{\mathtt{PR}^G(u)} + \ln \frac{2}{\epsilon})} \quad = \quad \alpha^{\frac{1}{1-\alpha}(\ln \frac{2}{\epsilon \mathtt{PR}^G(u)})} \qquad (3)$$

Let $x = 1 - \alpha$. Since $e^{-x} \geq 1 - x$ for $0 \leq x \leq 1$, $\alpha \leq e^{-(1-\alpha)}$. Hence,

$$\ln\left(\frac{1}{\alpha}\right) \geq \ln e^{1-\alpha} = 1 - \alpha.$$

Plugging this into Equation 3, we have:

$$\begin{aligned}
\alpha^{t+1} &\leq \alpha^{\frac{1}{\ln(1/\alpha)}(\ln \frac{2}{\epsilon \mathtt{PR}^G(u)})} = \alpha^{\log_{1/\alpha}(\frac{2}{\epsilon \mathtt{PR}^G(u)})} \\
&= \frac{1}{\frac{2}{\epsilon \mathtt{PR}^G(u)}} = \frac{\epsilon \mathrm{PR}^G(u)}{2}.
\end{aligned}$$

The desired bound on $\frac{\alpha^{t+1}}{\mathtt{PR}^G(u)}$ immediately follows. □

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

$$|\mathbf{P}^{t+1}(1,u) - \mathtt{PR}^G_t(u)| = \alpha^{t+1}\mathbf{M}^{t+1}(1,u).$$

Since $\mathbf{M}^{t+1}(1,u) \leq 1$,

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

Therefore,

$$\frac{|\mathbf{P}^{t+1}(1,u) - \mathtt{PR}^G_t(u)|}{\mathtt{PR}^G(u)} \leq \frac{\alpha^{t+1}}{\mathtt{PR}^G(u)}.$$

Claim 25 shows that the latter is at most $\epsilon/2$. The theorem follows. □

5.4 Optimizing by pruning

To improve 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 t is small, then only a small fraction of its score eventually propagates to $\texttt{PR}_t^G(u)$ and thus omitting v from the computation of $\texttt{PR}_t^G(u)$ 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 therefore if we prune the crawl at low influence nodes we are unlikely to neglect high influence nodes from the crawl.

The pruning heuristic is implemented by calling the procedure depicted in Figure 6. The procedure removes all nodes whose influence is below some threshold value T from layer t. Thus, these nodes will not be expanded in the next iteration.

```
procedure Prune($t$)
1: for each $v \in \text{layer}_t$ do
2:     if $\alpha^t \inf_t(v) < T$ then
3:         remove $v$ from $\text{layer}_t$
4:     end if
5: end for
```

Figure 6: The pruning procedure.

5.5 Discussion on the pruning error

The problem of the pruning heuristic is that stopping the crawl whenever the PageRank value does not change much does not guarantee an approximation. Figure 7 is an example that shows how pruning a node with influence below the threshold might also cause pruning nodes with influence higher than the threshold.

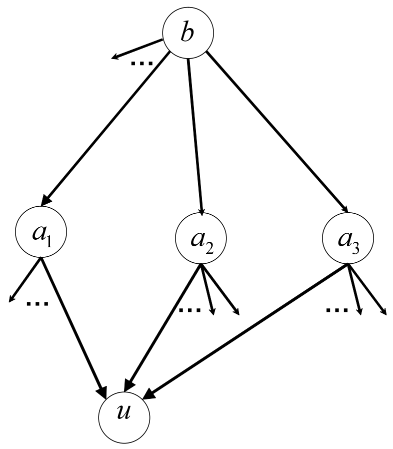


Figure 7: Pruning error example.

Consider $T = 0.01$, $\alpha = 0.85$, $\text{outdeg}(a_1) = 10$, $\text{outdeg}(a_2) = \text{outdeg}(a_3) = 100$, and $\text{outdeg}(b) = 8$. The pruning procedure will remove node a_2 from $layer_1$ since $\alpha \cdot \frac{1}{\text{outdeg}(a_2)} = 0.0085 < T$. The same holds for node a_3. Now, when the algorithm will reach node b in the next layer, it will decide to prune it also since

$$\alpha^2 \cdot \frac{1}{\text{outdeg}(b) \cdot \text{outdeg}(a_1)} = 0.00903 < T.$$

Without the pruning of nodes a_2 and a_3, the influence of node b would have been higher and the node would not have been pruned since

$$\alpha^2 \cdot (\frac{1}{\text{outdeg}(b) \cdot \text{outdeg}(a_1)} + \\ \frac{1}{\text{outdeg}(b) \cdot \text{outdeg}(a_2)} + \frac{1}{\text{outdeg}(b) \cdot \text{outdeg}(a_3)}) = \\ 0.01084 > T.$$

This example shows that the pruning heuristic might cause a significant error in the PageRank calculation and the result cannot be truly reliable. In spite of that, Chen *et al.* [11] showed that most of the target nodes have very small errors, while a few others have fairly large errors, which means the example we have presented is rare on the web graph.

6 PageRank vs. Reverse PageRank

n 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) low 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 low in-degree and is therefore more suitable for local PageRank approximation, using the algorithm we have presented in chapter **??**. We also show empirically that this algorithm performs better on the reverse web graph than on the web graph.

6.1 Experimental setup

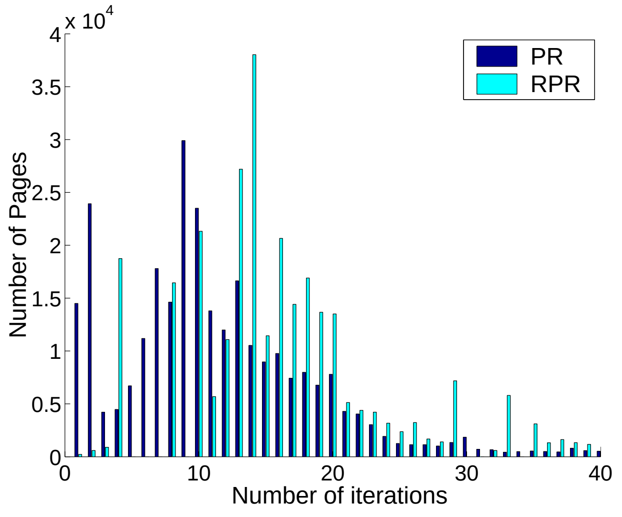
We base our empirical analysis on a 280,000 page crawl of the `www.stanford.edu` domain performed in September 2002 by the WebBase project[6] and a 22,000 page crawl of the `www.cnn.com` site we performed in February 2007. We built the adjacency matrices of these graphs, which enabled us to calculate 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.* [29] 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.* [15] 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.

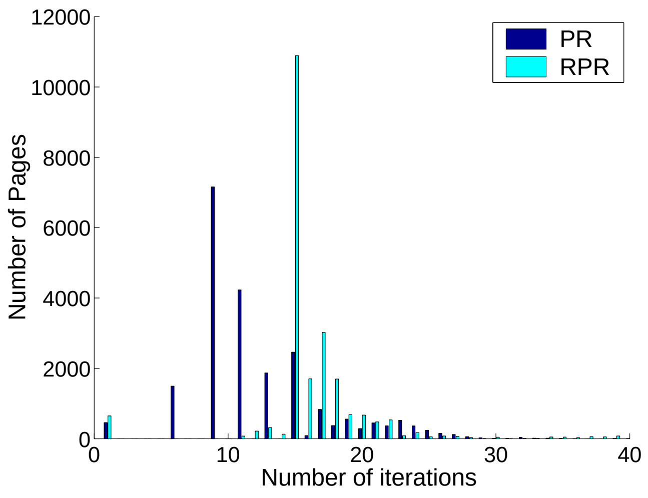
6.2 Convergence rate

We start by analyzing the PageRank convergence rate. Kamvar, Haveliwala, and Golub [28] 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 8, 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` and `cnn.com` graphs on which PageRank and Reverse PageRank converge in t iterations. We determine that PageRank converges on a page u in t steps, if $\frac{|PR_t(u) - PR_{t-1}(u)|}{PR_{t-1}(u)} < 10^{-3}$.

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



(a) `stanford.edu`



(b) `cnn.com`

Figure 8: Number of iterations until convergence for PageRank and Reverse PageRank.

As can be seen from the results, Reverse PageRank converges only slightly slower than PageRank: in less than 20 iterations, 80% of the `stanford.edu` nodes and 90% of the `cnn.com` nodes converges.

6.3 Crawl growth rate

Previous studies [48] 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. In the `cnn.com` graph, the maximum in-degree is 7,666, while its maximum out-degree is 64.

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.

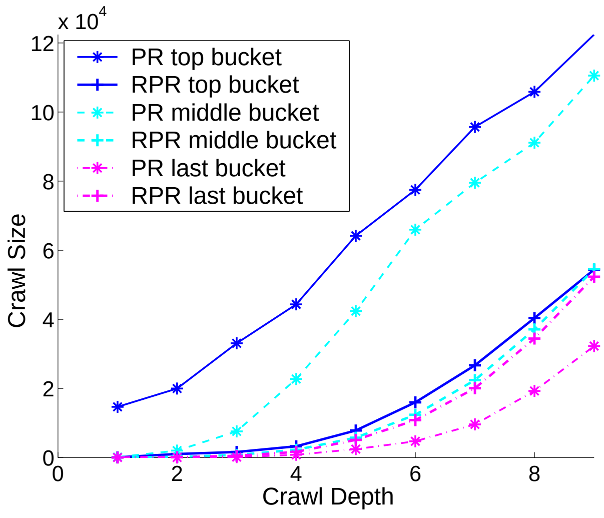
In Figure 9, we plot the average size of a backward BFS crawl as a function of the crawl depth for the `stanford.edu` and for `cnn.com` graphs and for their reverse graphs. To create the plot for the regular graph, we selected random nodes from the graph as follows. We ordered all the nodes in the graph 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 $t = 1, \ldots, 9$, we calculated the average number 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 9 the results for the top bucket (12 pages with highest PageRank/Reverse PageRank), the middle bucket (768 pages with intermediate PageRank/Reverse PageRank) and the last bucket (85,000 pages with lowest PageRank/Reverse PageRank).

The graphs 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 PageRank/Reverse PageRank. For example, for `stanford.edu` 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%). Also for `cnn.com` the results were similar. 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.

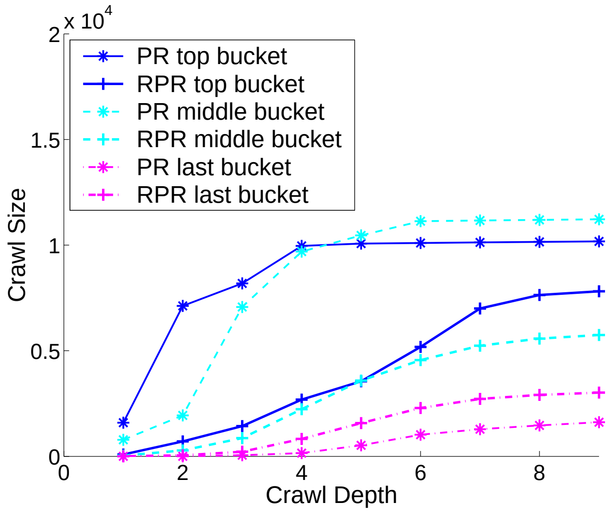
6.4 Algorithm's performance

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 10.

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, for `stanford.edu`, 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. For `cnn.com` the results are even better, with PageRank cost ten times higher than Reverse PageRank on the first bucket. On the other hand, for the low ranked nodes, the increased crawl growth rate on the reverse graph

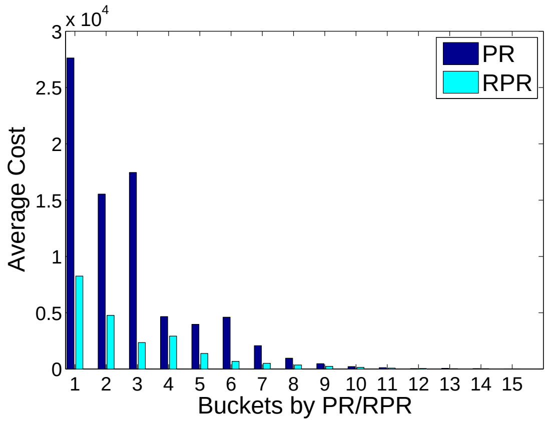


(a) `stanford.edu`

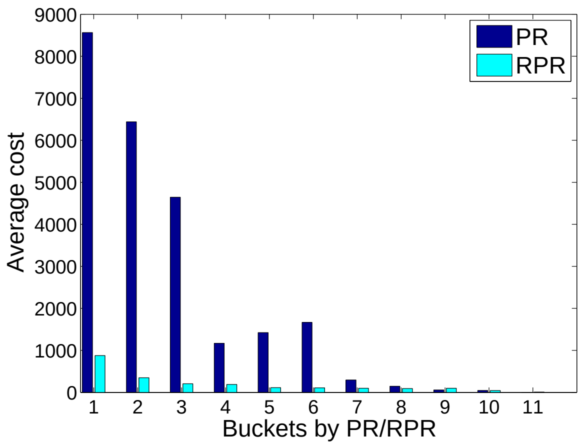


(b) `cnn.com`

Figure 9: Average growth rates of backward BFS crawls at the graph and its reverse.



(a) `stanford.edu`



(b) `cnn.com`

Figure 10: Average cost of the pruning algorithm by PageRank/Reverse PageRank values. Results of runs on the graph and its reverse.

and the regular graph are almost the same. For example, the average cost for `stanford.edu` 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 [22], to detect highly influential nodes in social networks [26], and to find hubs in the web graph [19]. In this chapter 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 [37], such as the power law degree distribution and the gap between in- and out- degrees. As we show below, the same holds for the taxonomy graph extracted from the Open Directory Project.

7.1 Influencers in social networks

The optimization problem of selecting the most influential members is one of the most well studied problems in social networking [47, 16, 31], introduced by Domingos and Richardson [16].

They presented a probabilistic model for the way members of a social network influence each other. They also presented the following algorithmic problem. Suppose that we have data on a social network and on the way members of the network influence each other. Suppose also that we would like to market a new product that we hope will be adopted by a large fraction of the network's members. By initially targeting a few "influential" members (for example, by giving them free samples of the product), we can trigger a cascade of influence in which individuals recommend the product to their friends, leading eventually to a large number of individuals trying it. How should we choose the few key individuals to be used as seeds for this process? More formally, given a social network and a number k, find the k "seed" nodes, S_k, that maximize the expected number of "active" nodes when starting with S_k as a set of initial active nodes. It is known that finding the optimal set is NP-hard [31].

In [32], Kempe, Kleinberg, and Tardos show that a natural greedy hill-climbing strategy provides a solution, which is always at least $1 - 1/e$ ($\approx 63\%$) of the optimal solution. Unfortunately, this algorithms runs in $O(n^2)$ time (where n is the size of the network), and therefore it is not very feasible for big networks. It is still open whether sub-quadratic time algorithms exist for this problem.

In [26], Java *et al.* suggest using Reverse PageRank as a heuristic for selecting the seed influential active nodes. This is motivated by the fact that nodes with high Reverse PageRank have short paths to many other nodes in the network, and moreover they are frequently the only gateways to these other nodes. Note that Reverse PageRank can be approximated in almost linear time for graphs on which it convergence quickly. Reverse PageRank can also be locally approximated as already shown above.

Experimental results. To demonstrate this method we ran experiments on the network of friendship links from the site LiveJournal[7], constructed from a crawl of this site performed in February 2006, as our data. Each node in LiveJournal corresponds to a user who has made his or her blog public through the site; each user can also declare friendship links to other users.

[7]`http://www.livejournal.com`. We used the same crawl that was used in [2].

These links are the edges of the social network. The data contains 3.5 million nodes and 47 million edges.

We compare a number of strategies for choosing an initial seed set of k nodes: k random nodes, the top k Reverse PageRank nodes, the top k PageRank nodes, and the k nodes of highest out-degree. For simplicity, we assume that every node that is reachable from a seed by a short path eventually becomes active (this is indeed a property shared by most of the probabilistic models for influence propagation).

For each seed selection method and for varying values of k and i, we measured the (relative) size of G^i, where G^i is the set of nodes reachable from the k seeds by paths of length at most i. We view G^i as the set of nodes that become active. As can be seen in Figure 11, our results reconfirm the observations of Java *et al.* Reverse PageRank outperforms all other strategies we studied. Surprisingly, Reverse PageRank even defeats the out-degree measure for G^1. This can be attributed to the fact that the marginal contribution of each node in the out-degree measure is lower than in the Reverse PageRank measure as a result of the overlap between the sets of nodes pointed by high out-degree nodes. Reverse PageRank measure does not suffer from this weakness since high in-degree nodes split their influence between their parents. Thus, nodes pointing exclusively to other nodes will get higher rank than nodes that share their children with other parents.

7.2 Hub web pages

There are a number of reasons for users to prefer navigation as a search tactic, as opposed to using direct search: difficulty in formulating appropriate queries, broad search tasks, and need to understand the surrounding context.

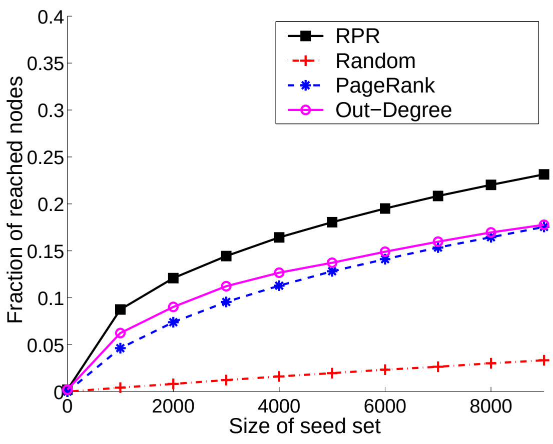
The notion of hub web pages was first introduced by Kleinberg in the framework of the HITS algorithm [33] as means for ranking search results. Kleinberg defined the hub score of a page as the sum of the authority scores of the pages it links to, where authoritativeness represents the relevance of the page to the search topic. In [19], Fogaras suggested using (topic-sensitive) Reverse PageRank for measuring hub scores. In some sense, this generalizes Kleinberg's measure, as pages with high Reverse PageRank tend to have short paths (of length possibly greater than 1) to many authorities. Fogaras experimented with his techniques on a crawl of the .ie (Ireland) country domain.

Recently, Pandit and Olston [43], proposed *navigation-aided retrieval* as a new mode retrieval, where the search engine does not necessarily return the most relevant documents, but rather the best starting points for navigation that will eventually lead to the desired relevant documents. We observe that the more general definition of hub web pages, due to Fogaras, seems very suitable in this scenario, as pages with high topic-sensitive Reverse PageRank pages lead to many relevant documents via short paths.

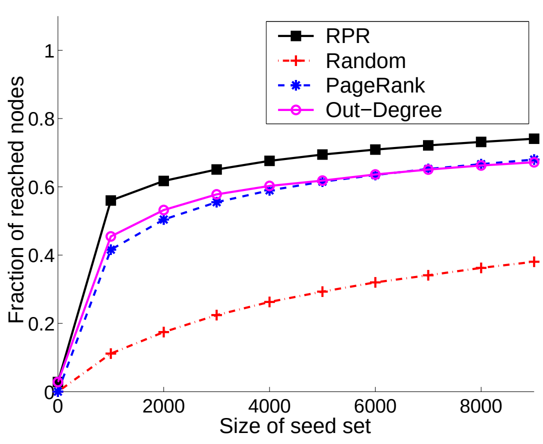
Experimental results. In order to gauge the quality of Reverse PageRank as a hub scoring measure, we built a meta-search engine over Yahoo! Search. Given a user query, we submitted the query to Yahoo! Search, using their developer API[8], calculated the Reverse PageRank of the top 100 matches using the local Reverse PageRank algorithm, and sorted the results by the Reverse PageRank values. To measure in-degrees of pages in the local Reverse PageRank algorithm, we used the Yahoo! Search API again. When expanding the subgraph around the target node, we considered only pages that contain the search terms. Thus, in effect we calculated the Reverse PageRank not on the whole web graph, but rather on the graph spanned by pages that contain the search terms.[9]

[8]http://developer.yahoo.com.

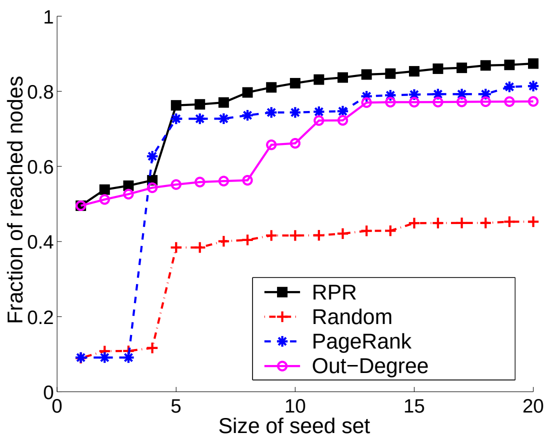
[9]Strictly speaking, this not entirely correct, as the in-degrees returned by Yahoo! take into account also pages



(a) G^1 - Reachable nodes at distance 1.

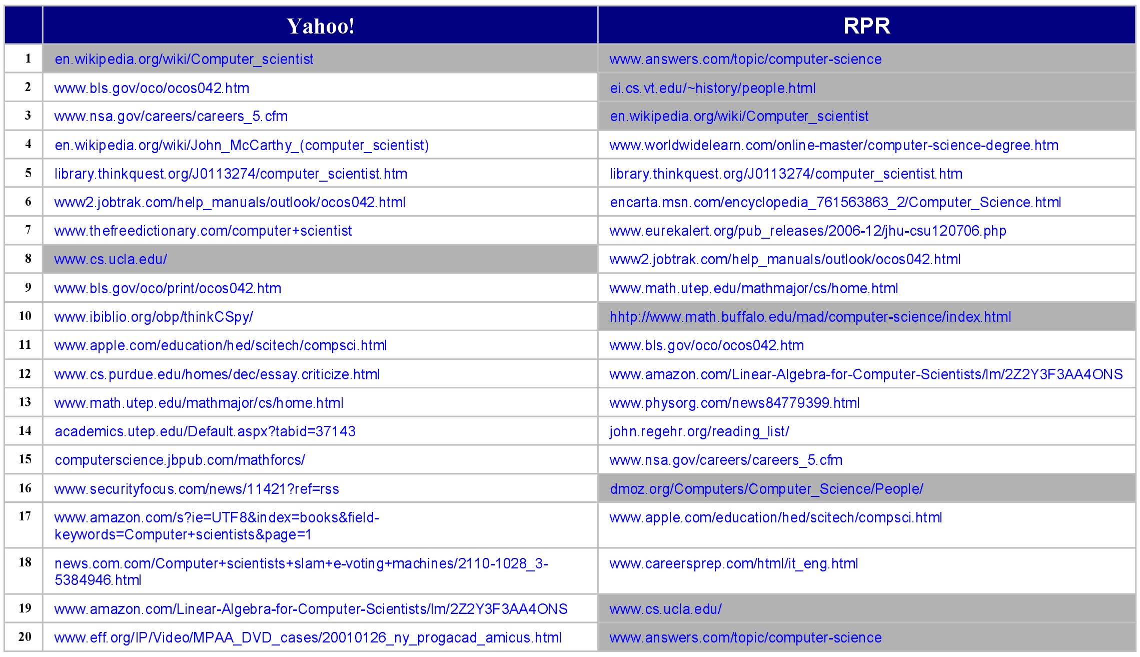


(b) G^2 - Reachable nodes at distance 2.



(c) G^4 - Reachable nodes at distance 4.

Figure 11: Influence in Social Networks - LiveJournal nodes reachable at varying distances from seed nodes.



	Yahoo!	RPR
1	en.wikipedia.org/wiki/Computer_scientist	www.answers.com/topic/computer-science
2	www.bls.gov/oco/ocos042.htm	ei.cs.vt.edu/~history/people.html
3	www.nsa.gov/careers/careers_5.cfm	en.wikipedia.org/wiki/Computer_scientist
4	en.wikipedia.org/wiki/John_McCarthy_(computer_scientist)	www.worldwidelearn.com/online-master/computer-science-degree.htm
5	library.thinkquest.org/J0113274/computer_scientist.htm	library.thinkquest.org/J0113274/computer_scientist.htm
6	www2.jobtrak.com/help_manuals/outlook/ocos042.html	encarta.msn.com/encyclopedia_761563863_2/Computer_Science.html
7	www.thefreedictionary.com/computer+scientist	www.eurekalert.org/pub_releases/2006-12/jhu-csu120706.php
8	www.cs.ucla.edu/	www2.jobtrak.com/help_manuals/outlook/ocos042.html
9	www.bls.gov/oco/print/ocos042.htm	www.math.utep.edu/mathmajor/cs/home.html
10	www.ibiblio.org/obp/thinkCSpy/	hhttp://www.math.buffalo.edu/mad/computer-science/index.html
11	www.apple.com/education/hed/scitech/compsci.html	www.bls.gov/oco/ocos042.htm
12	www.cs.purdue.edu/homes/dec/essay.criticize.html	www.amazon.com/Linear-Algebra-for-Computer-Scientists/lm/2Z2Y3F3AA4ONS
13	www.math.utep.edu/mathmajor/cs/home.html	www.physorg.com/news84779399.html
14	academics.utep.edu/Default.aspx?tabid=37143	john.regehr.org/reading_list/
15	computerscience.jbpub.com/mathforcs/	www.nsa.gov/careers/careers_5.cfm
16	www.securityfocus.com/news/11421?ref=rss	dmoz.org/Computers/Computer_Science/People/
17	www.amazon.com/s?ie=UTF8&index=books&field-keywords=Computer+scientists&page=1	www.apple.com/education/hed/scitech/compsci.html
18	news.com.com/Computer+scientists+slam+e-voting+machines/2110-1028_3-5384946.html	www.careersprep.com/html/it_eng.html
19	www.amazon.com/Linear-Algebra-for-Computer-Scientists/lm/2Z2Y3F3AA4ONS	www.cs.ucla.edu/
20	www.eff.org/IP/Video/MPAA_DVD_cases/20010126_ny_progacad_amicus.html	www.answers.com/topic/computer-science

Figure 12: Results for the query "computer scientists" according to Yahoo! Search and Reverse PageRank ranking.

Figures 12 and 13 present our results for several broad-topic queries. We compared the original Yahoo! Search results with the Reverse PageRank sorted results. Figure 12 shows the top 20 results for the query "computer scientists". The results that were classified as hubs (see the classification heuristic below), are shaded. Among the top 20 original Yahoo! Search results, only results 1 and 8 are hubs. On the other hand, among the top 20 Reverse PageRank results, results 1,2,3,10,16,19, and 20 were classified as hubs. For example, the first and the third results are articles in `http://www.answers.com` and `http://en.wikipedia.org` that link to many other related articles (ranked as 39 and 1 by Yahoo!), and the second result (`http://ei.cs.vt.edu/~history/people.html`) is a hub of many computer scientists (ranked as 24 by Yahoo!).

Figure 13 depicts the fraction of hub results for the following queries 1. "computer scientists" 2. "global warming" 3. "folk dancing" 4. "queen Elizabeth". For each query, we counted the number of hub results among the top 20 results. To classify whether a page is a hub or not, we used the following heuristic: a page p is a hub, if at least 5 of its out-neighbors that are not in the same domain contain the search terms. As can be seen, the fraction of hub results is higher in the Reverse PageRank ranking for all queries.

Remarks:

1. Note that Yahoo! generally tries to optimize the "authoritativeness" of the top ranking results, rather than their hub scores, so it is not surprising the density of hubs in its top results is not very high. The comparison made above just tries to demonstrate that the RPR-induced ranking indeed promotes hubs over authorities.

2. We used a very simple heuristic to classify hubs. It was used only to give an indication to the ability of RPR measure to promote hubs in the rank of query results. Examining RPR according to a more accurate hub score definition is left for a future work.

that do not contain the search terms.

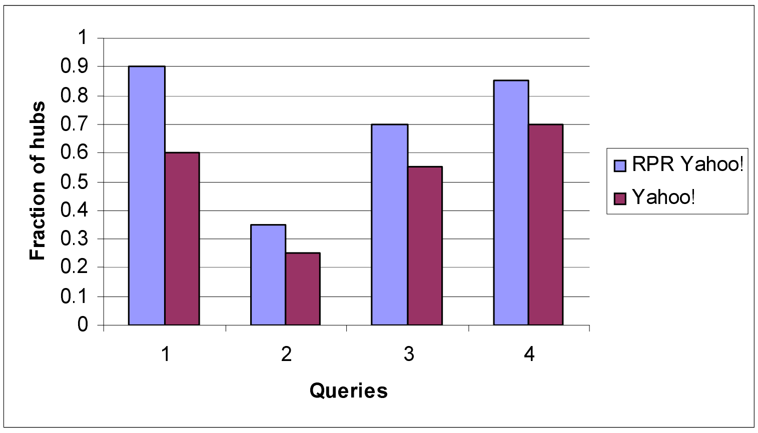


Figure 13: Fraction of hubs in the top 20 results for the queries: 1. "computer scientists" 2. "global warming" 3. "folk dancing" 4. "queen Elizabeth".

7.3 Finding crawl seeds

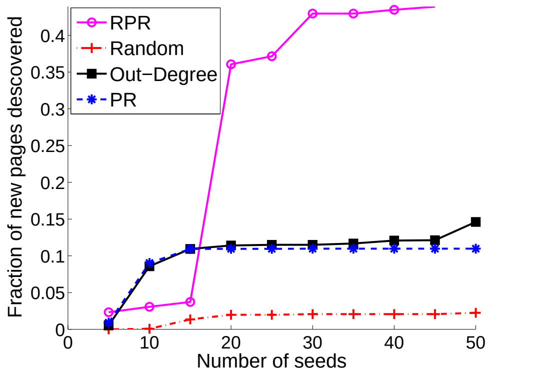
A crawler is a program, commonly used by a search engine, that retrieves Web pages. Roughly, a crawler starts off with initial page, retrieves it, extracts all URLs in it, and adds them to a queue of URLs to be scanned. Then the crawler retrieves a URL from the queue (in some order), and repeats the process. Every page that is scanned is given to the search engine that saves it, creates an index, possibly summarizing and analyzing the content of the page. Since the crawler must deal with huge volumes of data and it has limited resources, design of a good crawler presents many challenges. The main challenge for the crawler is how to decide which URLs to scan and in which order such that "important" pages are scanned first. There are some known crawling algorithms such as Breadth-First [45], Best-First [12], PageRank [12], Shark-Search [25], and InfoSpiders [41].

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.* [13] 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.* formally define the overhead of a crawler to be the average number of old pages it needs to refetch per new pages being discovered. 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.

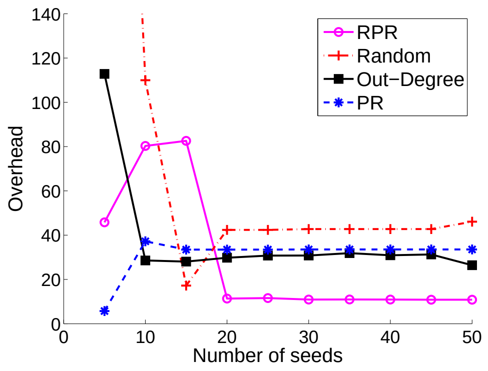
Selecting seeds using Reverse PageRank. We show that RPR is an effective strategy for finding good seeds. Our algorithm simply chooses the nodes with highest Reverse PageRank values to be the seed set. The intuition behind this is the following. A page p has high RPR if many pages are reachable from 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.

Experimental results. To evaluate this seed selection strategy, we used two 1 million page Stanford WebBase crawls[10]. The two crawls are of the same sites and were conducted one week

[10]`http://www-diglib.stanford.edu/~testbed/doc2/WebBase`.



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



(b) Overhead versus the number of seeds.

Figure 14: 4-level BFS crawl.

apart in May 2006. The later crawl consists of 132,000 new pages. We compared four seed selection strategies: the k pages with highest 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 traversal for t levels starting from these seeds on the second crawl. Figure 14 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.

7.4 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. [51]).

In the experiments below, we focus on measuring semantic relatedness between concepts represented as nodes in the Open Directory Project[11] (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 [51] 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 [20]. 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 $\texttt{RPR}_x$ and $\texttt{RPR}_y$. $\texttt{RPR}_x$ is the personalized Reverse PageRank vector of the ODP graph

[11]`http://www.dmoz.org.`

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 $\texttt{RPR}_x(a)$ 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 $\texttt{RPR}_x$ 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 $\texttt{RPR}_x(y)$ and the measure of x as a sub-concept of y is the score $\texttt{RPR}_y(x)$; (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 $\texttt{RPR}_x$ and $\texttt{RPR}_y$. At first glance, RPR similarity seems more accurate than RPR, but RPR has a computational advantage since we can calculate $\texttt{RPR}_x(y)$ 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* [14]. 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 on the ODP. Another graph-based measure of semantic relatedness is the *path-based measure* [46], 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 wanted to verify that the reverse ODP graph admits the two conditions that allow 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 15(a) shows the ordering of the test concepts by relatedness to the main concept "Einstein" using each one of the techniques. Table 15(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	Sodoku	Helicopter
Sodoku	Pizza	Sodoku

(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 15: 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

Human judgement	RPR	RPR similarity	Path-based
Software (8.5)	Software	Software	Laboratory
Keyboard (7.58)	Internet	Internet	Keyboard
Internet (7.62)	Keyboard	Keyboard	News
Laboratory (6.78)	News	Laboratory	Software
News (4.47)	Laboratory	News	Internet
	0.6	0.6	-0.4

(a) Relatedness to “Computer”.

Human	RPR	RPR relatedness	Path-based
Lobster (8.7)	Lobster	Lobster	Food
Food (8.34)	Food	Food	Lobster
Sea (7.47)	Sea	Sea	Sea
	1	1	0.33

(b) Relatedness to “Seafood”.

Figure 16: Relatedness between concepts.

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, Issac” 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 local approximation algorithm to find semantic relatedness.

In the second experiment, we used human judgments extracted from the standard dataset WordSimilarity-353 [18], to evaluate the quality of the relatedness measures on the ODP graph. This test collection, available on the web[12], contains English word pairs along with human-assigned similarity judgments. For the terms “Computer” and “Seafood”, which appear as nodes at the ODP taxonomy, we selected from the WordSimilarity-353 dataset all the pairs one of whose terms was “Computer” or “Seafood” and whose other term was also a node in the ODP taxonomy. We then compared the ranking induced by the human judgments on these other terms with the semantic relatedness scores produced by Reverse PageRank-based rankings and the path-based measure. The results, depicted in Tables 16(a) and 16(b), show the rankings obtained under each of the above techniques, as well as their correlation with the human-based ranking (using the Kendall-Tau coefficient). As it can be seen from the tables, the Reverse PageRank measures have strong correlation with the human judgments, while the path-based method does not.

7.5 TrustRank

Another known use of Reverse PageRank is in the context of TrustRank, first introduced by Gyöngyi *et al.* in [22]. The authors show that spam sites can be pushed down in PageRank ordering if we personalize PageRank using a few trusted hub sites. The basic assumption underlying TrustRank is that good pages usually point to good pages and seldom have links to spam pages. Therefore, after convergence good sites will have relatively high trust scores, while spam sites will have poor trust scores.

Note that TrustRank provides meaningful scores only to nodes that are reachable from the trusted seeds. Therefore, it is desirable to choose seeds that have high “coverage”, i.e., ones from which many other pages are reachable by short paths. For this reason, Gyöngyi *et al.* use Reverse PageRank to choose the trusted seeds. They rank all web pages by their Reverse PageRank

[12]`http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/wordsim353.html`.

scores and then manually pick among the top 1,250 the ones that seem indeed trusted.

8 Conclusions and future work

In this thesis 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.* [11] 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 [4], the copying model [36], etc.

Finally, we have presented applications of Reverse PageRank, two of them are novel. One of the applications is finding influencers in social networks. We have demonstrated the Reverse PageRank measure on blogs networks. It may be interesting to check this measure on other social networks such as *Facebook*[13]. Another application we have presented is finding hub Web pages. We have compared the Reverse PageRank ranking to Yahoo! ranking. For a future work we leave to compare the Reverse PageRank ranking to the HITS algorithm.

Our first novel 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 [12].

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*[14] graph.

9 Acknowledgment

We wish to thank Lars Backstrom for letting us use LiveJournal crawl.

References

[1] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *Proceedings of the 12th International Conference on World Wide Web (WWW)*, pages 280–290, 2003.

[13] `http://www.facebook.com/`.

[14] `http://www.wikipedia.org/`.

[2] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 181–190, 2007.

[3] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Sampling algorithms: Lower bounds and applications. In *Proceedings of the 33th ACM Symposium on Theory of Computing (STOC)*, pages 266–275, 2001.

[4] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[5] P. Berkhin. A survey on PageRank computing. *Internet Mathematics*, 2(1):73–120, 2005.

[6] M. Bianchini, M. Gori, and F. Scarselli. Inside PageRank. *ACM Transactions on Internet Technology (TOIT)*, 5(1):92–128, 2005.

[7] M. Brinkmeier. PageRank revisited. *ACM Transactions on Internet Technology (TOIT)*, 6(3):282–301, 2006.

[8] A. Z. Broder, R. Lempel, F. Maghoul, and J. O. Pedersen. Efficient PageRank approximation via graph aggregation. *Information Retrieval*, 9(2):123–138, 2006.

[9] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

[10] 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.

[11] Y. Chen, Q. Gan, and T. Suel. Local methods for estimating PageRank values. In *Proceeding of the 13th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 381–389, 2004.

[12] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

[13] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins. The discoverability of the web. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 421–430, 2007.

[14] J. Dean and M. R. Henzinger. Finding related pages in the World Wide Web. *Computer Networks*, 31(11–16):1467–1479, 1999.

[15] S. Dill, R. Kumar, K. Mccurley, S. Rajagopalan, D. Sivakumar, and A. Tomkins. Self-similarity in the web. *ACM Transactions on Internet Technology (TOIT)*, 2(3):205–223, 2002.

[16] P. Domingos and M. Richardson. Mining the network value of customers. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 57–66, 2001.

[17] N. Eiron, K. S. McCurley, and J. A. Tomlin. Ranking the web frontier. In *Proceedings of the 13th International Conference on World Wide Web (WWW)*, pages 309–318, 2004.

[18] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing search in context: the concept revisited. In *Proceedings of the 10th International Conference on World Wide Web (WWW)*, pages 406–414, 2001.

[19] D. Fogaras. Where to start browsing the Web? In *Proceeding of the 3rd International Conference on Innovative Internet Community Systems (I2CS)*, pages 65–79, 2003.

[20] E. Gabrilovich and S. Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 250–257, 2007.

[21] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2006.

[22] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating Web Spam with TrustRank. In *Proceeding of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 576–587, 2004.

[23] T. H. Haveliwala. Topic-sensitive PageRank: a context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796, 2003.

[24] T. H. Haveliwala and S. D. Kamvar. The second eigenvalue of the Google matrix. Technical report, Stanford University, 2003.

[25] M. Herscovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. The shark-search algorithm. an application: tailored Web site mapping. In *Proceedings of the 7th international conference on World Wide Web (WWW)*, pages 317–326, 1998.

[26] A. Java, P. Kolari, T. Finin, and T. Oates. Modeling the spread of influence on the Blogosphere. Technical report, University of Maryland, Baltimore County, 2006.

[27] G. Jeh and J. Widom. Scaling personalized Web search. In *Proceedings of the 12th International Conference on World Wide Web (WWW)*, pages 271–279, 2003.

[28] S. Kamvar, H. Haveliwala, and G. Golub. Adaptive methods for the computation of PageRank. *Linear Algebra and its Applications*, 386:51–65, 2004.

[29] S. Kamvar, T. Haveliwala, and G. Golub. Adaptive methods for the computation of pagerank, 2003.

[30] 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.

[31] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.

[32] D. Kempe, J. Kleinberg, and E. Tardos. Influential nodes in a diffusion model for social networks. In *Proceeding of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1127–1138, 2005.

[33] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[34] C. Kohlschütter, P. A. Chirita, and W. Nejdl. Efficient parallel computation of PageRank. In *Proceedings of the 28th European Conference on Information Retrieval (ECIR)*, pages 241–252, 2006.

[35] G. Kollias and E. Gallopoulos. Asynchronous computation of PageRank computation in an interactive multithreading environment. In *Web Information Retrieval and Linear Algebra Algorithms*, 2007.

[36] 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.

[37] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The web and social networks. *Computer*, 35(11):32–36, 2002.

[38] R. Lempel and S. Moran. Rank-stability and rank-similarity of link-based web ranking algorithms in authority-connected graphs. *Information Retrieval*, 8(2):245–264, 2005.

[39] F. McSherry. A uniform approach to accelerated pagerank computation. In *Proceedings of the 14th International Conference on World Wide Web (WWW)*, pages 575–582, 2005.

[40] F. McSherry. Full Web PageRanking On a Laptop. Georgia Tech Workshop on Complex Networks and their Applications (DIMACS), 2007.

[41] F. Menczer. ARACHNID: Adaptive retrieval agents choosing heuristic neighborhoods for information discovery. In *Proceedings of the 14th International Conference on Machine Learning (ICML)*, pages 227–235, 1997.

[42] 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.

[43] S. Pandit and C. Olston. Navigation-Aided Retrieval. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 391–400, 2007.

[44] J. X. Parreira, D. Donato, S. Michel, and G. Weikum. Efficient and decentralized PageRank approximation in a Peer-to-Peer Web search network. In *Proceeding of the 32th International Conference on Very Large Data Bases (VLDB)*, pages 415–426, 2006.

[45] B. Pinkerton. Finding what people want: Experiences with the WebCrawler. In *Proceedings of the 2th International Conference on World Wide Web (WWW)*, volume 18(6), 1994.

[46] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man and Cybernetics*, 19(1):17–30, 1989.

[47] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 61–70, 2002.

[48] 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. *ACM Transactions on the Web (TWEB)*, 1(2), 2007.

[49] A. Sinclair. *Algorithms for Random Generation and Counting: a Markov Chain Approach.* Birkhauser Verlag, Basel, Switzerland, 1993.

[50] A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.

[51] M. Strube and S. P. Ponzetto. WikiRelate! computing semantic relatedness using Wikipedia. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI)*, pages 1419–1424, 2006.

[52] Y. Wang and D. J. DeWitt. Computing PageRank in a distributed Internet search engine system. In *Proceeding of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 420–431, 2004.