

# Local Computation of PageRank: the Ranking Side

Marco Bressan  
Department of Information Engineering  
University of Padova  
bressanm@dei.unipd.it

Luca Pretto  
Department of Information Engineering  
University of Padova  
pretto@dei.unipd.it

## ABSTRACT

Imagine you are a social network user who wants to search, in a list of potential candidates, for the best candidate for a job on the basis of their PageRank-induced importance ranking. Is it possible to compute this ranking for a low cost, by visiting only small subnetworks around the nodes that represent each candidate? The fundamental problem underpinning this question, i.e. computing locally the PageRank ranking of  $k$  nodes in an  $n$ -node graph, was first raised by Chen et al. (CIKM 2004) and then restated by Bar-Yossef and Mashiach (CIKM 2008). In this paper we formalize and provide the first analysis of the problem, proving that any local algorithm that computes a correct ranking must take into consideration  $\Omega(\sqrt{kn})$  nodes – even when ranking the top  $k$  nodes of the graph, even if their PageRank scores are “well separated”, and even if the algorithm is randomized (and we prove a stronger  $\Omega(n)$  bound for deterministic algorithms). Experiments carried out on large, publicly available crawls of the web and of a social network show that also in practice the fraction of the graph to be visited to compute the ranking may be considerable, both for algorithms that are always correct and for algorithms that employ (efficient) local score approximations.

**Categories and Subject Descriptors:** H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval; G.2.2 [Discrete Mathematics]: Graph Theory – Graph algorithms

**General Terms:** Theory, Algorithms, Experimentation

**Keywords:** IR theory: ranking, link and graph mining, web and social media search, local computation, PageRank

## 1. INTRODUCTION

This work tackles the problem of computing locally the ranking induced by PageRank on  $k$  target nodes of an  $n$ -node graph, studying if the correct ranking can be obtained for a low cost, i.e. by visiting only small subgraphs around those target nodes. Previous work shows that computing

PageRank *scores* locally is infeasible (i.e. the cost is not bounded by  $\text{polylog}(n)$ ) in the worst case; one could think, however, that the *ranking* provided by the algorithm (which is the only aspect that matters for many applications) might be obtained “more easily” than the output scores: after all, the correct ranking can be reached with scores even vastly different from the correct ones. Instead, it turns out that the ranking problem is, in a sense, as non-local as the score computation problem: in a nutshell, we prove that no deterministic algorithm exists which guarantees the correct ranking by visiting less than a number of nodes linear in the size of the input graph, and no randomized algorithm exists which guarantees the correct ranking by visiting in expectation less than a number of nodes proportional to the square root of the size of the input graph – even if only the top ranked nodes are examined and if their PageRank scores are well separated. Experimental results show that, in a large crawl of the `.it` web graph, the estimated cost of local ranking is substantially smaller than the graph size (but still very high), while in a smaller crawl of the LiveJournal friendship graph the estimated cost is an important fraction of the size of the graph and even higher than in the `.it` web graph.

This introductory section motivates our work and provides related work (Subsection 1.1), and then outlines the structure of the paper (Subsection 1.2).

### 1.1 Motivation and Related Work

This paper ideally builds a bridge between the research on the ranking induced by PageRank and that on the local computation of PageRank.

The first research stream arises from the consideration that PageRank [10], originally proposed as a means to infer the importance of web pages from the topological structure of the web graph, is nowadays a reference algorithm in computer science, especially for the information retrieval, knowledge management and data mining communities. Paradoxically, while its utility as a web search algorithm is decreasing in favour of new techniques such as clickthrough-based measures, its importance is still growing, due to a large number of applications in diverse fields: for instance, in web crawling [13], web spam detection [19], social network mining [20], ranking in databases [17], structural re-ranking [23], opinion mining [16], word sense disambiguation [29], credit and reputation systems [22], gene ranking [27] and bibliometrics [32]; indeed, PageRank has been rated among the top 10 algorithms in data mining [31]. This large and growing number of applications suggests to study PageRank in the abstract, considering it as a graph algorithm which accepts a graph in

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

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

Copyright 2011 ACM 978-1-4503-0717-8/11/10 ...\$10.00.

input and provides a score to each node of the graph as output. While in the original web application these scores were combined with traditional text-based information retrieval scores, in most of the other applications they are directly used to *rank* the nodes of the input graph. For instance, in *web crawling*, PageRank is used to decide ‘in what order a crawler should visit the URLs it has seen, in order to obtain more “important” pages first’ [13]. In *bioinformatics*, modified versions of PageRank are used to rank genes in order of their “importance”, offering ‘an improved ranking of genes compared to pure expression change rankings’ [27]. This naturally moved research towards the ranking induced by PageRank, investigating its stability under graph perturbations [8, 25], the convergence of its iterative computation [28, 30], and its dependence on damping factor variations [26, 9].

On the other hand, a vast body of research has been carried out on the local computation of PageRank scores. This research is motivated by the fact that the input graph is often accessible only for a high cost – or one may simply have access only to the local structure of the graph, without any knowledge of the full topology. The first case happens, for instance, when the input graph is too large to fit completely in one specific level of the *memory hierarchy* and must be stored in larger (and slower) levels, where (random) accesses have a significant cost [21]. The second case happens, for instance, when a *web user* can access the web graph only by querying a “link server”, e.g. using the ‘link:’ option provided by some search engines, or when a *social network user* has limited access to the friendship graph due to privacy settings (see [18]). In all these cases it would be useful to compute PageRank locally, i.e. exploiting only the structure of a small subgraph around the target nodes – indeed, this is the approach of [1]. To this end, [12] provides heuristics to locally approximate PageRank scores, [2] gives a well-founded local algorithm to approximate the contributions of the nodes of a graph to the PageRank score of a target node, while [3] proves the infeasibility of locally approximating the PageRank scores in the worst case, and provides lower bounds for deterministic and randomized local score approximation algorithms.

Neither theoretical nor experimental work has ever been carried out on the local computation of PageRank-induced ranking – indeed, the problem itself has never been formally defined, even if previous work [12, 3] has repeatedly suggested the need for this kind of study.

## 1.2 Outline of the Paper

This work formally defines the problem of locally computing PageRank-induced rankings and provides the first theoretical and experimental results on this subject. The paper is organized as follows.

Section 2 gives a short overview of the PageRank algorithm with an emphasis on the mathematical formulation which allows a local computation of the output scores.

Section 3 gives an informal introduction to the problem in hand, and proves that the reference algorithm used to locally approximate PageRank scores can not guarantee a correct ranking when visiting less than a number of nodes linear in the size of the input graph, or may even never stabilize.

Section 4 formally defines the problem of local ranking, and Section 5 provides the main theoretical results of the paper, showing that the results of Section 3 extend to *every* algorithm which attempts a local computation of PageRank-

induced rankings. In short, it proves that the local computation of PageRank-induced ranking is infeasible in the worst case both for deterministic and for randomized algorithms – even if only the top ranked nodes are examined and if their PageRank scores are well separated.

Section 6 describes two experiments suggesting that the cost of local ranking in real graphs is actually high and strongly dependent on the input graph.

Section 7 discusses our results and their consequences, and states a few open problems before concluding with the bibliography.

## 2. PAGERANK

Let  $G = (V, A)$  be an  $n$ -node graph with no dangling nodes (i.e. nodes with no outgoing arcs); if dangling nodes originally exist, the graph is preprocessed by adding to each of them outgoing arcs towards each node of the graph. An arc from  $u$  to  $v$  is interpreted as a conferral of importance from  $u$  to  $v$ . The entries of the transition probability matrix  $\mathbf{M} = [m_{i,j}]$  of  $G$  are

$$m_{i,j} = \begin{cases} \frac{1}{\text{outdegree}(i)} & \text{if an arc exists from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Let the damping factor  $\alpha$  be a real number in  $[0, 1)$  and let

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

where  $\mathbf{U}$  is a matrix whose entries equal  $1/n$ ;  $\mathbf{T}$  can be seen as the transition probability matrix of a Markov chain. The PageRank row vector  $\mathbf{P} = [P(v_1), P(v_2), \dots, P(v_n)]$  is defined as the limit probability vector of this Markov chain, and can be obtained either as the unique probability vector which is the solution of the linear system  $\mathbf{P} = \mathbf{P}\mathbf{T}$  or as the limit of the iteration  $\mathbf{P}^{(n)} = \mathbf{P}^{(n-1)}\mathbf{T}$ , where the starting vector  $\mathbf{P}^{(0)}$  can be any probability vector. When considering a local computation of PageRank, however, none of these roads can be followed, since the entire graph structure is not available; in this case, straightforward algebraic manipulations are exploited which take to the formula:

$$\mathbf{P} = \frac{1 - \alpha}{n} \mathbf{1}(\mathbf{I} - \alpha\mathbf{M})^{-1} = \frac{1 - \alpha}{n} \mathbf{1} \sum_{\tau=0}^{+\infty} (\alpha\mathbf{M})^\tau$$

where  $\mathbf{1} = [1, 1, \dots, 1]$ . This leads to the following formula to compute the PageRank score of a node  $v$ :

$$P(v) = \frac{1 - \alpha}{n} \sum_{\tau=0}^{+\infty} \alpha^\tau \sum_{z \in G} \text{inf}_\tau(z, v) \quad (1)$$

where the influence  $\text{inf}_\tau(z, v)$  of  $z$  on  $v$  gives the probability of ending in  $v$  by following a chain of  $\tau$  arcs from  $z$ . Therefore, the score of node  $v$  can be seen as the weighted sum of the influences of its ancestors at each layer  $\tau$ , or as the sum of their contributions.

## 3. LIMITS AND PITFALLS OF THE BRUTE FORCE ALGORITHM

This section investigates the brute force algorithm, one of the simplest methods to locally approximate the PageRank score of a single node, in light of the local ranking problem. Although unlikely to be among the best local score approximation algorithms to lever on in the local ranking problem (see Section 6), the brute force is still a reference

algorithm for at least two reasons. First, it is the basis of virtually any known local *score* approximation algorithm (such as those proposed in [3] and [12]) which in turn are natural bases for local *ranking* algorithms. Second, it naturally derives from the expression of PageRank given by Equation (1) that turns out to be very useful in the analysis of those algorithms. Thus it is important to understand why the brute force may fail to be the basis for a good local ranking algorithm – one that, ideally, should incur a limited cost and return a correct result. Indeed, as we shall prove after a short review, a naive brute force-based ranking algorithm may incur extremely high costs to guarantee a correct result (Theorem 1) or oscillate indefinitely between two “opposite” results (Theorem 2).

Given an  $n$ -node graph  $G$  and a target node  $v \in G$ , the brute force algorithm performs a breadth-first visit of the ancestors of  $v$ , querying at iteration  $\tau$  all the ancestors of layer  $\tau$  and cumulating their influence on  $v$  weighted by  $\frac{1-\alpha}{n}\alpha^\tau$ . After iteration  $\ell$ , the resulting *brute force score* at layer  $\ell$  is

$$P^{(\ell)}(v) = \frac{1-\alpha}{n} \sum_{\tau=0}^{\ell} \alpha^\tau \sum_{z \in G} \text{inf}_\tau(z, v) \quad (2)$$

If graph  $G$  has no dangling nodes,  $P^{(\ell)}(v)$  converges from below to the PageRank score of  $v$  as given by Equation (1). A natural way to compute the relative ranking of two target nodes  $u$  and  $v$  is then to compare their brute force scores  $P^{(\ell)}(u)$  and  $P^{(\ell)}(v)$  once  $\ell$  has reached a sufficiently large threshold  $\ell_0$ . Unfortunately, such a threshold is unknown a priori, and this may lead to a premature halt and an incorrect ranking. Indeed we prove that, for any given  $\ell_0$ , there exist graphs where the “decisive” layer is at depth  $\ell_0 + 1$  while the ranking induced by the output of the brute force algorithm at *every* iteration up to  $\ell_0$  is the complete reversal of the correct ranking. Before stating the result formally in the next theorem, we provide a lemma that will greatly simplify our proofs.

**LEMMA 1.** *Let  $G$  be a graph,  $v \in G$  a dangling node, and build  $G'$  adding a self-loop  $(v, v)$  to  $G$ . Let  $B(v)$  and  $B'(v)$  be the brute force scores of  $v$  computed respectively on  $G$  and  $G'$ . Then  $B'(v) = \frac{B(v)}{1-\alpha}$ .*

**PROOF.** Every path from a generic node  $z$  to  $v$  in  $G'$  is the unique concatenation of a path from  $z$  to  $v$  in  $G$  and  $t \geq 0$  self-loops  $(v, v)$ . Conversely, every path  $p : z \rightarrow v$  in  $G$  generates, for  $t = 0, \dots, \infty$ , the paths from  $z$  to  $v$  in  $G'$  that are the concatenation of  $p : z \rightarrow v$  and  $t$  self-loops  $(v, v)$  that damp the contribution of  $z$  by a factor  $\alpha^t$ . Thus the expression of  $B'(v)$  given by Equation (2) can be rewritten as:

$$\begin{aligned} B'(v) &= \frac{1-\alpha}{n} \sum_{\tau=0}^{\infty} \alpha^\tau \sum_{z \in G} \text{inf}_\tau(z, v) \sum_{t=0}^{\infty} \alpha^t \\ &= \frac{1-\alpha}{n} \sum_{\tau=0}^{\infty} \alpha^\tau \sum_{z \in G} \text{inf}_\tau(z, v) \frac{1}{1-\alpha} \\ &= B(v) \frac{1}{1-\alpha} \end{aligned}$$

□

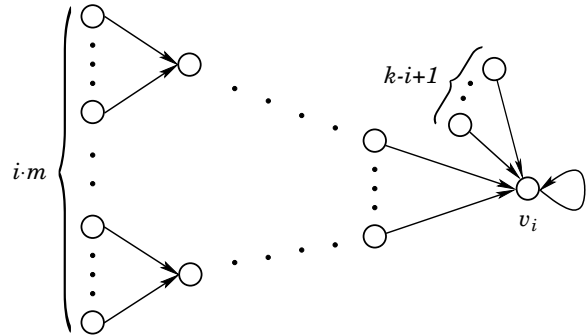
Lemma 1 proves that one can compute the brute force score of a node whose only outgoing arc forms a self-loop by

disregarding it at each iteration and multiplying the result by a constant factor. We can now state Theorem 1.

**THEOREM 1.** *For any  $k \geq 2$ , any  $\alpha \in (0, 1)$  and any  $\ell_0 > 0$  there exists a graph where the ranking induced by  $P^{(\ell)}$  on the top  $k$  nodes  $v_1, \dots, v_k$  is:*

$$\begin{aligned} v_1 &< \dots < v_k && \text{for } \ell \leq \ell_0 \\ v_k &< \dots < v_1 && \text{for } \ell > \ell_0 \end{aligned}$$

**PROOF.** Due to space limitations, we give a sketch of the proof exhibiting a graph  $G$  that satisfies the statement.  $G$  consists of  $k$  subgraphs  $G_1, \dots, G_k$ . Subgraph  $G_i$  (Figure 1) contains node  $v_i$  and its self-loop (making the graph free of dangling nodes),  $k - i + 1$  orphan nodes that have  $v_i$  as their sole child, and  $\ell_0 + 1$  layers of ancestors structured as a tree of depth  $\ell_0$  with root node  $v_i$  and indegree  $d$  ( $d$  will be computed below) pointed by an additional  $\ell_0 + 1$ -th layer of  $im$  orphans ( $m$  will be computed below).



**Figure 1: Subgraph  $G_i$  (Theorem 1) containing node  $v_i$  and all its ancestors up to layer  $\ell_0 + 1$ .**

Consider  $v_i$  and  $v_j$  for  $1 \leq i < j \leq k$ . For  $\ell \leq \ell_0$ , node  $v_i$  receives the contribution of  $j - i$  parents more than  $v_j$ . But for  $\ell \geq \ell_0 + 1$ , node  $v_j$  receives from layer  $\ell_0 + 1$  the contribution of  $(j - i)m$  ancestors more than  $v_i$ . Thus, intuitively,  $v_j$  receives a lower contribution from the first  $\ell_0$  layers but a higher contribution from layer  $\ell_0 + 1$ . Indeed, choosing  $m > \frac{1-\alpha^{\ell_0+1}}{\alpha^{\ell_0}(1-\alpha)}$ , we obtain  $P^\ell(v_i) > P^\ell(v_j)$  for  $\ell \leq \ell_0$  and  $P^\ell(v_i) < P^\ell(v_j)$  for  $\ell \geq \ell_0 + 1$ .

To make  $v_1, \dots, v_k$  the top  $k$  nodes in the graph, balance the  $im$  arcs from layer  $\ell_0 + 1$  so that the number of parents of any ancestor at layer  $\ell_0$  is at most  $\lceil \frac{im}{d^{\ell_0}} \rceil$  which, for a sufficiently large  $d$ , is no more than  $d$ . Note that  $v_1, \dots, v_k$  have more than  $d$  parents and, in general, at any layer they have more ancestors (with outdegree 1, which translates to a higher contribution) than any other node in the graph – thus, they are the top  $k$ . □

By Theorem 1, for any  $\ell_0 > 0$  there exist graphs where, to compute the correct relative ranking of the top  $k$  nodes, the brute force algorithm not only must take into account at least the first  $\ell_0 + 1$  layers, but halting at *any* layer below  $\ell_0 + 1$  yields the *same* complete reversal of the correct ranking – a misleading “stability” condition.

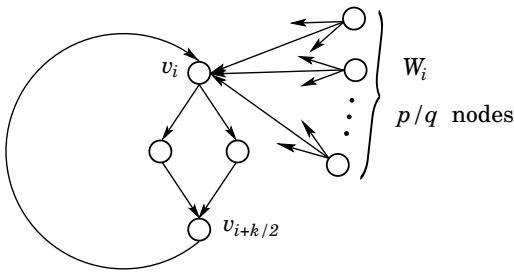
Although to correctly rank the top  $k$  nodes of the graph of Theorem 1 the brute force algorithm incurs a very high cost (exponential in  $\ell$  and linear in  $n$ ), it still converges in a finite number of iterations. Unfortunately, the situation can worsen in presence of cycles that produce periodical *oscillations* in the relative ranking of nodes. The following theorem

proves that in this case the number of iterations required to converge to a stable ranking may be unbounded:

**THEOREM 2.** *For any even integer  $k > 1$  and any rational  $\alpha \in (0, 1)$  there exists a graph where the ranking induced by  $P^{(\ell)}$  on the top  $k$  nodes  $v_1, \dots, v_k$  satisfies, for any  $i \in [1, \frac{k}{2}]$ , any  $j \in [\frac{k}{2} + 1, k]$ , and any  $\ell \geq 0$ :*

$$\begin{aligned} v_i &= v_j & \text{for } \ell \equiv 0 \pmod{3} \\ v_i &< v_j & \text{for } \ell \equiv 1 \pmod{3} \\ v_i &> v_j & \text{for } \ell \equiv 2 \pmod{3} \end{aligned}$$

**PROOF.** Due to space limitations, we give a sketch of the proof exhibiting a graph  $G$  that satisfies the statement.  $G$  consists of  $k/2$  identical subgraphs  $G_1, \dots, G_{k/2}$ . Subgraph  $G_i$  (Figure 2) contains target nodes  $v_i$  and  $v_{i+k/2}$ , two nodes that are the sole children of  $v_i$  and the sole parents of  $v_{i+k/2}$ , and a set  $W_i$  of  $p$  nodes with  $q$  children each ( $v_i$  being one of them).



**Figure 2: Subgraph  $G_i$  (Theorem 2) has a cycle that causes a perpetual oscillation in the relative ranking given by the brute force algorithm.**

Intuitively, both target nodes receive the contribution of nodes in  $W_i$ , but  $v_i$  only at layers  $\ell \equiv 1 \pmod{3}$  and  $v_{i+k/2}$  only at layers  $\ell \equiv 0 \pmod{3}$  for  $\ell > 0$ ; this makes the relative ranking induced by their brute force scores at layer  $\ell$  oscillate with  $\ell$ . More formally, choose  $p$  and  $q$  such that the overall contribution of nodes in  $W_i$  is  $p/q = 1/(1 + \alpha)$ ; this is possible since  $\alpha$  is rational by hypothesis. Plugging this value into the expressions of the brute force scores of  $v_i$  and  $v_{i+k/2}$  at layer  $\ell$  given by Equation (2), it is easy to see that  $P^{(\ell)}(v_i) = P^{(\ell)}(v_j)$  for  $\ell \equiv 0 \pmod{3}$ ,  $P^{(\ell)}(v_i) < P^{(\ell)}(v_j)$  for  $\ell \equiv 1 \pmod{3}$ , and  $P^{(\ell)}(v_i) > P^{(\ell)}(v_j)$  for  $\ell \equiv 2 \pmod{3}$ . Furthermore, the target nodes are the top  $k$  ranked in the graph, since each child of  $v_i$  has a brute force score smaller than that of both  $v_i$  and  $v_{i+k/2}$  at any layer  $\ell > 1$ , while nodes of set  $W_i$  are orphans and thus always have the lowest brute force scores in the graph.  $\square$

By Theorem 2, there exist graphs where the relative ranking given by the brute force algorithm on the top  $k$  nodes never converges nor shows stability. Note that, depending on  $\alpha$ ,  $p$  and  $q$ , the size of the graph in Figure 2 can become arbitrarily large but also range in the tens – a situation where the brute force algorithm presents an excellent cost performance, yet its output oscillates indefinitely.

Although by Theorems 1 and 2 the brute force algorithm may in theory become impractical, other algorithms could perform better for at least three reasons. First, they could visit the ancestors of the target nodes using a strategy that includes highly-contributing ancestors earlier than the simple breadth-first does. Second, they could compute the approximate PageRank scores more accurately, for example

taking into account the cycles in the graph to eliminate oscillations and include “in one shot” the total contribution of an ancestor appearing in an infinite number of layers. Third, as the number of visited nodes increases, the approximation error on the PageRank scores drops (eventually going beyond the resolution of a physical machine) and the correct ranking of the target nodes may become apparent after just a few steps, especially if their scores are not too close.

These observations raise the necessity to discuss and formally define the problem of local ranking. The next section is fully devoted to this task.

## 4. LOCAL RANKING OF NODES

This section formally restates the local ranking problem informally introduced in Section 1, taking into account the observations raised at the end of Section 3.

Formally, a *local algorithm* is an algorithm that has no direct access to (the arcs of) a graph  $G$ , but must instead query a *link server* for  $G$  that accepts (the ID of) a node  $v$  in input and returns (the IDs of) the parents and children of  $v$  in output. The *local ranking problem* consists in ranking a subset of nodes of a graph using only local algorithms. Since the major bottleneck of local algorithms is the communication with the link server, their cost can be defined as the number of queries performed. The cost of locally ranking two target nodes  $u, v \in G$  is then the minimum cost incurred by any correct algorithm  $A$  to rank  $u$  and  $v$ .

In this paper, the local ranking problem is considered in a PageRank context. Formally, given a graph  $G$ , a damping factor  $\alpha$ , and  $k$  target nodes  $v_1, \dots, v_k \in G$ , the local ranking problem requires to rank the target nodes in nonincreasing order of their PageRank scores (with ties broken arbitrarily) using only local algorithms. In many applications, however, only the top PageRank scores really matter; furthermore, often only the ranking induced by well-separated scores matters – if two scores are very close to each other, they could be considered equivalent (after all, PageRank itself gives an approximate model of the reality), and they are practically indistinguishable if their difference is smaller than the resolution of a modern machine. To deal with this last issue, we modify the problem and require to rank all and only the target nodes whose relative PageRank score difference is not less than a given  $\epsilon$ . Formally, given a graph  $G$ , a damping factor  $\alpha$ , and  $k$  target nodes  $v_1, \dots, v_k \in G$ , the *local  $\epsilon$ -ranking problem* requires to rank each pair  $u, v$  of target nodes with  $P(u)/P(v) \geq 1 + \epsilon$  in decreasing order of their PageRank scores using only local algorithms. The definitions of cost of an algorithm  $A$  and cost of local ranking given above can be ported in this context with obvious modifications.

It is thus natural to ask how high is the cost of local  $\epsilon$ -ranking – even considering only the top nodes of a graph. In the next section we prove theoretical lower bounds on this cost, for both randomized and deterministic algorithms.

## 5. THE COST OF LOCAL RANKING

In general, locally computing the exact PageRank score of a single node may require a number of queries proportional to the size  $n$  of the whole graph. More surprisingly, a very high number of queries may be required even to compute an  $\epsilon$ -approximation of the PageRank score of a single node, i.e. a value between  $1 - \epsilon$  and  $1 + \epsilon$  times the score itself. In

particular (see [3]), to  $\epsilon$ -approximate the score of a node for a reasonable constant  $\epsilon$ , any deterministic local algorithm incurs a cost of  $\Omega(n)$  queries in the worst case, while any randomized (with deterministic cost) Monte Carlo local algorithm with constant confidence incurs an expected cost of  $\Omega(\sqrt{n})$  queries in the worst case.

It is not known, however, if the same bounds apply to the problem of locally computing the relative  $\epsilon$ -ranking of two or more target nodes, as described in Section 4. We prove that this is indeed the case: the following Theorem 3 shows that, in the worst case, locally ranking the top  $k$  nodes of a graph requires  $\Omega(\alpha\sqrt{n/\epsilon})$  queries for both Las Vegas algorithms and Monte Carlo algorithms with constant confidence, while Theorem 4 strengthens the bound to  $\Omega(n)$  for deterministic algorithms.

**THEOREM 3.** *Choose integers  $k > 1$  and  $n_0 \geq 6k^3$ , a damping factor  $\alpha \in (0, 1)$ , and an  $\epsilon \in \left[\frac{\alpha^2 k^2}{4n_0}, \frac{\alpha^2}{24k}\right]$ . Then*

- for any Las Vegas local algorithm  $A$  there exists a graph of size  $n \in \Theta(n_0)$  where the top  $k$  nodes  $v_0, \dots, v_{k-1}$  are  $\epsilon$ -separated and, to compute their relative ranking,  $A$  performs an expected  $\Omega(\alpha\sqrt{\frac{n}{\epsilon}})$  queries.
- for any Monte Carlo local algorithm  $A$  with constant confidence there exists a graph of size  $n \in \Theta(n_0)$  where the top  $k$  nodes  $v_0, \dots, v_{k-1}$  are  $\epsilon$ -separated and, to compute their relative ranking,  $A$  performs  $\Omega(\alpha\sqrt{\frac{n}{\epsilon}})$  queries.

The proof is based on a reduction from the 1-OR problem. In this problem, the instance is a binary string  $\mathbf{x}$  of length  $m \geq 1$  such that either all its bits are 0, in which case the solution is 0, or exactly one is 1, in which case the solution is 1 – i.e., the solution is  $\|\mathbf{x}\|$ . Local algorithms for 1-OR can retrieve the value of a bit only via a *bit server*. We use Yao’s principle [33] to show that the expected cost of any Las Vegas randomized local algorithm for 1-OR is at least  $\frac{m}{2}$  in the worst case:

**LEMMA 2.** *The expected worst-case cost of any Las Vegas randomized local algorithm for 1-OR is at least  $\frac{m}{2}$ .*

**PROOF OF LEMMA 2.** Any (Las Vegas) algorithm issues a sequence of queries and stops either when it finds a 1 or every bit has been queried (every algorithm behaving differently would produce an incorrect result and/or incur unnecessary costs). Consider an input probability distribution where each of the  $m + 1$  possible  $m$ -bit strings ( $m$  of them contain exactly a 1, and one contains only zeros) has probability  $\frac{1}{m+1}$ . Whatever the sequence of queries, the probability that the  $j$ -th query returns 1 (and thus that the cost is  $j$ ) is  $\frac{1}{m+1}$ . Therefore the expected cost is at least

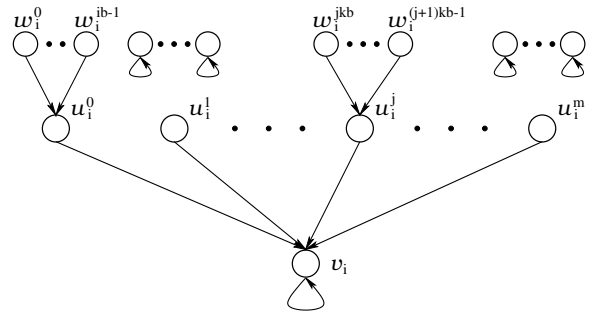
$$\sum_{j=1}^m j \frac{1}{m+1} = \frac{m}{2}$$

By Yao’s principle, this is a bound on the expected number of queries performed by any randomized Las Vegas algorithm on its worst-case input.  $\square$

**PROOF OF THEOREM 3.** Let  $A$  be a randomized Monte Carlo (Las Vegas) local algorithm that ranks the top  $k$  nodes of a graph performing (in expectation)  $S_A$  queries to the link server; we use  $A$  to build a randomized Monte Carlo (Las Vegas) local algorithm  $B$  that solves  $k - 1$  independent

instances of 1-OR performing (in expectation)  $S_B \leq S_A$  queries to the bit server. Lemma 2 gives a lower bound on (the expected value of)  $S_B$ , and thus on (the expected value of)  $S_A$ .

Let  $\mathbf{x}_1, \dots, \mathbf{x}_{k-1}$  be  $k - 1$   $m$ -bit instances of 1-OR and  $b$  a positive integer ( $m$  and  $b$  will be computed below). We build a link server that lets  $A$  run on a graph  $G$  consisting of  $k$  disjoint subgraphs  $G_0, G_1, \dots, G_{k-1}$ . For  $i = 1, \dots, k - 1$ , subgraph  $G_i$  (Figure 3) contains the target node  $v_i$  and its self-loop,  $m + 1$  nodes  $u_i^0, \dots, u_i^m$  that have  $v_i$  as their sole child,  $ib$  nodes  $w_i^0, \dots, w_i^{ib-1}$  that have  $u_i^0$  as their sole child and, for  $j = 1, \dots, m$ ,  $kb$  nodes  $w_i^{jkb}, \dots, w_i^{(j+1)kb-1}$  that have  $u_i^j$  as their sole child if  $\mathbf{x}_i(j) = 1$  or a self-loop if  $\mathbf{x}_i(j) = 0$ . Subgraph  $G_0$  contains the reference target node  $v_0$  and its self-loop,  $m + 1$  nodes  $u_0^0, \dots, u_0^m$  that have  $v_0$  as their sole child and  $kb$  nodes  $w_0^0, \dots, w_0^{kb-1}$  whose sole child is  $u_0^0$ .



**Figure 3: Subgraph  $G_i$  (Theorem 3).** Node  $u_i^0$  always has exactly  $ib$  parents; node  $u_i^j$  has  $kb$  parents if  $\mathbf{x}_i(j) = 1$ , else it has an associated group of  $kb$  “disconnected parents”.

The link server for  $G$  is described by the following rules:

1. when queried for  $u_i^j$  with  $i, j \geq 1$ , it queries the bit server for  $\mathbf{x}_i(j)$ ; if  $\mathbf{x}_i(j) = 1$  then it returns  $v_i$  as the sole child and  $w_i^{jkb}, \dots, w_i^{(j+1)kb-1}$  as the parents, else it returns  $v_i$  as the sole child and no parents.
2. when queried for  $w_i^j$  with  $i \geq 1$  and  $j \geq kb$ , it computes  $j' = \lfloor \frac{j}{kb} \rfloor$  and queries the bit server for  $\mathbf{x}_i(j')$ ; if  $\mathbf{x}_i(j') = 1$  then it returns  $u_i^{j'}$  as the sole child and no parents, else it returns  $w_i^{j'kb}$  as the sole child and no parents.
3. otherwise, it answers without querying the bit server.

It is easy to see that this is a link server for the graph  $G$ . Given the output of algorithm  $A$ , algorithm  $B$  computes the solution of  $\mathbf{x}_i$  as follows. If  $A$  ranks  $v_i$  lower than  $v_0$ , then  $B$  returns 0; if  $A$  ranks  $v_i$  higher than  $v_0$ , then it returns 1. We prove that if  $A$  ranks correctly  $v_0, \dots, v_{k-1}$  then  $B$  solves correctly the  $k - 1$  instances of 1-OR.

Since  $G$  has no dangling nodes, by Lemma 1 the scores of the target nodes are  $\frac{1}{1-\alpha}$  times the brute force score computed on the same nodes ignoring their self-loops. Thus, according to Equation (2) these scores become:

$$P(v_0) = \frac{1}{n} (1 + \alpha(m+1) + \alpha^2 kb)$$

$$P(v_i) = \frac{1}{n} (1 + \alpha(m+1) + \alpha^2 (i+k\|\mathbf{x}_i\|)b) \quad i \geq 1$$

By hypothesis, algorithm  $A$  ranks  $v_0, v_1, \dots, v_{k-1}$ ; in particular, it computes the relative ranking of  $v_0$  and  $v_i$  for  $i = 1, \dots, k-1$ . But  $v_0$  is ranked lower than  $v_i$  if and only if  $k < i + k\|\mathbf{x}_i\|$ , which is true if and only if  $\|\mathbf{x}_i\| = 1$ . Conversely,  $v_i$  is ranked lower than  $v_0$  if and only if  $i + k\|\mathbf{x}_i\| < k$ , which is true if and only if  $\|\mathbf{x}_i\| = 0$ . Thus, if  $A$  ranks correctly  $v_0, \dots, v_{k-1}$ , then  $B$  solves correctly the  $k-1$  instances of 1-OR.

We now exhibit values of  $m$  and  $b$  such that  $G$  and the target nodes satisfy the thesis. In particular, choose

$$m = \left\lceil \frac{\alpha\sqrt{n_0/\epsilon}}{12k} \right\rceil \quad \text{and} \quad b = \left\lfloor \frac{2\sqrt{\epsilon n_0}}{\alpha k} \right\rfloor$$

It is immediate to verify that the size  $n$  of the graph is in  $\Theta(mbk^2)$  which, substituting  $m$  and  $b$ , is in  $\Theta(n_0)$ . The hypothesis  $n_0 \geq 6k^3$  guarantees also that  $b \geq 1$  and  $m \geq kb \geq 2$ ; therefore  $v_0, v_1, \dots, v_{k-1}$  are the top  $k$  nodes in the graph, since each of them has at least  $m+1 > kb$  parents and at least one grandparent – more than any other node in the graph. Furthermore, any two target nodes are  $\epsilon$ -separated since one of them has the same number of parents but at least  $b$  grandparents more than the other, and thus their scores differ by  $\Delta P \geq \frac{\alpha^2 b}{n}$ . This value, divided by the maximum possible score of a target node (obtained for  $i = k-1$  and  $\|\mathbf{x}_i\| = 1$ ), gives a lower bound on the score separation of any two target nodes:

$$\frac{\Delta P}{P} \geq \frac{\alpha^2 b}{n} / \frac{1}{n} (1 + \alpha(m+1) + \alpha^2(k-1+k)b) \geq \frac{\alpha^2 b}{5m}$$

where the last inequality follows from  $kb \leq m$  and  $1 \leq m/2$ . Plugging in the above values for  $b$  and  $m$  we obtain:

$$\frac{\Delta P}{P} \geq \frac{\alpha^2 b}{5m} = \frac{\alpha^2 \left\lfloor \frac{2\sqrt{\epsilon n_0}}{\alpha k} \right\rfloor}{5 \left\lceil \frac{\alpha\sqrt{n_0/\epsilon}}{12k} \right\rceil} \geq \frac{\alpha^2 \frac{\sqrt{\epsilon n_0}}{\alpha k}}{5 \frac{\alpha\sqrt{n_0/\epsilon}}{6k}} = \frac{6}{5}\epsilon > \epsilon$$

which proves that the target nodes are  $\epsilon$ -separated.

We now compute the cost of locally ranking the target nodes  $v_1, \dots, v_k$ . The link server performs at most one query to the bit server for each query performed by  $A$ , therefore  $S_B \leq S_A$ . If  $A$  (and thus  $B$ ) is a Las Vegas algorithm, Lemma 2 gives a worst-case expected cost of at least  $m/2$  bit queries for each of the  $k-1$  instances solved by  $B$ , and thus  $E[S_A] \geq E[S_B] \geq \frac{m}{2}(k-1) \in \Omega(mk)$ . If  $A$  (and thus  $B$ ) is a Monte Carlo algorithm with constant confidence (note that  $B$  is correct if  $A$  is, thus the confidence of  $B$  is equal to at least the confidence of  $A$ ), a sensitivity argument for randomized algorithms with bounded error [11] gives a worst-case cost of  $\Omega(m)$  bit queries for each of the  $k-1$  instances, and thus  $S_A \geq S_B \in \Omega(mk)$ . In every case  $S_A \in \Omega(mk)$  which, plugging in the above value for  $m$  and recalling that  $n_0 \in \Theta(n)$ , becomes:

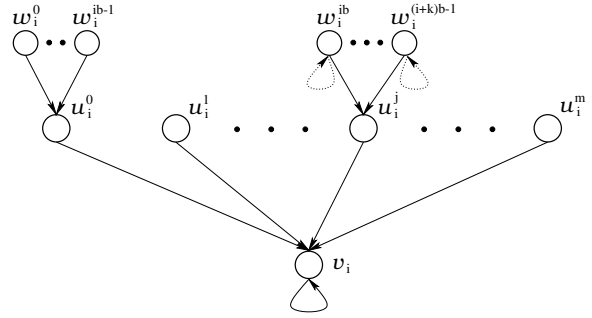
$$S_A \in \Omega\left(\left\lceil \frac{\alpha\sqrt{n_0/\epsilon}}{12k} \right\rceil k\right) = \Omega\left(\alpha\sqrt{\frac{n}{\epsilon}}\right)$$

which concludes the proof.  $\square$

The bound for deterministic algorithms is even stronger. Intuitively, an adversarial link server can adaptively build a worst-case graph by refusing to give enough information until  $\Omega(n)$  queries have been performed. More formally:

**THEOREM 4.** Choose integers  $k > 1$  and  $n_0 \geq k^2$ , a damping factor  $\alpha \in (0, 1)$ , and an  $\epsilon \leq \frac{\alpha^2}{20k}$ . For any deterministic local algorithm  $A$  there exists a graph of size  $n \in \Omega(n_0)$  where the top  $k$  nodes  $v_0, \dots, v_{k-1}$  are  $\epsilon$ -separated and, to compute their relative ranking,  $A$  performs  $\Theta(n)$  queries.

**PROOF.** Let  $A$  be a deterministic local algorithm that ranks the top  $k$  nodes of a graph performing  $S_A$  queries to the link server; we exhibit an adversarial link server that lets  $A$  run on a worst-case graph  $G$  similar to that of Theorem 3 and consisting of  $k$  disjoint subgraphs  $G_0, G_1, \dots, G_{k-1}$ . Let  $m$  and  $b$  be two positive integers (we compute them below). For  $i = 1, \dots, k-1$ , subgraph  $G_i$  (Figure 4) contains target node  $v_i$  and its self-loop,  $m+1$  nodes  $u_i^0, \dots, u_i^m$  that have  $v_i$  as their sole child,  $ib$  nodes  $w_i^0, \dots, w_i^{ib-1}$  that have  $u_i^0$  as their sole child and  $kb$  nodes  $w_i^i, \dots, w_i^{(i+k)b-1}$  that either have self-loops or, for some  $1 \leq j \leq m$ , have  $u_i^j$  as their sole child. Subgraph  $G_0$  contains target node  $v_0$  and its self-loop,  $m+1$  nodes  $u_0^0, \dots, u_0^m$  that have  $v_0$  as their sole child, and  $kb$  nodes  $w_0^0, \dots, w_0^{kb-1}$  that have  $u_0^0$  as their sole child.



**Figure 4:** Subgraph  $G_i$  (Theorem 4). Node  $u_i^0$  always has  $ib$  parents, while nodes  $w_i^{ib}, \dots, w_i^{(i+k)b-1}$  may or may not have  $u_i^j$  as their sole child.

The link server exploits the fact that the mapping of node IDs (i.e. the content of queries) to nodes in the graph is arbitrary, and forces algorithm  $A$  to query almost all the nodes before giving enough information to compute a correct ranking. Indeed, the link server follows this simple rule:

- on the first  $mk$  queries, it never maps an ID to any of the nodes  $w_i^{ib}, \dots, w_i^{(i+k)b-1}$  for  $i = 1, \dots, k-1$  or any of their children.

Note that this is a legitimate behaviour since each of the  $k$  subgraphs  $G_0, \dots, G_{k-1}$  has at least  $m$  nodes that are neither  $w_i^{ib}, \dots, w_i^{(i+k)b-1}$  nor their children.

We now exhibit values of  $m$  and  $b$  such that  $G$  and the target nodes satisfy the theorem. In particular, choose

$$m = \left\lceil \frac{n_0}{k} \right\rceil \quad \text{and} \quad b = \left\lfloor \frac{n_0}{k^2} \right\rfloor$$

The size of the graph,  $n \in \Theta(mk + k^2b)$ , is thus in  $\Theta(n_0)$ . The hypothesis  $n_0 \geq k^2$  guarantees that  $m \geq kb$  and  $b \geq 1$ ; thus  $v_0, \dots, v_{k-1}$  are the top  $k$  nodes in the graph, since at any layer they have more ancestors than any other node.

It remains to prove that the target nodes are  $\epsilon$ -separated. Since the number of parents and grandparents of  $v_i$  is the same as in the graph of Theorem 3, and since  $kb \leq m$ ,

the difference between scores satisfies the same lower bound  $\frac{\Delta P}{P} \geq \frac{\alpha^2 b}{5m}$ . The substitution of  $b$  and  $m$  gives

$$\frac{\Delta P}{P} \geq \frac{\alpha^2 b}{5m} \geq \frac{\alpha^2 \lfloor \frac{n_0}{k^2} \rfloor}{5 \lfloor \frac{n_0}{k} \rfloor} \geq \frac{\alpha^2 \frac{n_0}{2k^2}}{5 \frac{2n_0}{k}} \geq \frac{\alpha^2}{20k} \geq \epsilon$$

which proves that the target nodes are  $\epsilon$ -separated.

Note that the link server can invalidate *any* ranking of the target nodes that  $A$  should output *before* querying at least  $mk$  nodes. Indeed, suppose that  $A$  has queried less than  $mk$  nodes and consider a target node  $v_i$  such that one of its parents,  $u_i^j$ , has not been queried (one such node always exist). If  $A$  ranks  $v_i$  higher than  $v_0$ , the link server disconnects  $w_i^b, \dots, w_i^{(i+k)b-1}$  from  $u_i^j$ , otherwise it connects them to  $u_i^j$ . The expression of  $P(v_i)$  (see proof of Theorem 3) shows that in either case  $A$  gives an incorrect ranking. Therefore  $A$  must incur a cost of at least  $mk$ ; substituting  $m = \lfloor \frac{n_0}{k} \rfloor$  and  $n_0 \in \Theta(n)$ , we conclude the proof:

$$S_A \geq mk = \left\lfloor \frac{n_0}{k} \right\rfloor k \geq n_0 \in \Omega(n)$$

□

Note that, as the sizes of the graphs of Theorems 3 and 4 increase, the upper bound on the choice of  $\epsilon$  falls within a small constant factor of  $\frac{\alpha^2}{k}$ , taking the lower bounds on the incurred costs within a small constant factor of, respectively,  $\sqrt{kn}$  (increasing to  $n/k$  for  $\epsilon \approx \frac{\alpha^2 k^2}{n}$ ) and  $n$ . Therefore, by Theorems 3 and 4 there exist graphs of size in the billions (comparable to the estimated size of the web graph) where, assuming a “standard” damping factor of 0.85, the PageRank scores of the top 10 nodes are  $\approx 0.1$ -separated, which in absolute terms is *orders of magnitude* greater than the average PageRank score  $\frac{1}{n}$ , yet any “useful” (i.e. with a reasonable confidence level) local  $\epsilon$ -ranking algorithm incurs  $\approx 100k$  queries (increasing to  $\approx 100M$  for  $\epsilon \approx 10^{-7}$ ) in the randomized case and  $\approx 1B$  queries in the deterministic case.

It is now clear that, at least *in theory*, a general efficient local ranking algorithm does not exist. In practice, real graphs may behave differently, and state-of-the-art algorithms may be able to exploit their properties to efficiently compute the correct ranking of the target nodes. The next section provides experimental results to investigate this issue.

## 6. EXPERIMENTS

This section describes two experiments, both partially leveraging on known score approximation algorithms ([3, 12]), aimed at estimating how far the theoretical lower bounds of Section 4 are in practice from the cost of local ranking on real graphs. Subsection 6.1 introduces the experimental setting, while Subsections 6.2 and 6.3 detail the two experiments and the results obtained.

### 6.1 Experimental Design

We ran the experiments on two publicly available real graphs provided by the Laboratory for Web Algorithmics, University of Milan (see [5, 7]). The first is a large snapshot (over 40M nodes and 1150M arcs) of the 2004 .it web domain; since the structure of the web graph is self-similar (i.e. fractal-like) [15], these experimental results naturally extend to the whole web graph. Furthermore, the .it graph is relatively isolated from the rest of the web because most pages are written in Italian, a language seldom used outside this

domain; thus, this graph is well suited for link analysis experiments – carving it out of the web does not imply deleting many links. The second graph is a fairly large snapshot (over 5M nodes and 79M arcs) of the 2008 LiveJournal friendship graph, where nodes represent users and a directed arc from  $u$  to  $v$  means that  $u$  repotes  $v$  as a friend (the graph is not symmetric). Conferral of importance has a natural meaning in this graph, and its nature is completely different from that of the web, providing a wider experimental basis to test our theory. Furthermore, being a completely isolated graph, no arcs have been discarded by the crawling process.

Graph preprocessing deserves a special note. It is well known that PageRank treats dangling nodes as having virtual outgoing arcs towards all the nodes of the graph [24]. This makes it difficult to approximate PageRank scores locally since the influence of these virtual arcs can only be guessed – the link server provides only the *real* arcs – unless their number is known a priori, in which case the correct scores can be computed applying a scaling factor that depends only on the fraction of dangling nodes in the graph [14, 6]. Surprisingly, the literature on local score approximation always neglects this issue. However, the estimation of such fraction is beyond the scope of this paper. This led us to eliminate all the dangling nodes by repeated pruning, as done in [12]. For both graphs, this removed less than 15% of the nodes and less than 6% of the arcs.

For the sake of clarity, in both experiments we focused on ranking *pairs* of target nodes, which we sampled among the top ranked nodes of the graphs. This latter choice has two reasons. First, in most practical applications only the top scores really matter. Second, nodes with a high score are likely to have many ancestors on many layers, and therefore are the best candidates to test the worst-case lower bounds provided by theorems of Section 4. We thus computed the exact PageRank scores of all the nodes of the graphs, using 100 iterations of the power method and a “standard” value of 0.85 for  $\alpha$ , and selected the top 10k nodes (i.e. less than the top 0.3%) from both graphs. Then we sampled 1000 pairs uniformly at random among all the pairs of nodes  $u, v$  that are  $(1 + \epsilon, 1 + 2\epsilon)$ -separated, i.e.  $(1 + \epsilon)P(v) \leq P(u) \leq (1 + 2\epsilon)P(v)$ , for values of  $\epsilon$  ranging from  $0.01 \cdot 2^0$  (scores differing by 1-2%) to  $0.01 \cdot 2^8$  (scores differing by 256-512%) in doubling steps. On each sample we ran the experiments detailed in the following two subsections.

### 6.2 Is Ranking Local in Real Graphs?

The goal of the first experiment is to estimate a lower bound on the cost of locally ranking  $\epsilon$ -separated nodes as a function of  $\epsilon$ . This bound can be seen as the size of the *minimal set* of nodes that must be visited to compute correctly the relative ranking of two target nodes:

*Definition 1.* Let  $u, v$  be nodes of graph  $G$  and let  $A$  be a local ranking algorithm. The *minimal set*  $S_A(G, u, v)$  for  $A$  contains the minimal number of nodes of  $G$  that  $A$  fetches to correctly rank  $u$  and  $v$ .

In other words,  $S_A(G, u, v)$  is the (not necessarily unique) smallest set of nodes such that  $A$  correctly ranks  $u$  and  $v$  if it fetches any set  $S \supseteq S_A(G, u, v)$ . Clearly, a general (i.e. valid for all algorithms) lower bound on the size of  $S_A(G, u, v)$  gives a lower bound on the cost of locally  $\epsilon$ -ranking  $u$  and  $v$  in  $G$ . Note that, for any choice of  $G, u$  and  $v$ , there always exists an algorithm  $A$  such that  $S_A(G, u, v) = \emptyset$ . For

example, a trivial algorithm returning a constant ranking could rank correctly some of the pairs of nodes in  $G$ ; unfortunately, it would rank incorrectly all the remaining. It is therefore natural to restrict the definition to algorithms that provide a correct ranking for *any* pair of target nodes  $u$  and  $v$  of *any* possible graph  $G$  – in other words, algorithms that *solve* the  $\epsilon$ -ranking problem. We further restrict to algorithms that compute approximate scores as the sum of the (known) contributions of the visited nodes on the target nodes, and thus do not overestimate the real score, similarly to existing algorithms (we conjecture that *any* algorithm that behaves differently would fail for some input  $G, u, v$ ). In this case, a(n ideal) “perfect” algorithm  $PER$  would build a set  $S_{PER}(G, u, v)$  containing all and only the top  $c$  contributors of the highest ranked node  $u$  for a sufficiently large  $c$  such that  $\tilde{P}(u) = \sum_{z \in S_{PER}(G, u, v)} \text{contrib}(z, u) \geq P(v)$ . It is easy to see that, on any superset  $S \supseteq S_{PER}(G, u, v)$ , algorithm  $PER$  estimates  $\tilde{P}(u) \geq P(v)$  and  $\tilde{P}(v) \leq P(v)$ , and infers the correct ranking from the inequality  $\tilde{P}(u) \geq \tilde{P}(v)$ , while any set smaller than  $S_{PER}(G, u, v)$  would lead to  $\tilde{P}(u) < P(v)$  and to an incorrect result for some  $(G, u, v)$ . Therefore the cardinality of  $S_{PER}(G, u, v)$  is a lower bound to the cost of locally ranking  $u$  and  $v$  in  $G$  for any correct algorithm. Note that, since the contribution of an ancestor  $z$  is  $\alpha$  times the average of the contributions of its children, at least one of them has a contribution not smaller than that of  $z$ . Therefore, if  $S_{PER}(G, u, v)$  contains  $z$ , it also contains one of its children, and thus  $S_{PER}(G, u, v)$  induces a connected graph which is in principle *reachable* by those (reasonable) algorithms that query only parents of already-queried nodes.

In practice, for each pair  $u, v$  we estimated their minimal set as follows. We collected the first 15 layers of the ancestors of  $u$ , stopping earlier if their number reached a given threshold ( $0.02n$  for the `.it` graph and  $0.2n$  for the LiveJournal graph). These ancestors induced a subgraph where we ran 40 iterations of the brute force algorithm with  $u$  as the target node. This yielded the contribution of each collected ancestor, which was used to sort them and build the estimated minimal set.

Figure 5 illustrates the average size of the estimated minimal sets as a function of  $\epsilon$  for both graphs. As expected, this size increases as  $\epsilon$  decreases – ranking weakly-separated nodes costs more than ranking well-separated ones. In the `.it` web graph, the cost of local ranking is 100 to 1000 times smaller than the size of the graph, i.e. in the order of  $10^4 - 10^5$  queries, which may be intolerably high for applications that use a remote link server. Surprisingly, in the LiveJournal graph minimal sets are much larger, in spite of the significantly lower graph size ( $< 4.8M$  vs.  $> 37M$  nodes) and average degree ( $< 16.5$  vs.  $> 30$ ). Indeed, except for  $\epsilon \approx 1$ , the number of collected ancestors almost always reached the threshold of  $0.2n$  and their contribution was not sufficient to give a correct ranking. Thus,  $\epsilon$ -ranking in the LiveJournal graph is strongly non-local.

### 6.3 Local Ranking with Real Algorithms

The second experiment evaluates the performance of two real (returning *wrong* results in some cases) algorithms, giving an empirical upper bound on the cost of local ranking if one accepts an unavoidably positive rate of error.

We first tested the brute force (BF) algorithm, which serves more as a benchmark than as a realistic efficient method for local ranking. BF explores the ancestors of each

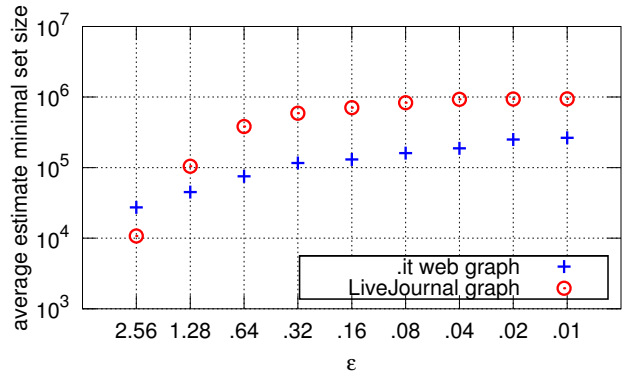


Figure 5: Average size of estimated minimal sets for  $(\epsilon, 2\epsilon)$ -separated nodes as a function of  $\epsilon$  for the `.it` web graph and the LiveJournal graph

of the two target nodes layer by layer, computing their approximate scores according to Equation (2) and inferring their relative ranking. Figures 6.1 - 6.2 (`.it` graph) and 6.3 - 6.4 (LiveJournal graph) show respectively the average cost (number of visited nodes) and the average precision (fraction of correctly ranked node pairs), for different values of  $(\epsilon, 2\epsilon)$ , as a function of the number of layers visited from 0 to 25 – a limit that seems sufficient to saturate the set of visited nodes. Unsurprisingly, BF incurs overwhelmingly high costs (several million queries) in both graphs, even to guarantee a precision  $\geq 0.9$  only for the most separated nodes.

We then tested ImPBF, an improved version of PBF, the pruned brute force algorithm [3]. PBF is an intuitively efficient variation of the BF algorithm that, at each layer  $\tau \geq 1$ , visits only the nodes whose estimated contribution (over paths known at iteration  $\tau$ ) to the score of the target node is above a given threshold. The conjecture underpinning this heuristics is that nodes with a small contribution over paths of length  $\leq \tau$  probably give a small *overall* contribution, and their parents likely do the same. Counterexamples show that sometimes this is false [4], but [12] shows that this heuristics works on real web graphs; and although other algorithms for local score approximation exist (see [12]), they have less clear theoretical backgrounds, are unrealistic (e.g. assuming to know the total number of arcs), or sometimes overestimate the true PageRank score – all assumptions not fitting our general model. We terminated PBF either when no more candidates had a sufficient estimated contribution or when it visited more than  $0.1n$  nodes, which in practice is  $\Theta(n)$ . On the subgraph induced by the visited nodes, ImPBF runs BF for 40 iterations (while PBF typically did not visit more than 20 layers) or until the relative per-iteration increment in the estimated score drops below 0.1%. This “squeezes” most of the overall contribution out of the visited nodes, giving a more accurate score approximation.

We ran ImPBF for different contribution thresholds, ranging from  $10^{-1}$  to  $10^{-7}$ . Figures 6.5 - 6.6 (`.it` graph) and 6.7 - 6.8 (LiveJournal graph) show respectively the average cost and the average precision of the ImPBF algorithm as a function of the contribution threshold, for different values of  $(\epsilon, 2\epsilon)$ . In the `.it` web graph, ImPBF reached a precision  $\geq 0.9$  for  $\epsilon > 0.02$  incurring a cost 2.5 to 5 times the estimated minimal set size and is thus reasonably close to the optimum. In the LiveJournal graph, ImPBF guaranteed

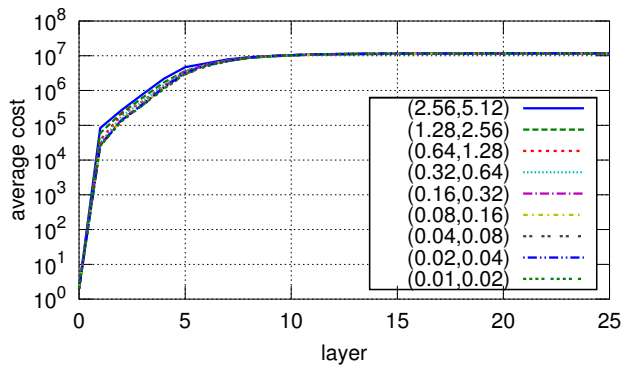


Figure 6.1: .it web graph, BF algorithm: average cost vs. layers visited, for different values of  $(\epsilon, 2\epsilon)$

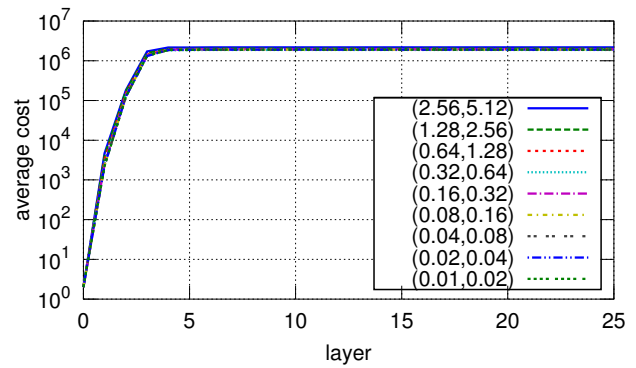


Figure 6.3: LiveJournal graph, BF algorithm: average cost vs. layers visited, for different values of  $(\epsilon, 2\epsilon)$

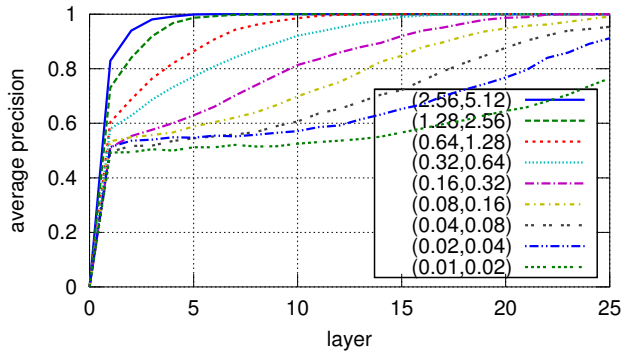


Figure 6.2: .it web graph, BF algorithm: average precision vs. layers visited, for different values of  $(\epsilon, 2\epsilon)$

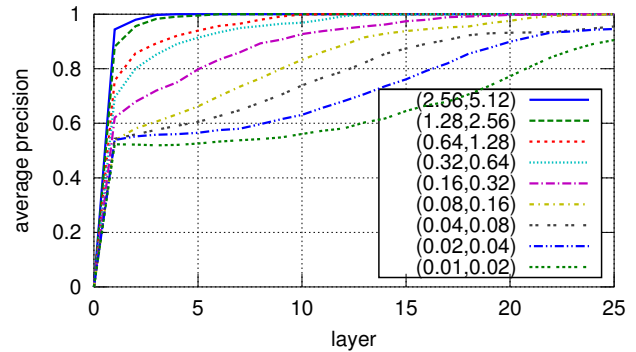


Figure 6.4: LiveJournal graph, BF algorithm: average precision vs. layers visited, for different values of  $(\epsilon, 2\epsilon)$

a precision  $\geq 0.9$  only for  $\epsilon \geq 0.64$ , incurring a cost of a few thousand nodes, or  $\approx 0.1$  times the average estimated minimal set size. For  $\epsilon < 0.64$ , its precision dropped below 0.9, falling below 0.7 for  $\epsilon < 0.16$ , and the cost increased to a large fraction of the estimated minimal set size – confirming that ranking in this graph is strongly non-local except for extremely-separated nodes. Counterintuitively, the precision of ImpBF in the LiveJournal graph tends to decrease as the contribution threshold lowers and the cost increases. This is likely due to the saturation of the  $0.1n$  cost threshold and confirms that local ranking is definitely non-trivial.

## 7. CONCLUSIONS

Motivated by a large and growing number of applications (such as web crawling and social network mining) that use PageRank as a pure ranking algorithm, we define and investigate the problem of computing locally the relative PageRank ranking of a set of  $k$  nodes in an  $n$ -node graph. We prove that a general, efficient local ranking algorithm does not exist: in the worst case, any correct algorithm must consider  $\Omega(\sqrt{kn})$  nodes if randomized and  $\Omega(n)$  nodes if deterministic, even when ranking the top  $k$  nodes in the graph and even if their scores are “well separated”. We show experimentally that, in practice, real (web and social) graphs behave almost as poorly, requiring intolerably high costs both for exact algorithms and for algorithms based on state-of-the-art local score approximation algorithms.

Two questions naturally arise from our theoretical and experimental results. First, what are the topological properties of a graph that make it a hard, or instead, easy instance for

local ranking? Second, how can these properties be exploited by local algorithms to compute the ranking efficiently?

## 8. ACKNOWLEDGEMENTS

This work was supported by Univ. Padova Strategic Proj. AACSE, by MIUR Proj. AlgoDEEP, and by PAT, FBK, and INFN Proj. Aurora-Science. We thank Massimo Melucci, Enoch Peserico, Geppino Pucci, Michele Scquizzato, Francesco Silvestri, and the anonymous reviewers for their valuable comments; Paolo Bertasi and Michele Bonazza for setting up the computing infrastructure.

## 9. REFERENCES

- [1] B. Amento, L. Terveen, and W. Hill. Does “authority” mean quality? Predicting expert quality ratings of web documents. In *Proc. of ACM SIGIR*, pages 296–303. 2000.
- [2] R. Andersen, C. Borgs, J. Chayes, J. Hopcroft, V. Mirrokni, and S.-H. Teng. Local computation of PageRank contributions. *Internet Mathematics*, 5(1–2):23–45, 2008.
- [3] Z. Bar-Yossef and L.-T. Mashiach. Local approximation of PageRank and reverse PageRank. In *Proc. of ACM CIKM*, pages 279–288. 2008.
- [4] Z. Bar-Yossef and L.-T. Mashiach. Local approximation of PageRank and reverse PageRank. Technical report, Israel Institute of Technology, 2008.
- [5] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.
- [6] P. Boldi, R. Posenato, M. Santini, and S. Vigna. Traps and pitfalls of topic-biased PageRank. In *Proc. of WAW*, pages 107–116. 2006.
- [7] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proc. of WWW*, pages 595–601. 2004.

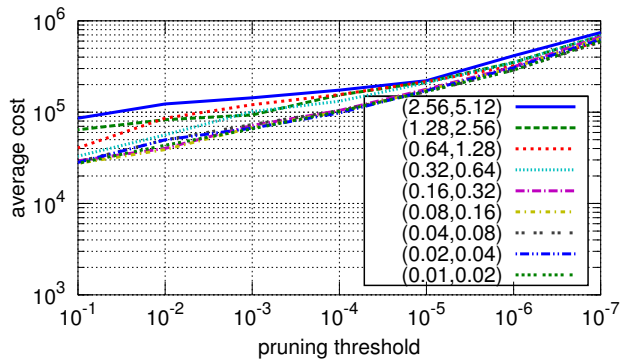


Figure 6.5: .it graph, ImPBF algorithm: average cost vs. contribution threshold, for different values of  $(\epsilon, 2\epsilon)$

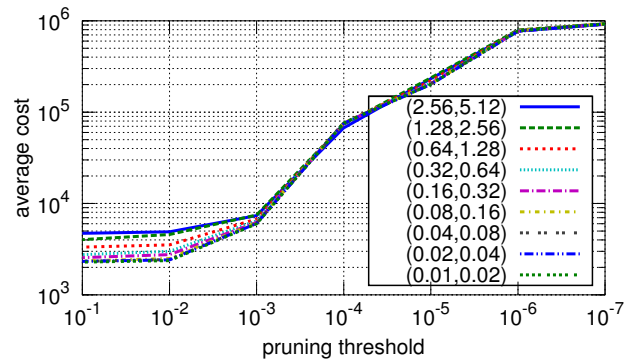


Figure 6.7: LiveJournal graph, ImPBF algorithm: avg. cost vs. contrib. threshold, for different values of  $(\epsilon, 2\epsilon)$

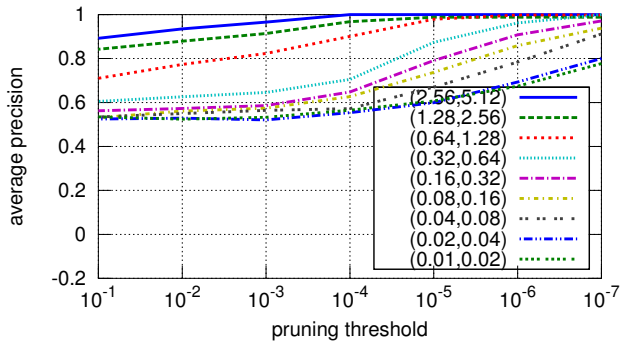


Figure 6.6: .it graph, ImPBF algorithm: average precision vs. contribution threshold, for different values of  $(\epsilon, 2\epsilon)$

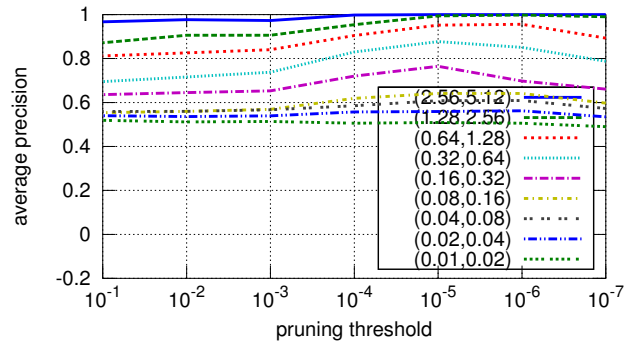


Figure 6.8: LiveJournal graph, ImPBF algorithm: avg. precision vs. contrib. threshold, for different values of  $(\epsilon, 2\epsilon)$

- [8] A. Borodin, G. O. Roberts, J. S. Rosenthal, and P. Tsaparas. Link analysis ranking: Algorithms, theory, and experiments. *ACM Transactions on Internet Technology*, 5(1):231–297, 2005.
- [9] M. Bressan and E. Peserico. Choose the damping, choose the ranking? *Journal of Discrete Algorithms*, 8(2):199–213, 2010.
- [10] S. Brin and L. Page. The anatomy of a large scale hypertextual Web search engine. In *Proc. of WWW*. 1998.
- [11] H. Buhrman and R. D. Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288(1):21–43, 2002.
- [12] Y.-Y. Chen, Q. Gan, and T. Suel. Local methods for estimating PageRank values. In *Proc. of ACM CIKM*, pages 381–389. 2004.
- [13] J. Cho, H. Garcia-Molina, and L. Page. Efficient crawling through URL ordering. *Computer Networks*, 30(1–7):161–172, 1998.
- [14] G. M. Del Corso, A. Gulli, and F. Romani. Fast PageRank Computation via a Sparse Linear System. *Internet Mathematics*, 2(3):251–273, 2005.
- [15] S. Dill, R. Kumar, K. S. McCurley, S. Rajagopalan, D. Sivakumar, and A. Tomkins. Self-similarity in the web. *ACM Transactions on Internet Technology*, 2(3):205–223, 2002.
- [16] A. Esuli and F. Sebastiani. PageRanking WordNet synsets: An application to opinion mining. In *Proc. of ACL*, pages 424–431. 2007.
- [17] F. Geerts, H. Mannila, and E. Terzi. Relational link-based ranking. In *Proc. of VLDB*, pages 552–563. 2004.
- [18] R. Gross and A. Acquisti. Information revelation and privacy in online social networks. In *Proc. of ACM WPES*, pages 71–80. 2005.
- [19] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with TrustRank. In *Proc. of VLDB*, pages 576–587. 2004.
- [20] J. Heidemann, M. Klier, and F. Probst. Identifying key users in online social networks: A PageRank based approach. In *Proc. of ICIS*, 2010.
- [21] J. L. Hennessy and D. A. Patterson. Computer architecture: A quantitative approach. 4th ed. Morgan Kaufmann, 2007.
- [22] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *Proc. of WWW*, pages 508–516. 2003.
- [23] O. Kurland and L. Lee. PageRank without hyperlinks: Structural re-ranking using links induced by language models. In *Proc. of ACM SIGIR*, pages 306–313. 2005.
- [24] A. N. Langville and C. D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.
- [25] R. Lempel and S. Moran. Rank-stability and rank-similarity of link-based Web ranking algorithms in authority-connected graphs. *Information Retrieval*, 8:219–243, 2005.
- [26] M. Melucci and L. Pretto. PageRank: When order changes. In *Proc. of ECIR*, pages 581–588. 2007.
- [27] J. L. Morrison, R. Breitling, D. J. Higham, and D. R. Gilbert. GeneRank: Using search engine technology for the analysis of microarray experiments. *BMC Bioinformatics*, 6:233, 2005.
- [28] E. Peserico and L. Pretto. What does it mean to converge in rank? In *Proc. of ICTIR*, pages 239–245. 2007.
- [29] P. Tarau, R. Mihalcea, and E. Figa. Semantic document engineering with WordNet and PageRank. In *Proc. of ACM SAC*, pages 782–786. 2005.
- [30] R. S. Wills and I. C. F. Ipsen. Ordinal ranking for Google's PageRank. *SIAM Journal on Matrix Analysis and Applications*, 30(4):1677–1696, 2009.
- [31] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2007.
- [32] S. Yan and D. Lee. Toward alternative measures for ranking venues: a case of database research community. In *Proc. of ACM/IEEE JCDL*, pages 235–244. 2007.
- [33] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. of IEEE FOCS*, pages 222–227. 1977.