Università
degli Studi
di Padova

DEPARTMENT OF
INFORMATION
ENGINEERING
UNIVERSITY OF PADOVA

Facoltà di Ingegneria
Dipartimento di Ingegneria dell'Informazione

Scuola di Dottorato di Ricerca in Ingegneria dell'Informazione – XXII Ciclo
Indirizzo: *Scienza e Tecnologia dell'Informazione e della Comunicazione*

Ranking Robustly

Direttore della Scuola
Ch.mo Prof. Matteo Bertocco

Supervisore
Ch.mo Prof. Enoch Peserico

Dottorando
Marco Bressan

Abstract

Is the ranking induced by PageRank robust with respect to factors that, intuitively, should have little weight? Motivated by a large and growing number of applications of PageRank as a ranking algorithm, we investigate this problem theoretically and experimentally along two complementary research lines.

The first research line explores to what extent the PageRank-induced ranking of the nodes of a graph depends on an arbitrary, graph-independent, user-model parameter – the damping factor. We prove that, on some graphs, the ranking depends totally on the damping factor and that, in these cases, sampling the rank of a node on any finite set of damping factors gives very little information about its overall stability. The novel tool of *lineage analysis* bypasses the problem and allows to check if the rank of a node is stable for all (even time-variant) damping factors. We introduce the notions of *strong rank* and *weak rank*, which measure the rank robustness of a graph's nodes, and derive two new ranking metrics that benchmark the performance of ranking algorithms with respect to different scenarios. Experimental results show that, in real-world graphs, ranking is relatively robust, and suggest ideal damping factors for PageRank in different application domains.

The second research line investigates whether it is possible to compute the relative PageRank-induced ranking of a node visiting only a small nearby subgraph. We answer negatively: to provide a correct ranking any algorithm must visit a number of nodes that is proportional to the size of the graph, if deterministic, or to its square root, if randomized. These results hold even when ranking the top nodes in the graph, even when the gap between their PageRank scores is large. Indeed our experiments show that, in some real-world cases, any algorithm must visit large number of nodes, even when inferring ranking through efficient local approximations of the PageRank scores. Therefore, ranking seems definitely not robust with respect to the removal of nodes from the graph.

In a nutshell, this work asks how much "information" a graph contains about the PageRank-induced importance of its nodes and whether this information is local or distributed – aiming at a full understanding of the robustness of PageRank as a ranking algorithm. It turns out, that, in terms of variations of the damping factor and of variations of the link structure in "remote" areas of the graph, PageRank is not very robust – neither in theory nor, in many cases, in practice.

Sommario

Il ranking indotto da PageRank è robusto rispetto a fattori che, intuitivamente, dovrebbero avere poco peso? Motivati da un ampio e crescente numero di applicazioni di PageRank come algoritmo di ranking, investighiamo questo problema teoricamente e sperimentalmente lungo due linee di ricerca complementari.

La prima linea di ricerca esplora quanto il ranking indotto da PageRank sui nodi di un grafo dipenda da un parametro del modello sottostante, arbitrario e indipendente dal grafo – il damping factor. Mostriamo che, in alcuni grafi, il ranking dipende totalmente dal damping factor a che, in questi casi, campionare il rank di un nodo per qualsiasi insieme finito di damping factor dà pochissima informazione sulla sua stabilità complessiva. Il nuovo strumento della *lineage analysis* supera il problema e permette di verificare se il rank di un nodo è stabile per tutti i damping factor, anche tempo-varianti. Introduciamo le nozioni di *strong rank* e *weak rank*, che misurano la robustezza dei rank dei nodi di un grafo, e deriviamo due nuove metriche che misurano le prestazioni degli algoritmi di ranking rispetto a differenti scenari. I risultati sperimentali mostrano che, nei grafi reali, il ranking è relativamente robusto, e suggeriscono damping factor ideali per PageRank in diversi domini applicativi.

La seconda linea di ricerca investiga se sia possibile calcolare il ranking relativo (indotto da PageRank) di un nodo visitando solo un piccolo sottografo circostante. Rispondiamo negativamente: per fornire un ranking corretto, ogni algoritmo deve visitare un numero di nodi che è proporzionale alla taglia del grafo, nel caso deterministico, e proporzionale alla sua radice quadrata, se randomizzato. Questi risultati valgono anche quando si fornisce il ranking dei nodi più importanti del grafo e la differenza tra i loro punteggi PageRank è ampia. In effetti i nostri esperimenti mostrano che, in alcuni grafi reali, ogni algoritmo deve visitare un grande numero di nodi, anche se utilizza approssimazioni efficienti del punteggio PageRank. Quindi, il ranking sembra essere non robusto rispetto alla rimozione di nodi dal grafo.

In breve, questo lavoro si chiede quanta “informazione” contiene un grafo circa l'importanza (data da PageRank) dei suoi nodi e se questa informazione sia localizzata o distribuita – mirando a una piena comprensione della robustezza di PageRank come algoritmo di ranking. Si scopre che, in termini di variazioni del damping factor e di variazione nella struttura in aree “remote” del grafo, PageRank non è molto robusto – né in teoria né, in molti casi, in pratica.

Acknowledgements

It is both a duty and a pleasure to thank the many people who helped me, supported me, and contributed to this work. It is also their merit if I successfully concluded my Ph.D.

First of all I want to thank my advisor Enoch Peserico for taking me through what, I believe, has been the best Ph.D. one could desire. He taught me how to be a well-rounded scientist, conduct beautiful research, and keep a critical but positive attitude; or at least he tried, for I still have much to learn.

I am really grateful to all the members, current and former, of the Advanced Computing Group. I thank Gianfranco Bilardi, Carlo Fantozzi, Andrea Pietracaprina, and Geppino Pucci as the leaders of the research group; and I thank my friends and colleagues Fausto Artico, Paolo Bertasi, Alberto Bertoldo, Federica Bogo, Emanuele Milani, Alberto Pettarin, Michele Scquizzato, Michele Schimd, Francesco Silvestri, Fabio Vandin, and Francesco Versaci, for their inestimable company in these years.

A special thank goes to Luca Pretto. His help and unfailing support were invaluable for my research and played a large part in this work.

I want to give a sincere thank to the Ph.D. School secretary, Alessandra Calore, on whom I relentlessly relied during my work.

Last but not least, I have to thank all my friends for their precious presence and, most important, my parents that have made this possible.

This work was supported in part by:

- University of Padova under Strategic Project STPD08JA32 “AACSE - Algorithms and Architectures for Computational Science and Engineering” and under a 3-years Ph.D. fellowship.

- Italian Ministry of University and Research (MIUR) under project AlgoDEEP prot. 2008TFBWL4.

- EU under the EU/IST Project 15964 “AEOLUS - Algorithmic Principles for Building Efficient Overlay Computers”.

Contents

Chapter 1

Introduction

This introductory chapter motivates our research and provides related work (Section 1.1), briefly reviews the PageRank algorithm (Section 1.2) and the crucial difference between score and rank(Section 1.3), before presenting an overview of our results and the organization of the rest of the thesis (Section 1.4).

1.1 Motivation and related work

PageRank [14] is probably the best known of *link analysis* algorithms; originally proposed as a means to infer the importance of web pages from the link structure of the web graph, today it is a reference algorithm in computer science, especially for the information retrieval, database and natural language processing 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: web crawling [17], credit and reputation systems [28], ranking in databases [23], text summarization [20], combating web spam [25], structural re-ranking [31], word sense disambiguation [41], ranking WordNet synsets [21], and graph-based natural language processing [40]. This suggests to study the algorithm in the abstract, considering it as a graph algorithm which accepts a graph in input and provides the scores of each node of the graph as output.

While in the original web application PageRank scores were combined with traditional text-based information retrieval scores, many of the other applications simply use the unmodified PageRank scores to rank items in order of precedence, whether for efficiency or by lack of other valid alternatives. For example, in web crawling scores are used to decide ‘in what order a crawler should visit the URLs it has seen, in order to obtain more “important” pages first’ [17]. This fact naturally moved research towards the ranking induced by PageRank, focusing on the convergence in rank of its iterative computation [36, 44] and on the dependence of PageRank-induced ranking on user-model factors [34, 13]. This suggests to consider PageRank as an algorithm to rank the nodes of a generic graph and to investigate its robustness to perturbations that should intuitively have a limited impact. In particular, we follow the two main research lines detailed in the following.

In the first research line, we investigate to what extent the ranking depends on (variations in) the damping factor, an arbitrary graph-independent parameter related to the user model, likely to differ between users, and hard to assess precisely (see Section 1.2). It is well understood [39, 10] that the PageRank *score* of each node of the graph changes with the damping factor. For example, a damping factor equal to zero assigns to all nodes the same score (independently from the graph structure), while progressively higher damping factors tend to "take more into account" the underlying link structure. Previous work proposed to assign a more "objective" score to each node as the average score over all possible damping factors [8], or generalized the damping factor to damping functions [5]. While it is also known that variations of the damping factor can affect the relative ranking of two or more nodes [34], its impact is not well understood nor it is known how heavy it is in general. Clearly, the ranking induced by PageRank elicits confidence only when relatively insensitive to small variations in the value of the damping factor. This compels to investigate this sensitivity – or in other words, the robustness of PageRank – to variations in the damping factor.

In the second research line, we investigate if it is possible to compute the relative rank of a node by discarding most of its ancestors and considering only a small subgraph around that node, effectively "pruning" the input graph. A vast body of research has indeed been carried out on the local computation of PageRank *scores*. This research is motivated by the fact that often the input graph is too large to fit in main memory, and in some cases the complete graph is not even available, instead being available only a local picture of the input graph. This happens, for instance, to a web user who can access the web graph only by querying a search engine with the '`link:`' option, or to a social network user who can only fetch the links of the graph (representing, for example, friendship relationships) by browsing the online profiles of other users. 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, some experimental research work in information retrieval actually do this (see e.g. [2]). To this regard, [16] provides heuristics to locally approximate PageRank scores, [3] 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 [6] proves the infeasibility of locally approximating the PageRank scores in the worst case, and provides lower bounds for deterministic and randomized local approximation algorithms. This raises the problem of locally computing not (only) the scores but (also) the relative rankings of the nodes in the graph, and lead to ask if it is possible to compute a correct ranking considering only a small subset of nodes – in other words, if the ranking induced by PageRank is robust under (intuitively "good") changes in the input graph.

1.2 PageRank

In its original form, PageRank is based on a model of a web surfer that probabilistically browses the web graph, starting at a node chosen at random according to the distribution given by a *personalization vector* $\mathbf{e} > 0$ whose generic component e_v is the probability of starting at node v; unless otherwise specified, in the following we adopt $\mathbf{e} = [1/n, \dots, 1/n]$ sake of simplicity. At each step, if the current node has outgoing links to $m > 0$ other nodes $v_1, \dots, v_m$, the surfer next browses with probability α one of those nodes (chosen uniformly at random), and with probability $1 - \alpha$ a node chosen at random according to $\mathbf{e}$. If the current node is a dangling node (i.e. it has no outgoing links) the surfer automatically chooses the next node at random according to $\mathbf{e}$. If the *damping factor* α is less than 1 (for the web graph a popular value is ≈ 0.85 [32, 35]), the probability $P_v(t)$ of visiting the node v at time t converges for $t \to \infty$ to a stationary probability P_v; this is the score PageRank assigns to v.

More formally, consider an n-node graph $G = (V, A)$ with no dangling nodes (i.e. no 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, modeling the fact that a web user getting stuck on a web page without outgoing links would restart the browsing process from a random page. The entries of the transition probability matrix $\mathbf{M} = [m_{i,j}]$ of G are

$$m_{i,j} = \begin{cases} \frac{1}{\text{outdegree}(i)} & \text{if an arc exists from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

Let the damping factor α be a real number between 0 and 1, and let

$$\mathbf{T} = \alpha\mathbf{M} + (1 - \alpha)\mathbf{U} \tag{1.2}$$

where $\mathbf{U}$ is a matrix whose entries equal $1/n$. The matrix $\mathbf{T}$ can be seen as the transition probability matrix of the Markov chain describing the browsing process of the random web surfer described above. The PageRank row vector $\mathbf{P} = [P(v_1), P(v_2), \ldots, P(v_n)]$ 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

$$\mathbf{P} = \mathbf{PT} \tag{1.3}$$

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 an arbitrary probability vector.

Intuitively, the more ancestors (both immediate and far removed) a node has, and the fewer descendants those ancestors have, the higher the score of that node. This is synthesized in the following expression of PageRank as a power series, obtained applying straightforward algebraic manipulations to Equation (1.3):

$$\mathbf{P} = \frac{1-\alpha}{n}\mathbf{1}(\mathbf{I} - \alpha\mathbf{M})^{-1} = \frac{1-\alpha}{n}\mathbf{1}\sum_{\tau=0}^{+\infty}(\alpha\mathbf{M})^{\tau} \tag{1.4}$$

where $\mathbf{1} = [1, 1, \ldots, 1]$. This leads to the following natural, intuitive characterization of the PageRank score of a node v:

$$P(v) = \frac{1-\alpha}{n}\sum_{\tau=0}^{+\infty}\alpha^{\tau}\sum_{z\in G}\mathrm{inf}_{\tau}(z, v) \tag{1.5}$$

where the *influence* $\mathrm{inf}_{\tau}(z, v)$ of z on v gives the probability of ending in v by following one of the chains of τ arcs from z given that no random jumps are performed. Equivalently, one can express $P(v)$ as:

$$P(v) = \sum_{z\in G}\mathrm{contrib}(z, v) \tag{1.6}$$

where the *contribution* $\mathrm{contrib}(z, v)$ of z to v is the total probability of ending in v from z. Therefore, the score of node v can be seen either as the weighted sum, over each layer $\tau = 1, \ldots, +\infty$, of the aggregate influence of its ancestors at layer τ, or as the sum, over each node z in the graph, of the contribution of z to v.

1.3 Score vs. rank

Formally – see Equation (1.3) – PageRank can be defined as the dominant eigenvector of the stochastic matrix $\mathbf{T}$ and is usually computed using the Power Method [32] or one of its derivations. Thus, one can analyse the impact of input variations on the score vector using the highly developed toolset of linear algebra [32]. This is no longer true when dealing with rank: loss of linearity and continuity in the mathematical model make analysis considerably more difficult and require different tools. Unfortunately, as remarked in Section 1.1, in many cases rank is far more important than score, effectively turning PageRank into a pure ranking algorithm; and since both the score of a node and its rank may change as the input changes, it is fundamental to understand their crucial difference.

On the one hand, the damping factor affects the score and thus the rank. Equation (1.5) shows that lower damping factors decrease the likelihood of following long paths of links without taking a random jump, and thus increase the contribution to a node of its more immediate ancestors. It is well known, however, that the sensitivity of the score vector to the damping factor is bounded; therefore, PageRank vectors obtained for close values of the damping factor are substantially equivalent for what concerns the *score*. However, while score is a continuous function of the damping factor, rank is not; and nothing guarantees any limit on how rapidly the global rank vector may change in response to variations in the damping factor (one could for example measure this sensitivity as the ratio between the Kendall rank correlation coefficient of two rank vectors and the difference of the two damping factors). Therefore, even abysmally small variations in the damping factor (and in the *score* vector!) could in principle lead to enormous variations in the *rank* vector.

On the other hand, "approximating" the rank may be hard. Equation (1.6) suggests to compute (an approximation of) the PageRank score of a node by considering only a small subset of (highly) contributing ancestors. Although in practice this appears to yield good results, in theory even approximating PageRank scores using only a subset of ancestors is impossible in the worst case [6]. It is not known, however, if the same results hold for the *ranking* induced by PageRank, even if one could think that it might be obtained "easier" than the score – after all, the correct ranking can be reached with scores even remotely close to the correct PageRank scores, while the opposite is trivially false. But while Equation (1.6) suggests that taking into account the contributions of more ancestors should always *improve* the approximation on the *score* of a node, which eventually converges to its exact value, it is much harder to say the same for rank – it could well happen that two nodes "churn around" in the global ranking as more and more of their ancestors are taken into account, leading to an extremely slow convergence (if any).

Therefore, rank is important because in principle it may exhibit extremely different characteristics – such as much higher sensitivity and much slower convergence – than the score (see as [36]) and deserves an independent analysis. This issue concerns not only PageRank but also *every* algorithm that infers the ranks from the scores (and has been addressed for the HITS algorithm in [37, 38]). Although beyond the scope of our work, which is focused on PageRank, this more general problem is certainly an interesting research direction.

1.4 Outline of the thesis

This section summarizes our results and illustrate their organization in the rest of the thesis.

Chapter 2 is devoted to the first research line, discussing to what extent variations in the damping factor affect the PageRank-induced ranking of nodes. Section 2.2 shows that, at least on some graphs, the PageRank-induced ranking of nodes depends completely on arbitrarily small variations of the damping factor. Section 2.3 provides analytical tools to verify if a node outranks another simultaneously for all (time-variant) damping factors and gives to this fact an intuitive justification, totally independent of PageRank, based on the novel concept of *lineage* of nodes. Section 2.4 introduces the novel concepts of *strong rank* and *weak rank*, which measure the "fuzziness" of the ranking of a node and of a whole graph, and also allow an evaluation of PageRank as a function of the damping factor. Section 2.5 describes our experiments on two real graphs, while Section 2.6 summarizes the results of the chapter. These results appeared in [12, 13].

Chapter 3 is devoted to the second research line. We define the problem of locally computing PageRank-induced rankings, providing the first theoretical and experimental results on this subject. Section 3.2 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 without taking into account a number of nodes linear in the size of the input graph, or may even never converge. Section 3.3 formally defines the problem of local ranking, and Section 3.4 provides the main theoretical results, showing 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 3.5 describes two experiments suggesting that the cost of local ranking in real graphs is actually high and strongly dependent on the input graph. Section 3.6 summarizes the results of of the chapter.

Finally, Chapter 4 summarizes our results and analyzes their significance, before looking at directions for future research and concluding with the bibliography.

Chapter 2

Choose The Damping, Choose The Ranking?

2.1 Introduction

This chapter addresses the fundamental question of how the ranking induced by PageRank can be affected by variations of the damping factor.

Section 2.2 shows that for any k, at least on some graphs, *arbitrarily small* variations in the damping factor can completely reverse the ranking of the top k nodes, or indeed make them assume all possible $k!$ orderings. It is natural to ask whether this can happen in "real" graphs encountered in the vast and growing number of application domains of PageRank. Experiments sampling ranking for a handful of different damping factors [24, 34] suggest this is not the case at least for the web graph (leaving open the issue of other application domains). However, verifying rank stability for discrete variations in the damping factor (e.g. 0.01 increments) is not sufficient to conclude that rank is stable as the damping factor varies over a *whole continuous interval* - just like sampling the function $f(x) = \sin(100\pi x)$ for $x = 0.01, 0.02, \dots$ is not enough to conclude that $f(x) = 0\ \forall x \in \mathbb{R}$.

Section 2.3 provides the analytical tools to address this issue. We show a simple, "natural" condition that is both necessary and sufficient to guarantee that a node outranks another *simultaneously* for all damping factors and all damping variables (informally, time-variant damping factors). This condition, based on the concept of *lineage* of a node, can be checked efficiently and has an intuitive justification totally independent of PageRank.

Section 2.4 leverages lineage analysis to introduce the novel concepts of *strong rank* and *weak rank*. These provide for each node a measure of the "fuzziness" of its ranking that is subtly but profoundly different from pure rank variation; they also provide a measure of the effectiveness of the random surfer model on a generic graph; finally, they allow an objective evaluation of the performance of “classic” ($\alpha = 0.85$) PageRank and some of its variations (e.g. $\alpha \to 0$).

Section 2.5 brings the analytical machinery of Section 2.3 to bear on two real graphs - a snapshot of the .it domain, and the CiteSeer citation graph [1]. Among other findings, we show that, on both graphs, rank is relatively stable and the ideal damping factor appears to be $0.8 - 0.9$ to obtain items of high importance to at least one user, but only $0.5 - 0.6$ to obtain those items important to every user.

Section 2.6 summarizes our results, analyses their significance, and reviews a few problems this work leaves open.

2.2 The damping makes the ranking

This section presents two theorems showing that, at least on some graphs, a minuscule variation of the damping factor can dramatically change the ranking of the top k nodes. More formally, we prove:

Theorem 2.1. *For every even $k > 1$ and every α satisfying $\frac{1}{k} < \alpha < 1 - \frac{1}{k}$, there is a graph G of $2k^2 + 4k - 2$ vertices such that PageRank's top k nodes are, in order, $\langle v_1, \ldots, v_k \rangle$ if the damping factor is α, and $\langle v_k, \ldots, v_1 \rangle$ if it is $\alpha + \frac{1}{k}$.*

Theorem 2.2. *Consider an arbitrary set Π of orderings of k nodes $v_1, \ldots, v_k$ ($|\Pi| \leq k!$), and an arbitrary open interval $\mathcal{I} \subset [0, 1]$, however small. Then there is a graph G such that PageRank's top k nodes are always $v_1, \ldots, v_k$ but appear in every order in Π as the damping factor varies within $\mathcal{I}$.*

By Theorem 2.1, there are graphs of $\approx 2M$ nodes (a size comparable to that of a citation archive or of a small first level domain) where a variation of the damping factor as small as 0.001 (e.g. from 0.850 to 0.851) can cause a complete reversal of the ranking of the top 1000 items; and the sensitivity to the damping factor can grow even higher for larger graphs. Theorem 2.2 is even more general: for any arbitrarily small interval of variation of the damping factor, there are graphs in which the top items assume *all* possible permutations (but providing bounds on the size of these graphs is beyond the scope of this work).

The rest of this section is organized as follows. Subsection 2.2.1 introduces some notation, necessary for the proofs of these two theorems, that will also be of use later. Subsection 2.2.2 is devoted to the proofs themselves, and may be skipped without impairing the understanding of the rest.

2.2.1 Notation

The score assigned by PageRank to the node v using a damping factor α, $P_v(\alpha)$ - i.e. the stationary probability of being on that node according to the model presented in Subsection 1.2 - can be seen as the sum, over all $\ell \geq 0$, of the probability of taking, ℓ timesteps in the past, the last random jump to a node v_ℓ at distance ℓ from v, and following any path of length ℓ from v_ℓ to v.

More formally, denote by $p \xrightarrow{\ell} v$ the fact that p is a path of $\ell + 1$ vertices $\langle v_\ell, \ldots, v_1, v \rangle$, from some vertex v_ℓ to v. Let branching(p) be the inverse of the product of the out-degrees $\omega_\ell, \ldots, \omega_1$ of $v_\ell, \ldots, v_1$, i.e. branching(p) $= 1/(\omega_\ell \ \ldots \ \omega_1)$ (informally, the probability of following the whole path if one starts on the first node

v_ℓ and no random jumps are taken). Then (see e.g. [5]):

$$P_v(\alpha) = \sum_{\ell=0}^{+\infty} (1-\alpha)\alpha^\ell \sum_{p \xrightarrow{\ell} v} \text{branching}(p) \cdot \mathbf{e}_{v_\ell} \tag{2.1}$$

where $\mathbf{e}_{v_\ell}$ is the entry of $\mathbf{e}$ associated with the first node, v_ℓ, of the path p. It is easy to verify that the term $\sum_{p \xrightarrow{\ell} v} \text{branching}(p) \cdot \mathbf{e}_{v_\ell}$, the *branching contribution at level* ℓ, equals the sum of the row of $\mathbf{M}^\ell$ corresponding to v, each weighted by the corresponding component of $\mathbf{e}$. Intuitively, this term corresponds to the score a node v would hold after ℓ timesteps if each node v_i started with a score $\mathbf{e}_{v_i}$ and, at each timestep, bequeathed all of its score dividing it evenly among its children. Note that branching contribution is completely independent of the damping factor; whereas the other term of the product, $(1-\alpha)\alpha^\ell$ (the probability of having taken the last jump exactly ℓ timesteps in the past), depends solely on the damping factor and is independent of the structure of the graph.

2.2.2 Rank can change completely: a proof

Proof. [of Theorem 2.1] For simplicity we set the personalization vector $\mathbf{e} = [\frac{1}{n} \cdots \frac{1}{n}]$, but the result can be easily extended to the general case $\mathbf{e} > 0$. Let m be the smallest integer such that $\alpha < \frac{k}{m} < \alpha + \frac{1}{k}$ — one such $m \leq k^2$ always exists, since $\frac{k}{m} - \frac{k}{m+1} = \frac{k}{m(m+1)} < \frac{1}{k}$ for $\frac{k}{m} \leq 1 - \frac{1}{k}$. G is formed by 4 sets of nodes (see Figures 2.1 and 2.2): $V = \{v_1, \dots, v_k\}$ is the set of k nodes whose ranking is reversed by the variation in α; $W = \{w_1, \dots, w_k\}$ and $U = \{u_1, \dots, u_{k-1}, u'_1, \dots, u'_{k-1}\}$ are two sets of nodes that are parents of nodes in V; $T = \{t_1, \dots, t_{2k^2}\}$ is a set of nodes that are parents both of nodes in V and of nodes in W. Nodes of V are sinks; nodes of U and W have outdegree $\frac{k}{2}$ except for w_k that is a sink; nodes of T have outdegree k. Nodes of T and U have in-degree 0; nodes of W are all linked (only) by $t_1, \dots, t_m$ (and thus all have the same score); nodes of V are all linked by $t_{m+1}, \dots, t_{2k^2}$, and by one or more nodes of W and/or U (and thus always have a higher score than every other node). More specifically, v_i receives a link from each of $u_1, \dots, u_{k-i}$ and $u'_1, \dots, u'_{k-i}$ for $i \leq k/2$ and a link from each of $u_i, \dots, u_{k-1}$ and $u'_i, \dots, u'_{k-1}$ for $i > k/2$ (receiving $2(k-i)$ links from U); as well as a link from each of $w_{k+1-i}, \dots, w_{k-1}$ for $i \leq k/2$ (meaning u_1 receives no such link) and a link from each of $w_1, \dots, w_{i-1}$ for $i > k/2$ (receiving $i-1$ links from W). Then, for all $i < k$, v_i receives two more links from U and one less link from W than v_{i+1}, and $P_{\bar\alpha}(v_i) - P_{\bar\alpha}(v_{i+1}) \propto (\bar\alpha \frac{2}{k/2}) - (\bar\alpha \frac{1}{k/2} + \bar\alpha^2 \frac{m}{k} \frac{1}{k/2}) = \frac{\bar\alpha}{k/2}(1 - \bar\alpha \frac{m}{k})$. The last term is positive for $\bar\alpha = \alpha$ and negative for $\bar\alpha = \alpha + 1/k$, proving the thesis. □

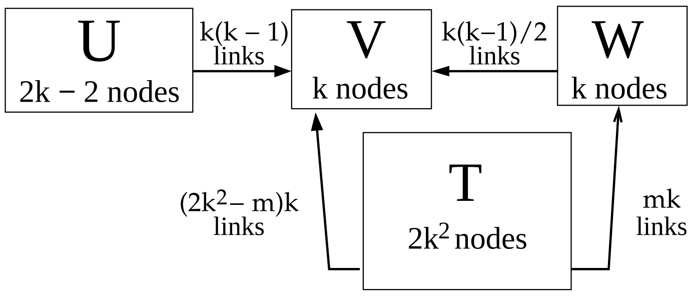

Figure 2.1: The graph G (Theorem 2.1) is formed by the 4 blocks of nodes U, V, W and T

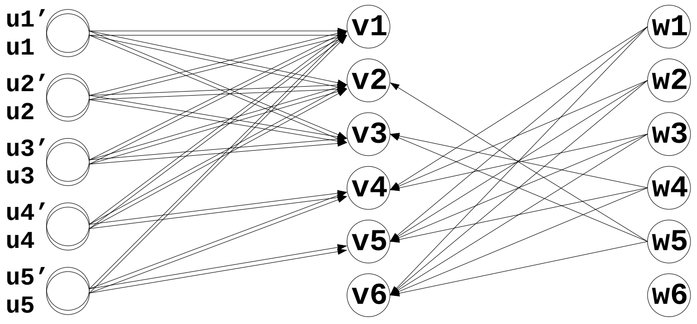

Figure 2.2: The links from U and W to V in the graph G (Theorem 2.1) for $k = 4$

Proof. [of Theorem 2.2]

We (arbitrarily) choose $F = |\Pi|$ distinct values $\alpha_1, \ldots, \alpha_F$ in $\mathcal{I}$, and, for each α_h, a k-uple of scores $s_{h1}, \ldots, s_{hk}$ for $v_1, \ldots, v_k$ that induces a distinct permutation in Π. For each node v_j we build a system $AX_j = B_j$, where

$$A = \begin{pmatrix} \alpha_1^1 & \ldots & \alpha_1^F \\ & \ldots & \\ \alpha_F^1 & \ldots & \alpha_F^F \end{pmatrix} \quad X_j = \begin{pmatrix} x_{1j} \\ \ldots \\ x_{Fj} \end{pmatrix} \quad B_j = \begin{pmatrix} s_{1j} \\ \ldots \\ s_{Fj} \end{pmatrix}$$

whose solutions $x_{1j}, \ldots, x_{Fj}$ are the branching contributions at levels $1, \ldots, F$ that the graph must provide to v_j in order to satisfy the score assignment (note that A is non-singular since it is the product of a Vandermonde matrix and a non-singular matrix). Although it might not be possible to construct a graph that provides these exact contributions, we show how to build one that maintains the same ranking of $v_1, \ldots, v_k$ for each value α_h of the damping factor. We use the solutions of the systems to iteratively build an *ancestor forest* of k trees, where at step h we add the nodes at depth h in each tree (step 0 adds the k roots $v_1, \ldots, v_k$).

We first assume that $x_{hj} > 0$ for all $j = 1, \ldots, k$, and build level h of the ancestor tree of v_j as follows. Let u be a node in level $h - 1$ of the ancestor tree of v_j, and

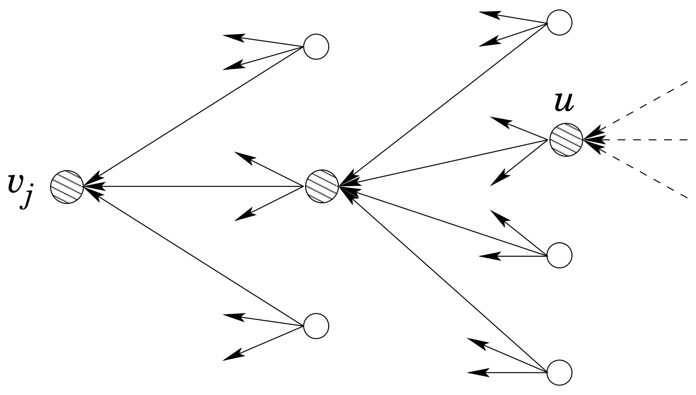

Figure 2.3: Ancestor tree for node v_j. Empty nodes are orphans.

let p be the unique (as will immediately be clear) path from u to v_j. Create n_{hj} new nodes and add them a link to u; then add $\omega_{hj} - 1$ children to each of these new nodes. Note that now there is a unique path from each of these nodes to v_j. Now the branching contribution from the h-th level is (n_{hj}/ω_{hj}) branching(p), and we can choose n_{hj}, ω_{hj} such that this contribution approximates x_{hj} to an arbitrary degree of precision. Finally set $\delta_h = 0$ (the reason will be clear soon).

If instead $x_{hj} \leq 0$ for some j, let $y_h = min_j\{x_{hj}\}$, and choose $\delta_h > 0$ such that $\delta_h + y_h > 0$. For $j = 1, \ldots, k$ replace x_{hj} with $x_{hj} + \delta_h$ (which is positive by construction) and proceed as above.

We now prove that PageRank ranks $v_1, \ldots, v_k$ in every order in Π as α takes values $\alpha_1, \ldots, \alpha_F$. For simplicity we set the personalization vector $\mathbf{e} = [\frac{1}{n} \cdots \frac{1}{n}]$, but the result can be easily extended to the general case $\mathbf{e} > 0$. Recalling Eq. 2.1, the PageRank score for v_j is

$$\begin{aligned}
P_{v_j}(\alpha_h) &= \frac{1-\alpha_h}{n} \sum_{\ell=0}^{\infty} \alpha_h^\ell \sum_{p \xrightarrow{\ell} v} \text{branching}(p) \\
&= \frac{1-\alpha_h}{n} \sum_{\ell=0}^{F} \alpha_h^\ell \cdot (x_{lj} + \delta_\ell) \\
&= \frac{1-\alpha_h}{n} \sum_{\ell=1}^{F} \alpha_h^\ell \cdot x_{lj} \; + \; \frac{1-\alpha_h}{n} \Big(1 + \sum_{\ell=0}^{F} \alpha_h^\ell \delta_\ell \Big) \\
&= \frac{1-\alpha_h}{n} \cdot s_{hj} \; + \; q_h
\end{aligned}$$

and thus, for a fixed h, the relative order of $v_1, \ldots, v_k$ follows the choice of the scores $s_{h1}, \ldots, s_{hk}$. To bring $v_1, \ldots, v_k$ to the top k positions, we increase their score by adding m parents to each v_j; i.e., we choose m equal to the maximum size (in nodes) of any ancestor tree. This increases the branching contribution at level 1 of each v_j by the same quantity, but does not affect the ordering; so $v_1, \ldots, v_k$ still assume all possible permutations in Π as α varies in $\alpha_1, \ldots, \alpha_F$.

Some readers might note that - at least in some application domains - one is unlikely to encounter graphs that are acyclic and/or disconnected and wonder about the applicability of Theorem 2.2 in such cases. It is easy to verify that the PageRank score vector does not change if one adds to every sink of a generic graph links to every other node in the graph, and that this modification makes the graph of Theorem 2.2 strongly connected. $\square$

2.3 Lineage analysis

It is natural to ask to what extent PageRank rankings of “real” graphs like the web graph are affected by variations in the damping factor, particularly in the light of the results of Section 2.2. Unfortunately, verifying that a node always outranks another as the damping factor varies by discrete increments (e.g. 0.01) *suggests, but can not conclusively prove* that the first node always outranks the second over a whole continuous interval of variation of the damping factor: by virtue of Theorems 2.1 and 2.2 the ranking could drastically change between those isolated sampling points. For the same reason, while previous experimental evidence obtained over a set of isolated values of the damping factor (e.g.[34]) *suggests* that PageRank is indeed stable even for large variations, this evidence can not be taken as conclusive. This section shows how to bypass this problem.

Just like Shannon’s sampling theorem proves that a continuous signal can be reconstructed from a finite number of samples so long as it is bandwidth-limited, Subsection 3.1 shows that, so long as PageRank can be approximated using only a finite number of iterations, one can exploit Sturm’s theorem to verify if a node outranks another for all damping factors in a finite number of steps. We also consider a more “robust” notion of rank dominance between nodes — not only for all damping factors, but also for all “damping variables” — informally, time-variant damping factors introduced in Subsection 2.3.2 and related to the *damping functions* of [5]. The core result of this section is a simple condition that can be checked efficiently and yet is both necessary and sufficient to guarantee that a node outranks another for all damping variables. Subsection 2.3.3 presents this “dominance” condition, which is based on the concept of *lineage* of a node, and turns out to be a very “natural” concept with a strong intuitive justification totally independent of PageRank.

2.3.1 Better rank for every damping factor?

In theory, one can check if the relative order of two nodes u, v changes as α varies by verifying if the difference between their PageRank series is both positive and negative in the interval $(0, 1)$. If one considers PageRank computations truncated after t iterations for some t (which is always true in practice), this can be done via Sturm’s theorem [26].

Let $\overline{x}$ be a generic root of a real polynomial $Q(x)$ with multiplicity m; then $\overline{x}$ is a root of $Q_1(x) = GCD(Q(x), \frac{\alpha}{dx}Q(X))$ with multiplicity $m - 1$. Sturm’s theorem [26] allows one to compute the number of real roots of $Q(x)$ of multiplicity ≥ 1 in the interval $(0, 1)$. Applying it to $Q_1(x) = GCD(Q(x), \frac{\alpha}{dx}Q(x))$, then to

$Q_2 = GCD(Q_1(x), \frac{\alpha}{dx}Q_1(x))$ and so on, one can compute the number of real roots of multiplicity respectively ≥ 2, ≥ 3 and so on, and thus the exact number of real roots of multiplicity $1, 2, 3$ and so on. It is then immediate to verify that the relative order of u and v changes as α varies over the interval $(0, 1)$ if and only if the difference between their truncated PageRank series $P^t_u(\alpha) - P^t_v(\alpha)$ has roots of odd multiplicity in that interval.

This method has a number of limitations. First, it does not allow an "incremental refinement" of the rank comparison: it is essentially impossible to extend the analysis obtained by truncating the PageRank series after t steps to include the effects of one more step without recomputing everything "from scratch". Second, its computational complexity is considerable, since Sturm's procedure for a single polynomial in general entails $\Omega(t)$ divisions between polynomials of degree $\approx t$. Third, it also requires considerable effort to avoid finite-precision errors, since it can involve a large number of subtractions and thus "catastrophic cancellations" [30, 43]. Fourth, it provides no natural interpretation of *why* a node outranks another for every damping factor. Fifth, the underlying model is not robust, as it does not capture the possibility of random surfers whose probability of "pressing the reset button" (one minus the damping factor) varies with time. The following subsection 3.2 explores this last issue in greater detail.

2.3.2 Damping variables

It is natural to extend the stochastic model of PageRank to one where the probability of following one of the current node's outgoing links is not a constant damping factor α, but is instead a *damping variable* $\alpha(\tau)$ that is a function of the number of steps τ taken since the last random jump. E.g. a web surfer might have a high probability of following a chain of links up to a depth of e.g. 3, and then of "hitting the reset button" with a random jump. In this case $\alpha(\tau)$ would be close to 1 for $\tau \leq 3$, but close to 0 for $\tau > 3$ (we formally define $\alpha(0) = 1$). We can generalize the analysis carried out in Subsection 2.2.1 to damping variables, and prove that:

Theorem 2.3. *If $\alpha(\tau) \leq (1 - \epsilon)$ for some $\epsilon > 0$ and for infinitely many τ, then the score assigned by PageRank to the node v is:*

$$P_v(\alpha(\cdot)) = \sum_{\ell=0}^{+\infty} J_{\alpha(\cdot)} \prod_{j=0}^{\ell} \alpha(j) \sum_{p \xrightarrow{\ell} v} \text{branching}(p) \cdot \mathbf{e}_{v_\ell} \tag{2.2}$$

where $J_{\alpha(\cdot)}$ is the limit for $t \to +\infty$ of the probability $J_{\alpha(\cdot)}(t)$ that a random jump occurs at time t, and $J_{\alpha(\cdot)} \prod_{j=0}^{i-1} \alpha(j)$ is the limit for $t \to +\infty$ of the *damping* *function* [5] denoting the probability of having taken the last random jump exactly i steps before time t.

While the meaning of Equation 2.2 and its relationship to Equation 2.1 are quite intuitive, proving that it holds requires some subtlety; indeed, if the condition $\alpha(\tau) \leq (1-\epsilon)$ is not satisfied (e.g. if one only requires $\alpha(\tau) < 1$) then the probability of being at node v at time t might not become stationary as $t \to \infty$. In this case, one might still *formally define* the score of a node through Equation 2.2, but relating it to a stochastic surfing model becomes considerably harder; and, perhaps more importantly, *computing* the score of a node becomes considerably more difficult, since the classic iterative algorithms based on the power method can fail to converge.

Note that PageRank in practice is computed using only a finite number $\bar{\tau}$ of iterations of the power method, effectively truncating the series in equations 2.1 and 2.2 to the τ^{th} term - i.e. employing, rather than a constant damping factor $\bar{\alpha}$, a damping variable $\alpha(\tau)$ equal to $\bar{\alpha}$ for $\tau \leq \bar{\tau}$ and to 0 for $\tau > \bar{\tau}$. This provides further justification for the introduction of damping variables.

The proof that, indeed, the stochastic process above does produce the stationary probability described by Theorem 2.3 may be skipped without impairing the understanding of the rest of the work.

Proof.[of Theorem 2.3] We first show that the probability $J_{\alpha(\cdot)}(\ell)$ that a random jump occurs at time ℓ converges to a limit. More formally, let $\epsilon > 0$ and let $\alpha : \mathbb{N} \mapsto [0,1]$ such that $\alpha(\tau) \leq 1-\epsilon$ for infinitely many τ. Then the following limit exists:

$$J_{\alpha(\cdot)} \triangleq \lim_{\ell \to +\infty} J_{\alpha(\cdot)}(\ell) \tag{2.3}$$

Let $L(\ell)$ be the time of the last jump before ℓ. By the law of total probability,

$$\begin{aligned} J_{\alpha(\cdot)}(\ell) &= \sum_{i=0}^{\ell-1} P(\text{ jump at } \ell | L(\ell) = i) P(\text{ jump at } i) \\ &= \sum_{i=0}^{\ell-1} \Big(1 - \alpha(\ell - i)\Big)\Big(\prod_{j=0}^{\ell-i-1} \alpha(j)\Big) J_{\alpha(\cdot)}(i) \\ &= \sum_{i=0}^{\ell-1} a_{\ell-i} J_{\alpha(\cdot)}(i) \end{aligned}$$

where $a_{\ell-i} = (1-\alpha(\ell-i))\left(\prod_{j=0}^{\ell-i-1} \alpha(j)\right)$. Rewrite the last summation as

$$\sum_{i=0}^{\ell-1} a_{\ell-i} J_{\alpha(\cdot)}(i) = \sum_{i=0}^{\ell-k-1} a_{\ell-i} J_{\alpha(\cdot)}(i) + \sum_{i=\ell-k}^{\ell-1} a_{\ell-i} J_{\alpha(\cdot)}(i)$$

We show that the first term vanishes and the whole summation tends to the second term, which converges. The first term satisfies

$$\sum_{i=0}^{\ell-k-1} a_{\ell-i} J_{\alpha(\cdot)}(i) \leq \sum_{i=0}^{\ell-k-1} a_{\ell-i} = \prod_{j=0}^{k} \alpha(j)$$

thus we choose a sufficiently large constant k_0 such that, for $\ell > k > k_0$, we have $\alpha(j) \leq 1-\epsilon$ for at least m values of j in $0, \dots, k$, where m is such that $(1-\epsilon)^m < \delta/2$; and then $\prod_{j=0}^{k} \alpha(j) < \delta/2$.

The second term converges for $\ell, k \to \infty$. In fact the last equation can be rewritten as $x(\ell) = \sum_{i=1}^{k} a_i x(\ell-i) + \delta'$, where $\delta' < \delta/2$. The characteristic polynomial is then $\sum_{i=1}^{k} a_i z^i - (1-\delta')$, and the sum of all terms a_i is $1 - \prod_{j=0}^{k} \alpha(j)$; then choose k such that the last quantity is strictly less than $1-\delta'$ (if $\alpha(j) = 0$ for $j < k$, $k_0 = j-1$ implies $\delta' = 0$). Now the roots lie outside the unit circle and the system converges to a limit $J_{\alpha(\cdot)}$.

We can now prove the Theorem. Choose $\delta > 0$. The probability of being on node v at time ℓ is

$$P_v^{\ell}(\alpha(\cdot)) = \sum_{i=0}^{\ell} \pi_v(i) J_{\alpha(\cdot)}(\ell - i)$$

where $\pi_v(i) = \prod_{j=0}^{i} \alpha(j) \sum_{p \xrightarrow{i} v} \text{branching}(p) \cdot \mathbf{e}_{p_i}$ is the probability of reaching v following i links after a random jump. We rewrite $P_v^{\ell}(\alpha(\cdot)) = \sum_{i=0}^{\lambda} \pi_v(i) J_{\alpha(\cdot)}(\ell-i) + \sum_{i=\lambda+1}^{\ell} \pi_v(i) J_{\alpha(\cdot)}(\ell-i)$ and bound the two terms. By equation 2.3, the i-th addend of the first term converges to $J_{\alpha(\cdot)} \pi_v(i)$ as $\ell \to +\infty$ for a fixed λ. Thus we can choose ℓ_0 such that, for every $\ell \geq \ell_0$,

$$\left| J_{\alpha(\cdot)} \sum_{i=0}^{\lambda} \pi_v(i) - \sum_{i=0}^{\lambda} \pi_v(i) J_{\alpha(\cdot)}(\ell-i) \right| \leq \sum_{i=0}^{\lambda} \left| J_{\alpha(\cdot)} - J_{\alpha(\cdot)}(\ell-i) \right| < \sum_{i=0}^{\lambda} \frac{\delta}{2(\lambda+1)} = \delta/2.$$

The second term is the probability of reaching v at time ℓ having taken the last random jump before time $\ell - \lambda$; therefore it is bounded by the probability of having taken a path of length λ, which is no more than $c(\lambda)$. We can now choose $\lambda \geq \lambda_0$

large enough to ensure that $c(\lambda) < (1-\epsilon)^m < \delta/4$.
The two bounds yield, for $\ell \geq \ell_0$ and $\lambda > \lambda_0$,

$$\begin{aligned}
&\left|J_{\alpha(\cdot)}\sum_{i=0}^{\ell}\pi_v(i) - P_v^{\ell}(\alpha(\cdot))\right| \\
\leq &\left|J_{\alpha(\cdot)}\sum_{i=0}^{\lambda}\pi_v(i) - \sum_{i=0}^{\lambda}\pi_v(i)J_{\alpha(\cdot)}(\ell-i)\right| + \left|J_{\alpha(\cdot)}\sum_{i=\lambda+1}^{\ell}\pi_v(i)\right| + \left|\sum_{i=\lambda+1}^{\ell}\pi_v(i)J_{\alpha(\cdot)}(\ell-i)\right| \\
\leq &\delta.
\end{aligned}$$

In conclusion, $\lim\limits_{\ell\to+\infty} P_v^{\ell}(\alpha(\cdot)) = J_{\alpha(\cdot)}\sum_{i=0}^{+\infty}\pi_v(i)$. □

For each node v, we now have a well defined score $P_v(\alpha(\cdot))$. The score vector has a representation which is very similar to that of the PageRank vector:

$$P(\alpha(\cdot)) \;=\; J_{\alpha(\cdot)}\sum_{\ell=0}^{+\infty} c_\ell \mathbf{M}^\ell \cdot \mathbf{e} \tag{2.4}$$

where $c_\ell = \prod_{j=0}^{\ell}\alpha(j)$. Note that for constant $\alpha(\cdot) = \alpha$ we have $J_{\alpha(\cdot)} = 1-\alpha$ and $c_\ell = \alpha^\ell$, which yields the classic PageRank formulation as a power series that can be found e.g. in [5].

2.3.3 Lineages

This subsection provides a simple condition both necessary and sufficient to guarantee that, for all damping variables $\alpha(\cdot)$, the score $P_v(\alpha(\cdot))$ of a node v is always at least as high as the score $P_w(\alpha(\cdot))$ of a node w.

Recall the term $\sum_{p\xrightarrow{\ell}v}\text{branching}(p)\cdot\mathbf{e}_{v_\ell}$ in Equation 2.2 - the level ℓ branching contribution to the score of v. Informally, the m^{th} generation of the *lineage* of v is equal to the sum of the branching contributions of all levels $\ell \leq m$. More precisely:

Definition 2.1.
The m^{th} generation of the lineage of v is $L_v(m) = \sum_{\ell=0}^{m}\sum_{p\xrightarrow{\ell}v}\text{branching}(p)\cdot\mathbf{e}_{v_\ell}$

We say that the lineage $L_v(\cdot)$ of a node v is greater than or equal to the lineage $L_w(\cdot)$ of a node w if it is greater or equal at every generation, in which case we write simply $L_v \geq L_w$.

There is a simple, intuitive interpretation of dominance between lineages. If we imagine that the authority/reputation/trust of a node is divided evenly among the nodes it points to, having a greater lineage at the m^{th} generation means receiving more authority from all nodes within at most m hops. Note that in such a comparison nodes that are further than m hops away are completely disregarded: this models the fact that, after all, authority/reputation/trust may be inherited, but only to a point. If one is uncertain to *which* point, one can be sure that a node has authority at least as high as that of another node only if its lineage is at least as high at *every generation*.

And, indeed, we show that having a lineage that is at least as high at every generation is strictly equivalent to receiving an equal or higher score by PageRank for every damping variable. More formally, we prove:

Theorem 2.4. $L_v \geq L_w \quad \Leftrightarrow \quad \forall\, \alpha(\cdot) \in (0,1),\; P_v\big(\alpha(\cdot)\big) \geq P_w\big(\alpha(\cdot)\big)$

Proof. We first prove the $\Rightarrow$ side. Recall $c_i = \prod_{j=0}^{i} \alpha(j)$. The score vector is a convex linear combination of lineages:

$$P_v(\alpha(\cdot)) \;=\; J_{\alpha(\cdot)} \textstyle\sum_{i=0}^{+\infty} b_i L_v(i)$$

where $b_i = c_i - c_{i+1} \geq 0$. Then $P_v(\alpha(\cdot)) \geq P_w(\alpha(\cdot))$.

We now prove the $\Leftarrow$ side. For every integer $k \geq 0$, consider

$$\alpha(i) = \begin{cases} 1-\epsilon & i = 0,\ldots,k \\ \epsilon & i > k \end{cases}$$

where $\epsilon \in (0,1)$ will be defined later. Let $P^{(k)}(\alpha(\cdot)) = J_{\alpha(\cdot)} \sum_{j=0}^{k} c_j \mathbf{M}^j \cdot \mathbf{e}$ – i.e. to the truncation of the PageRank series to the term of order k. Then we have:

$$\begin{aligned} P_v(\alpha(\cdot)) \;&=\; J_{\alpha(\cdot)} \sum_{i=0}^{k} c_i (\mathbf{M}^i \cdot \mathbf{e})_v + J_{\alpha(\cdot)} \sum_{i=k+1}^{\infty} c_i (\mathbf{M}^i \cdot \mathbf{e})_v \\ &\leq\; P_v^{(k)}(\alpha(\cdot)) + J_{\alpha(\cdot)} \sum_{i=k+1}^{\infty} (1-\epsilon)^k \epsilon^{i-k}\; n \\ &\leq\; P_v^{(k)}(\alpha(\cdot)) + J_{\alpha(\cdot)}\; \epsilon\; (1-\epsilon)^{k-1}\; n \end{aligned}$$

Considering nodes v and w we have:

$$P_v^{(k)}(\alpha(\cdot)) \;+\; J_{\alpha(\cdot)}\; \epsilon\; (1-\epsilon)^{k-1}\; n \;\geq\; P_v(\alpha(\cdot)) \;\geq\; P_w(\alpha(\cdot)) \;\geq\; P_w^{(k)}(\alpha(\cdot))$$

which can be rewritten as

$$\begin{aligned}&(1-\epsilon)^k L_v(k) + \epsilon \sum_{i=0}^{k-1} (1-\epsilon)^i L_v(i) + \epsilon J_{\alpha(\cdot)} (1-\epsilon)^{k-1} n \\ &\geq (1-\epsilon)^k L_w(k) + \epsilon \sum_{i=0}^{k-1} (1-\epsilon)^i L_w(i)\end{aligned}$$

that by hypothesis holds for arbitrarily small $\epsilon > 0$, implying $L_v(k) \geq L_w(k)$. □

Note that in practice we do not need to check lineage dominance at *every generation*; if we restrict ourselves to damping variables that are 0 for $\tau > t$, we only need to check at most t generations. This is equivalent to considering PageRank computations truncated after at most t iterations (effectively truncating the series in Equations 2.2 and 2.4 to the t^{th} term), which is always true in practice. Thus, one can check if the relative order of two nodes is the same for every damping variable in $O(t)$ comparisons; by contrast, the algorithm based on Sturm's theorem described in Subsection 2.3.1 requires $\Omega(t^2)$ operations.

In addition, comparing lineages yields computations that have considerably fewer problems of numerical stability than those involving Sturm chains. While an in-depth exploration of this issue is beyond the scope of our work (see e.g. [44]), it is immediate to verify that computing the ℓ^{th} lineage involves summations where each term is a) positive and b) obtained as the inverse of a product of a number of integers equal to the lineage. The products can be computed with a loss of accuracy equal to at most $\lceil \log_2(\ell) \rceil$ bits; the sum entails an additional loss of at most 1 bit of accuracy (see e.g. [30, 43]).

Thus, on any graph of size n and outdegree at most g on which PageRank converges in at most ℓ iterations, a floating point arithmetic with exponents of $max(\log_2 \log_2(\ell n), \log_2 \log_2(g^\ell))$ bits and significands greater than $\approx \log_2(1/\epsilon) + \log_2(\ell)$ bits suffices to discriminate between lineages within a factor $1+\epsilon$ of each other. In particular, IEEE 754 double precision arithmetic appears more than sufficient to guarantee several dozen bits of precision on the web Graph and on any "lesser" graph.

2.4 Damping-independent ranking

Lineages and Theorem 2.4 provide powerful tools to compare two nodes over the spectrum of all damping variables. This section leverages them to introduce the concepts of *strong rank* and *weak rank* (Subsection 2.4.1), and the related ranking algorithms *StrongRank* and *WeakRank* (Subsection 2.4.2). These provide interesting measures of the "fuzziness" of rankings, of the "orderability" of the nodes of a graph, and of the performance of different ranking algorithms based on the random surfer model.

2.4.1 Strong and weak rank

Given a node v, and assuming for simplicity all ties in lineage score are broken (e.g. arbitrarily), all other nodes fall into three sets: the set $S(v)$ of nodes *stronger* than v (with a greater lineage), the set $W(v)$ of nodes *weaker* than v (with a lesser lineage), and the set $I(v)$ of nodes *incomparable* with v (with a lineage greater at some generations and lesser at others). The cardinalities of these sets define the *weak* and *strong* rank of v:

Definition 2.2. *The* weak rank $\rho_w(v)$ *and the* strong rank $\rho_s(v)$ *of a node* v *are, respectively,* $|S(v)|+1$ *and* $|S(v) \cup I(v)|+1$.

Note that $\rho_w(v)-1$ is the number of those nodes that outperform v for all damping variables, whereas $\rho_s(v)-1$ is the number of those nodes which outperform v for at least one damping variable. Thus $\rho_w(v)$ is a lower bound to the minimum (i.e. best) rank achievable by v, while $\rho_s(v)$ is an upper bound to the maximum (i.e. worse) rank achievable by v; and $\rho_s(v)-\rho_w(v)$ is then an upper bound to the maximum variation in v's rank. Note that any or all of these three bounds might hold strictly, and there is a subtle but profound difference between $\rho_s(v)-\rho_w(v)$ and the maximum variation of v's rank that makes the former more descriptive of the "fuzziness" of v's performance. E.g. suppose that v holds 10^{th} rank for all damping variables, but 999 other nodes fill the top 9 positions in turn for different damping variables. In this case $\rho_s(v) = 1000$, since v does not fare definitively better than any of those 999 nodes; and $\rho_w(v) = 1$, since none of those 999 nodes fares definitively better than v.

Strong and weak rank also provide a measure of the *global* rank fuzziness in a generic graph - that is, of the extent to which the graph can be ordered satisfying simultaneously every type of user (with different user behaviours described by different damping variables). For a graph G, consider the number $s_k(G)$ of nodes with strong rank k or less. If $s_k(G) \approx k$, then every user's top k set has a high density of nodes also in the top k set of *every other* user. Conversely, if $s_k(G) \ll k$, then few nodes will be of "universal importance". Similarly, consider the number $w_k(G)$ of nodes with weak rank k or less. If $w_k(G) \approx k$, then relatively few nodes are sufficient to include the k most important nodes of *every other* user. Conversely, if $w_k(G) \gg k$, then the the behaviours of different users are sufficiently diverse that, in order to ensure that no user misses the items of the greatest importance to him, any algorithm must return a very large collection of items.

The ratio $w_k(G)/s_k(G)$ can then be seen as a measure of the *inevitable* price of obliviousness to the user's model: the smaller it is, the more well-orderable G is. In the ideal case, $s_k(G) = k = w_k(G)$, all users have exactly the same preferences and every damping variable yields the same ordering.

2.4.2 StrongRank and WeakRank

Strong rank and weak rank automatically induce two new ranking algorithms, *StrongRank* and *WeakRank*, that rank nodes respectively in order of strong and weak rank. Neither necessarily corresponds to PageRank for some damping variable. StrongRank tends to return items that are each of at least moderate importance for every user (with different user behaviours described by different damping variables). WeakRank tends to return, for every user, at least a few items that are of high importance for that user. The evaluation of the effectiveness of StrongRank and WeakRank as practical ranking algorithms (either on the web or in other application domains) is certainly a promising direction of future research.

It is immediately obvious, however, that StrongRank and WeakRank can provide benchmarks to classify the performance of other other ranking algorithms based on the surfer model. If for every k many of the top k items returned by a ranking algorithm are also among the top k items returned by StrongRank (the "intersection metric" of [22]), one can reasonably deduce that a large fraction of items returned by that algorithm is of at least moderate importance for every user. Similarly, a large intersection with WeakRank points to an algorithm that returns, for every user, a large fraction of that user's top choices.

A complete analysis of the complexity of StrongRank and WeakRank would require taking into account the properties of the target graph (and thus of the application domain) as well as caching and parallelizability issues. This is beyond the scope of our work; however, the remainder of this section provides a few basic results (that can be skipped without impairing the understanding of the rest of the chapter).

It is not difficult to prove that the *worst case* complexity of PageRank (for *one* value of the damping factor) on a graph of n nodes equals that of StrongRank and

WeakRank:

Theorem 2.5. *The worst case complexity of computing StrongRank and WeakRank up to lineage ℓ on an n node graph is $O(\ell n^2)$, equal to that of computing the first ℓ iterations of PageRank.*

StrongRank and WeakRank can then be computed for any graph of up to millions of nodes on a PC in a few days; and very few application domains of PageRank entail larger graphs (the World Wide Web being one notable exception). Graphs of much larger size n are manageable by PageRank itself only if of low (average) degree $g \ll n$; and in such graphs, one is often interested only in the top k ranking nodes, with $k \ll n$. When these graphs are $\Delta-$*well orderable*, i.e. there are at most $k\Delta$ nodes with WeakRank less than k and at least k/Δ with StrongRank less than k (this seems to hold with $2 \leq \Delta \leq 4$ for $k > 100$ in social networks like the web, see Section 2.5) we can refine Theorem 2.5 into Theorem 2.6:

Theorem 2.6. *The worst case complexity of computing the top k ranks of StrongRank and WeakRank up to lineage ℓ on a $\Delta-$well orderable graph of average degree g and n nodes is, respectively, $O(n\ell g(1 + \frac{\log(k)}{g} + \frac{\Delta(k+\ell)(\log(\Delta)+k)}{ng}))$ and $O(n\ell g(1 + \frac{\log(k)}{g} + \frac{k^2\Delta^3+\Delta(k+\ell)(\log(\Delta)+k)}{ng}))$ vs. a complexity of $O(n\ell g)$ for the top k ranks and the first ℓ iterations of PageRank.*

For $max(\log(n), g, \ell) \leq k \leq \sqrt{n}$ the complexity of StrongRank and WeakRank becomes respectively $O(n\ell g(1 + \frac{\log(k)+\Delta}{g}))$ and $O(n\ell g(1 + \frac{\log(k)+\Delta^3}{g}))$. Thus, even in the case of the (indexed) web graph it still appears possible to compute the top $\approx 10^5$ ranks of StrongRank and WeakRank in time comparable to that of PageRank (within an order of magnitude - several hours on a single PC).

The rest of this section is devoted to the proof of Theorem 2.6, of which Theorem 2.5 is corollary.

Proof. We first describe two initial "preprocessing" steps that are common to StrongRank and WeakRank, before detailing how each algorithm computes its top k ranked nodes; finally, we analyse the worst case complexity for the top k ranks and the first ℓ iterations of PageRank.

The first preprocessing step computes the lineages, up to level ℓ, of all the n nodes of the graph. This can be done in time $O(ngl)$ by iteratively computing the lineage at each level i, which takes an average $O(g)$ (the average (in)degree of the graph) for each of the n nodes. The second preprocessing step extracts, for each of the ℓ levels, the top k nodes (as ranked by lineage) in non-decreasing order. This takes $O(n\log(k))$ steps for each level using a max-heap of size k. The cost of performing these two steps adds up to $O(lng + ln\log(k))$.

To find the top k nodes ranked by StrongRank, it is sufficient to consider all the nodes with strong rank less than $k\Delta$, which by definition of Δ are at least k. These nodes are necessarily in the intersection of the sets of top $k\Delta$ nodes (ranked by lineage) at each level. Since Δ is unknown in advance, one can use a binary search over the size of these sets until an intersection of size between k and $k+\ell$ is found; this takes $O(\log(k\Delta))$ steps, each computing the intersection between ℓ sets of size $O((k+\ell)\Delta)$ each; and the cost sums up to $O(\log(k\Delta)\ell(k+\ell)\Delta)$. For each of the (less than) $k+\ell$ nodes in the intersection found, one computes its strong rank as the number of nodes which are ranked higher (by lineage) in at least one level. This takes $O(kl\Delta)$ steps for each node, for a total of $O((k+\ell)kl\Delta)$ steps. Thus the cost of StrongRank, including the two preprocessing steps, is $O(lng+ln\log(k)+\log(k\Delta)\ell(k+\ell)\Delta+(k+\ell)lk\Delta)$. When k is reasonable (i.e. $\log(k)=O(g)$) and when, as in the case of the web graph, Δ is a small constant and $\ell=O(k)$ (i.e. PageRank converges in a few thousand iterations) then, in the formula above, a) the second term is $O(lng)$; b) the third term is $O(\ell k\Delta g)=O(lng)$ (since $k\Delta\leq n$); c) the fourth term is $O(\ell k^2\Delta)=O(\ell n\Delta)$. This yields a total cost of $O(lng+ln\Delta)$ steps.

To find the top k nodes ranked by WeakRank, it is sufficient to consider all nodes with a weak rank less than k, which are at least k and, by definition of Δ, no more than $k\Delta$. These nodes are necessarily in the union of the sets of top $k\Delta$ nodes (as ranked by lineage) at each level; i.e. the union of the same sets considered in the computation of StrongRank above, yielding the same cost. This union has cardinality at most $k\Delta^2$ — otherwise there would be more than $(k\Delta)\Delta$ nodes with weak rank less than $k\Delta$, which is impossible by definition of Δ. For each of these $O(k\Delta^2)$ nodes, its weak rank is the number of nodes ranked higher (by lineage) at each level. This (using lineage rankings) takes no more than $kl\Delta$ steps for each node, and $O(k^2\ell\Delta^3)$ total steps. Thus the cost of WeakRank, including the two preprocessing steps, is $O(lng+ln\log(k)+\log(k\Delta)\ell(k+\ell)\Delta+k^2\ell\Delta^3)$; under the same assumptions made above, this becomes $O(lng+ln\Delta^3)$.

Note that the cost of computing the top k nodes ranked by PageRank using the first ℓ iterations is equal to the cost of the first preprocessing step, $O(lng)$ (as it is the most efficient method), plus the cost of extracting the top k nodes, which takes $O(n\log(k))$ using a max-heap. This yields a total cost of $O(lng)$ steps. □

In the web graph Δ appears to be bounded by a small constant not exceeding 2.5 (see Subsection 2.5.2 below). Thus we could run StrongRank and WeakRank for $k=20000$ even for a fairly large snapshot of the .it domain in a few hours on a personal computer.

2.5 Experimental results

This section brings the analytical machinery of Section 2.3 to bear on “real” graphs - a 2004 snapshot of the .it domain web graph, and a 2007 snapshot of the CiteSeer citation graph. Subsection 2.5.1 briefly presents the data and the experimental setup. Subsection 2.5.2 evaluates the extent to which the nodes of those graphs can be ordered in a fashion satisfying simultaneously every user (with different user behaviours described by different damping variables). Finally, Subsection 2.5.3 analyses the performance of PageRank as α varies from nearly 0 to nearly 1 by evaluating the intersection of its top k node set with the top k node set of StrongRank and WeakRank.

2.5.1 Data and experimental setup

We analyse the $40M$ node, $1G$ link snapshot of the 2004 .it domain published by [42], and the $0.7M$ node, $1.7M$ link snapshot of the late 2007 CiteSeer citation graph [1] (nodes are articles and links are citations). We use the WebGraph package [42],[11] for their manipulation.

.it, as a national first level domain, provides a large and (unlike e.g. .com or .gov) “well-rounded” portion of the web graph with a size still within reach of our storage and computational resources. Furthermore, the primary language of .it is not shared by any other country (unlike e.g. .uk or .fr), minimizing distortions in ranking introduced by the inevitable cut of links with the rest of the web. We perform two pre-processing steps on the .it snapshot. First, we remove intra-domain links (which constitute strongly biased conferral of authority, see [29] and [27]). Second, we merge all dynamic pages generated from the same base page, dealing with pages (like the homepage of a forum’s dynamic language) automatically linked by a huge collection of other template-generated pages. Both steps appear to markedly improve the human-perceived quality of the results.

Then, we compute the lineage of each node up to the 128^{th} generation for both graphs. Note that, in theory, lineages should be compared for *all* generations. Stopping at the 128^{th} generation is equivalent to considering damping variables $\alpha(\tau)$ that are 0 for $\tau > 128$, and (typical) PageRank implementations that compute the score vector using at most 128 iterations and disregarding authority propagation over paths longer than 128 hops. This appears reasonable because for all damping factors/variables except those extremely close to 1, the branching contribution of levels beyond the 128^{th} is dwarfed by rounding errors of finite precision computing machinery; and because, empirically, the set of the top k items returned by

StrongRank and WeakRank when considering only the first ℓ generations appears to almost completely stabilize as ℓ grows larger than $60-100$ (Fig. 2.4 and 2.5 show the normalized intersection between the top k item sets returned by StrongRank and WeakRank when considering 128 lineages, and the top k item sets returned when considering ℓ lineages for $\ell = 1, \ldots, 128$).

2.5.2 Choose the damping, choose the ranking?

Figures 2.6 and 2.7 show the number of $k-$strongly ranked nodes and the number of $k-$weakly ranked nodes for the .it domain graph and the CiteSeer citation graph as k varies from 1 to 16000. The two graphs exhibit a strikingly similar behaviour, with a few differences.

Both in the .it and in the CiteSeer graph the ratio w_k/k between the number w_k of $k-$weakly ranked nodes and k is never higher than 6, and converges relatively quickly to a value between 1.5 and 2.5 (it is never larger than 4 for $k \geq 7$ on the .it graph, and for $k \geq 127$ on the CiteSeer Graph). This ratio represents the *inevitable cost of obliviousness to the user model*: the set of all those items that are outranked by less than k other items for some damping variable has size at least w_k.

The ratio s_k/k between the number s_k of $k-$strongly ranked nodes and k is considerably smaller than 1 in both graphs - particularly in the CiteSeer graph. This ratio represents the fraction of items that are *robustly* in the top k, for all damping factors and variables. For the CiteSeer graph, the ratio is $\approx$ 0.05 for $k < 200$, converging to a value between 0.5 and 0.6 for $k > 10000$. For the .it graph, it is slightly larger: between 0.1 and 0.2 for $k < 200$, converging to a value between 0.4 and 0.5 for $k > 10000$.

Thus, ranking sensitivity to the damping factor in “real” graphs appears not nearly as high as that of the synthetic graphs of Section 2.2, but still considerable - particularly for the top $10-100$ items. To return all items that would appear among the top k for *some* damping factor or variable, even the “best” ranking algorithm might have to return from 2 to 4 times as many items; and although *any* choice of the damping factor will guarantee among the top k a non-negligible core of items that would also be returned among the top k for every other choice, this core appears relatively small, between 5% and 40% of the total.

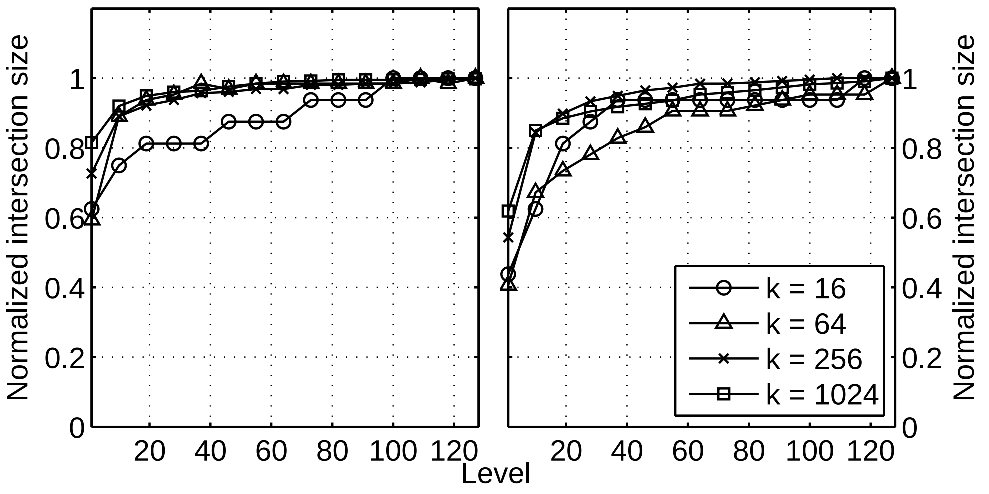

Figure 2.4: .it web graph: convergence of top k-strong (left) and top k-weak (right) sets as a function of lineage generation, for different values of k

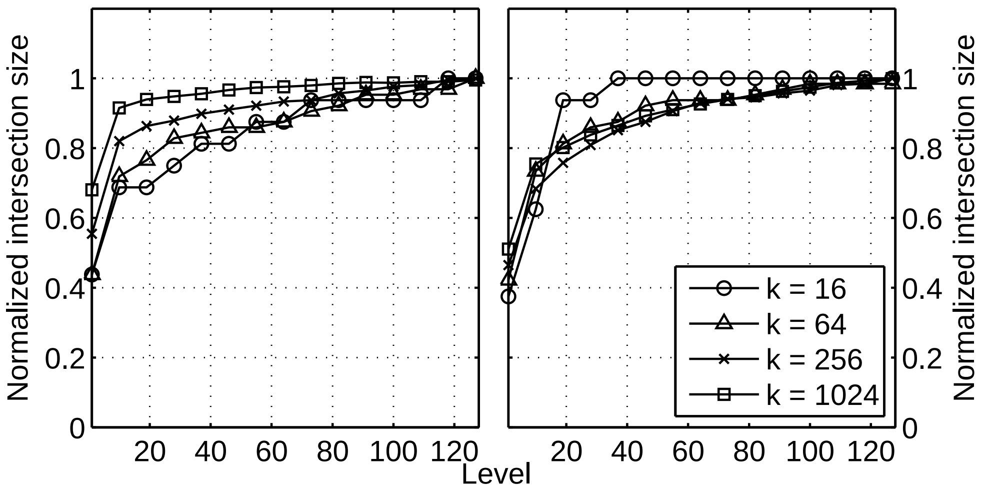

Figure 2.5: CiteSeer citation graph: convergence of top k-strong (left) and top k-weak (right) sets as a function of lineage generation, for different values of k

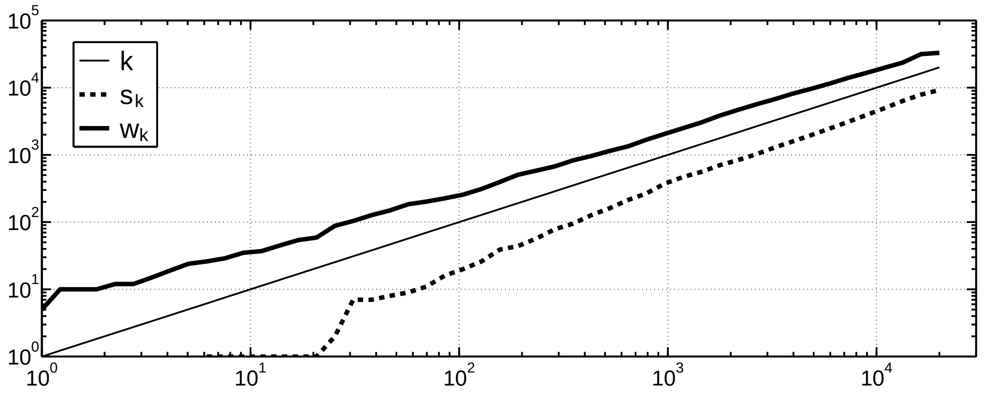

Figure 2.6: .it web graph: s_k (number of k-strongly ranked nodes) and w_k (number of k-weakly ranked nodes), as a function of k

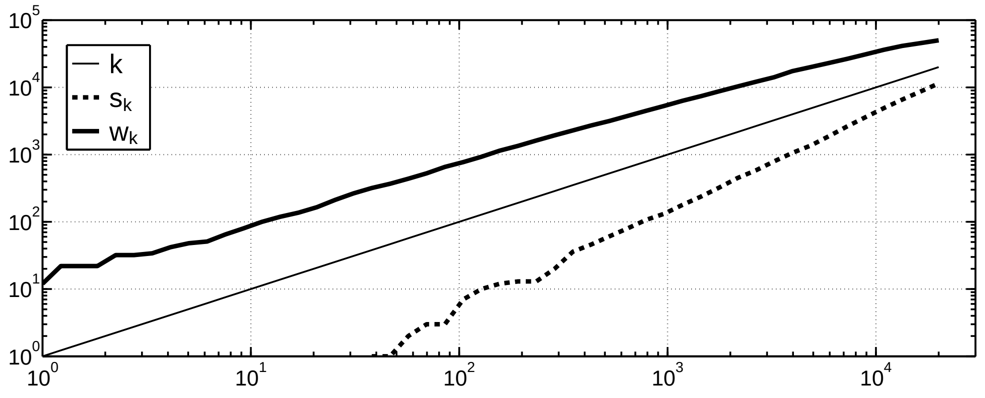

Figure 2.7: CiteSeer citation graph: s_k (number of k-strongly ranked nodes) and w_k (number of k-weakly ranked nodes), as a function of k

2.5.3 The best damping factor

In this subsection we deploy StrongRank and WeakRank as a testbed to evaluate the performance of PageRank for different damping factors (see Subsection 2.3.3) on the .it graph and on the CiteSeer graph. Figures 2.8,2.9,2.10 and 2.11 show the fraction of the top k items returned by PageRank that are also among the top k items returned respectively by StrongRank and WeakRank, for $k = 16$, 64, 256 and 1024, as the damping factor varies between 0.01 and 0.99 in steps of 0.01.

Again, the behaviour of PageRank is strikingly similar on the two graphs. For both of them and for all four values of k, the size of the intersection set is between $0.38k$ and $0.93k$ for all damping factors sampled (but note that, in theory, anything could happen between those discrete sampling points - see Section 2.2!). Thus, for all damping factors, including ones very close to 0 that make the computation of the score vector particularly efficient [10], PageRank always returns a set of results in common with both StrongRank and WeakRank of reasonable size.

For both graphs, the largest intersection with WeakRank is achieved for damping factors in the $0.8 - 0.9$ range; whereas the largest intersection with StrongRank is achieved for damping factors in the $0.5 - 0.6$ range. Which is more desirable? Ultimately, it depends on the nature of the application. For example, a web search engine typically returns to a user many more "leads" than that user will follow. In this context a false negative (not returning a "good" page) is far more damaging than a false positive (returning a "mediocre" page). Thus, a large intersection with WeakRank is more desirable, being indicative of an algorithm that returns, for every user, a large fraction of that user's top choices. It is then perhaps not surprising that the typical value assigned to the damping factor in search engines is indeed 0.85! On the other hand, in a trust/reputation system a false positive (returning as highly trusted an item the user would not trust) is far more damaging than a false negative (not returning as trusted an item the user would actually trust). A large intersection with StrongRank is then more desirable, being indicative of an algorithm that returns only items that are at least moderately trusted by *every* user model; and a damping factor closer to 0.5 - as suggested in [4] - might be a better choice.

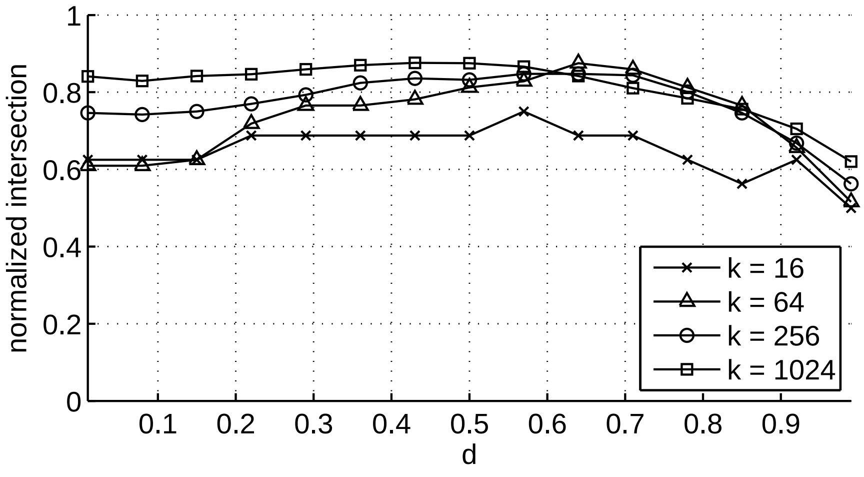

Figure 2.8: .it graph: PageRank/StrongRank intersection metric as α varies

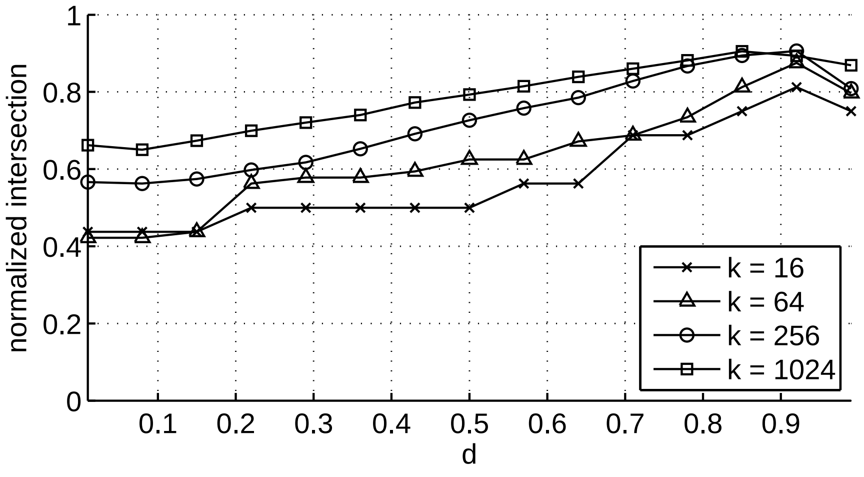

Figure 2.9: .it graph: PageRank/WeakRank intersection metric as α varies

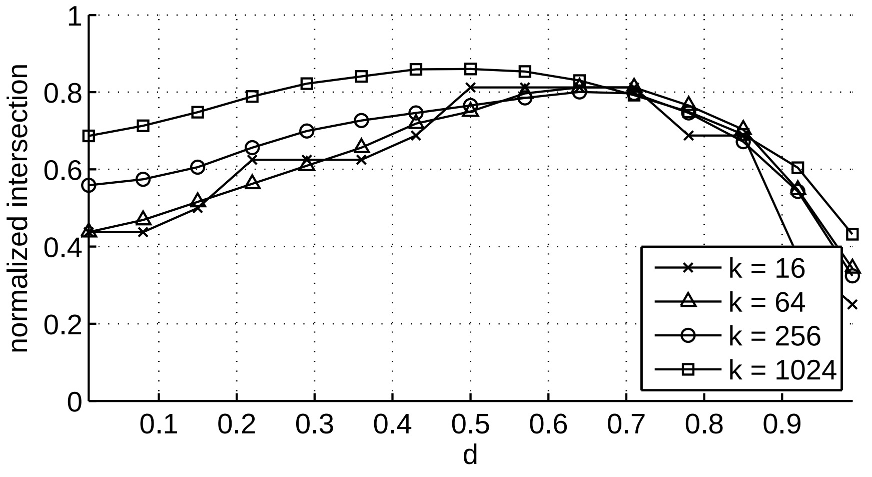

Figure 2.10: CiteSeer: PageRank/StrongRank intersection metric as α varies

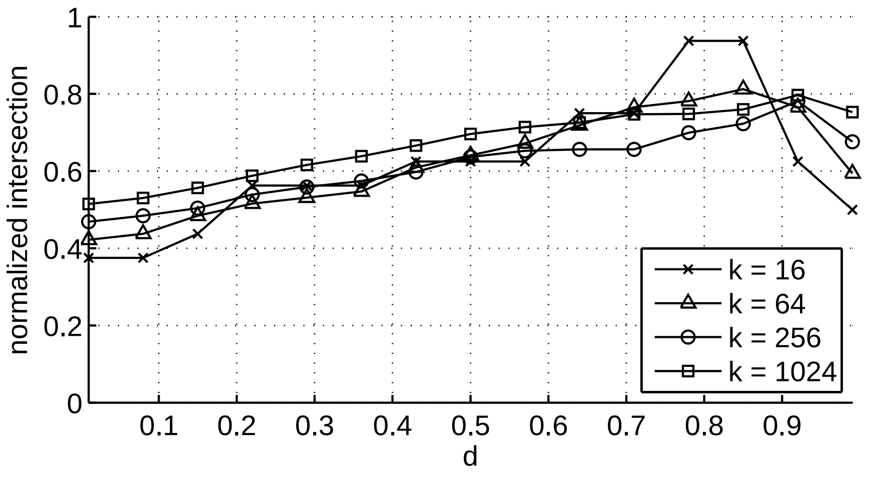

Figure 2.11: CiteSeer: PageRank/WeakRank intersection metric as α varies

2.6 The damping makes the ranking – in theory if not in practice

This chapter addresses the fundamental question of how variations in PageRank's damping factor can affect the ranking of nodes. We showed that, at least on some graphs, *arbitrarily small* variations in the damping factor can make the top ranked nodes assume all possible orderings, which is deeply unsatisfying should one be aiming at an "objective" ranking. Previous experiments suggested this was not the case at least for the web graph (leaving open the issue of other application domains), but verifying rank stability on a finite set of damping factors is not sufficient to conclude that rank is stable as the damping factor varies over a whole continuous interval. We solved this problem developing *lineage analysis*, that allows to compare the rank of two nodes "in one shot" for *all* (even time-variant) damping factors using just a finite set of lineage measurements. Lineage analysis also provides a simple, very "natural" interpretation of rank dominance for all damping factors that is completely independent of PageRank; and it induces the notions of *strong rank* and *weak rank* of a node, related to, but subtly different from, those of best and worse rank - and more descriptive, since they capture the level of churn "around" a node whose rank is relatively stable overall yet highly unstable compared to its individual competitors. Experimental results show that, in real graphs, the rank is relatively stable but not completely insensitive to variations in the damping factor.

From weak and strong rank we derive two new ranking algorithms, StrongRank and WeakRank, which provide useful benchmarks to compare different link analysis algorithms in terms of their ability to, respectively, return "universally useful" results, and to ferret out results of high importance to niches of users. A more thorough evaluation of StrongRank and WeakRank is certainly a promising direction for future research. StrongRank and WeakRank can also be used to evaluate different values of the damping factor — validating the "folk lore" result that 0.85 is ideal, at least for search engines and other applications where it is far more important to avoid false negatives then false positives. For applications where the reverse is true, our results suggest that a value closer to 0.5 might be more effective, reflecting the results of [4]. It is surprising that, in this regard, the .it and CiteSeer graphs exhibit exactly the same characteristics - it would be interesting to investigate if (and perhaps why) this is the case in other application domains of PageRank.

Chapter 3

Local Computation of PageRank : The Ranking Side

3.1 Introduction

This work tackles the problem of computing locally the ranking induced by PageRank on k target nodes of an input graph, studying if the correct ranking can be obtained at a low cost, i.e. by visiting only small subgraphs around those target nodes.

Section 3.2 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 without visiting a number of nodes linear in the size of the input graph, or may even never stabilize.

Section 3.3 formally defines the problem of local ranking, taking into account the case of nodes whose PageRank scores are significantly separated, and Section 3.4 gives the main theoretical results, proving that local 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 3.5 describes two experiments suggesting that the cost of local ranking in real graphs is actually high and strongly dependent on the input graph.

Section 3.6 summarizes our results.

3.2 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 ranking algorithms (see Section 3.5), the brute force is still a reference algorithm for at least two reasons. First, it is the basis of virtually every known local score approximation algorithm (such as those proposed in [6] and [16]). These in turn are natural bases for local ranking algorithms. Second, the brute force algorithm naturally derives from the expression of PageRank given by Equation (1.5) that turns out to be very useful in the analysis of those algorithms. Thus it is important to understand why the brute force algorithm 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 3.1) or oscillate indefinitely between two "opposite" results (Theorem 3.2).

Recall from Section 1.2 that the PageRank row vector $\mathbf{P} = [P(v_1), P(v_2), \ldots, P(v_n)]$ is defined as the limit probability vector of a Markov chain, and can be obtained either as the unique probability vector which is the solution of the linear system $\mathbf{P} = \mathbf{PT}$ 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 avenues can be followed, since the entire graph structure is not available; in this case, a natural way of computing the PageRank score $P(v)$ of a single node v is given by Equation 1.5:

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

where the influence $\mathrm{inf}_{\tau}(z, v)$ of z on v gives the probability of ending in v by following a chain of τ arcs from z. Indeed, 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 τ all the ancestors of layer τ and cumulating their influence on v weighted by $\frac{1-\alpha}{n}\alpha^{\tau}$. After iteration ℓ, the resulting *brute force score* at layer ℓ is

$$P^{(\ell)}(v) = \frac{1-\alpha}{n} \sum_{\tau=0}^{\ell} \alpha^{\tau} \sum_{z \in G} \mathrm{inf}_{\tau}(z, v) \tag{3.1}$$

If graph G has no dangling nodes, $P^{(\ell)}(v)$ converges from below to the PageRank score of v given by Equation (1.5). 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 ℓ has reached a sufficiently large threshold ℓ_0. Unfortunately, such threshold is unknown a priori, and this may lead to a premature halt and an incorrect ranking. Indeed we prove that, for any given ℓ_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 ℓ_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 3.1. *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 u to v in G' is the unique concatenation of a path from u to v in G and $t \geq 0$ self-loops (v, v). Conversely, every path $p : u \to v$ in G generates, for $t = 0, \ldots, \infty$, the paths from u to v in G' that are the concatenation of $p : u \to v$ and t self-loops (v, v) that damp the contribution by a factor α^t. Thus the expression of $B'(v)$ given by Equation (3.1) can be rewritten as:

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

□

By Lemma 3.1, one can compute the brute force score of a node whose only outgoing arc forms a self-loop by excluding it from each layer and multiplying the result by a constant factor. We can now state Theorem 3.1.

Theorem 3.1. *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 $v_1, \ldots, v_k$ is:*

$$\begin{aligned} &v_1 < \ldots < v_k \quad \textit{for } \ell \leq \ell_0 \\ &v_k < \ldots < v_1 \quad \textit{for } \ell > \ell_0 \end{aligned}$$

Proof. We exhibit a graph G that satisfies the statement. G consists of k subgraphs $G_1, \ldots, G_k$. Subgraph G_i (Figure 3.1) 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 ℓ_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).

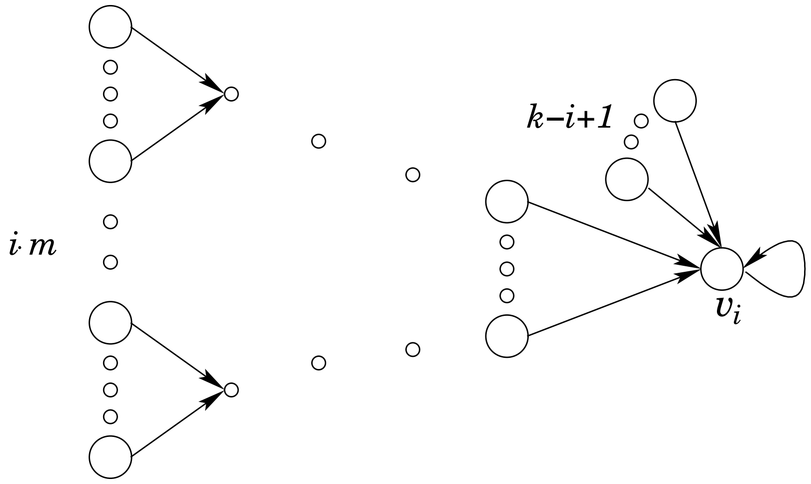

Figure 3.1: Subgraph G_i (Theorem 3.1) containing node v_i and all its ancestors.

Node v_i has one less parent but m more ancestors at layer $\ell_0 + 1$ than node v_{i-1} and thus, intuitively, receives a lower contribution from the first layers but a higher contribution from the last ones. We now exhibit values of m and d that satisfy the statement of the theorem. Consider v_i and v_j for $1 \le i < j \le k$. For $\ell \le \ell_0$ node v_i receives, from layers 1 to ℓ, the contribution of $j - i$ parents more than v_j through paths that possibly include many self-loops on the target node:

$$P^{(\ell)}(v_i) = P^{(\ell)}(v_j) + \frac{1-\alpha}{n}(j-i)\sum_{\tau=1}^{\ell} \alpha^{\tau} > P^{(\ell)}(v_j)$$

and thus v_i is ranked higher than v_j. For $\ell = \ell_0 + 1$, node v_i receives from layer $\ell_0 + 1$ the contribution of $(j-i)m$ ancestors less than v_j and the expression of $P^{(\ell)}(v_i)$ becomes:

$$P^{(\ell)}(v_j) + \frac{\alpha}{n}(j-i)(1-\alpha^{\ell_0+1}) - \frac{1-\alpha}{n}(j-i)m\alpha^{\ell_0+1}$$

which, for $m > \frac{1-\alpha^{\ell_0+1}}{\alpha^{\ell_0}(1-\alpha)}$, is smaller than $P^{(\ell)}(v_j)$ and thus v_i is ranked lower than v_j. It is easy to see that this ranking also holds for any $\ell \ge \ell_0 + 1$ and that it is the correct ranking since the brute force scores converge to the PageRank scores.

To make $v_1, \ldots, v_k$ the top k nodes in the graph, balance the im arcs incoming from layer $\ell_0 + 1$ so that the number of parents of any ancestor at layer ℓ_0 is at most $\lceil \frac{im}{d^{\ell_0}} \rceil$ which, for a sufficiently large d, is no more than d. Note that $v_1, \ldots, v_k$ 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. $\square$

By Theorem 3.1, for any $\ell_0 > 0$ there exist graphs where, to compute the correct relative ranking of the top k nodes, not only the brute force algorithm 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 3.1 the brute force algorithm incurs a very high cost (exponential in ℓ 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 3.2. *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 $v_1, \ldots, v_k$ satisfy, for any $i \in [1, \frac{k}{2}]$, any $j \in [\frac{k}{2} + 1, k]$, and any $\ell \geq 0$:*

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

Proof. We exhibit a graph G that satisfies the statement. G consists of $k/2$ identical subgraphs $G_1, \ldots, G_{k/2}$. Subgraph G_i (Figure 3.2) contains nodes v_i and $v_{i+k/2}$, the set of nodes W_i (described below) and 2 nodes that are the sole children of v_i and the sole parents of $v_{i+k/2}$.

Let $w = \sum_{x \in W_i} \frac{1}{\text{outdegree}(x)}$. According to Equation (3.1), the scores of v_i and $v_{i+k/2}$ are respectively:

$$P(v_i) = \frac{1-\alpha}{n} \left(1 + \alpha(1 + w) + 2\alpha^2 + \alpha^3 + \alpha^4(1 + w) + \ldots\right)$$
$$P(v_{i+k/2}) = \frac{1-\alpha}{n} \left(1 + 2\alpha + \alpha^2 + \alpha^3(1 + w) + \alpha^4 + \ldots\right)$$

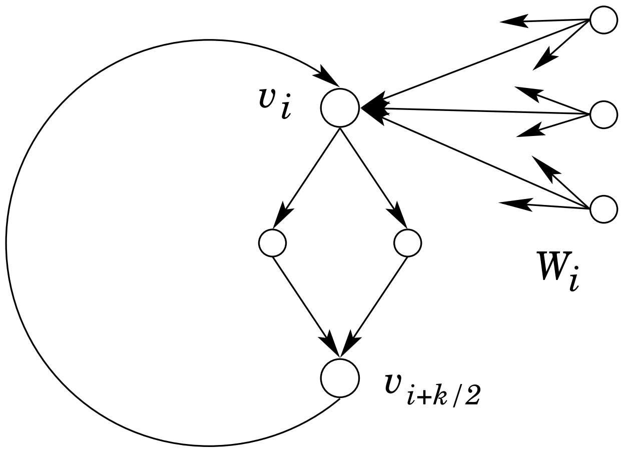

Figure 3.2: Subgraph G_i (Theorem 3.2) has a cycle that causes a perpetual oscillation in the relative ranking given by the brute force algorithm.

and their difference $D(v_{i+k/2}, v_i) = P(v_{i+k/2}) - P(v_i)$ is:

$$\begin{aligned} D(v_{i+k/2}, v_i) &= \frac{1-\alpha}{n}\left(\alpha(1-w) - \alpha^2 + \alpha^3 w\right)\sum_{t\geq 0}\alpha^{3t} \\ &= \frac{1-\alpha}{n}\alpha\left((1-\alpha) - w(1-\alpha^2)\right)\frac{1}{1-\alpha^3} \end{aligned}$$

For v_i and $v_{i+k/2}$ to oscillate indefinitely in rank as ℓ grows, their brute force scores must converge to the same value or, equivalently, $D(v_{i+k/2}, v_i) = 0$, which is true for $w = \frac{1}{1+\alpha}$. Since α is rational by hypothesis, w is also a rational p/q for some integers p and q and thus let W_i contain p nodes with q children each (v_i being one of them).

We now prove that the relative rankings oscillate. Note that for any $1 \leq i \leq k/2$ and any $k/2+1 \leq j \leq k$ we have $D^{(\ell)}(v_j, v_i) = D^{(\ell)}(v_{i+k/2}, v_i)$. Substituting $w = \frac{1}{1+\alpha}$ in the expression of $D^{(\ell)}(v_{i+k/2}, v_i)$ and taking $\tau = \lfloor \ell/3 \rfloor$, we have for the three cases $\ell \equiv 0, 1, 2 \mod 3$:

$$\begin{aligned} D^{(\ell)}(v_j, v_i) &= D^{(3\tau)}(v_j, v_i) = D^{(3\tau)}(v_{i+k/2}, v_i) \\ &= \frac{1-\alpha}{n}\left(\alpha\frac{\alpha}{1+\alpha} - \alpha^2 + \alpha^3\frac{1}{1+\alpha}\right)\sum_{t=0}^{\tau-1}\alpha^{3t} \\ &= \frac{1-\alpha}{n}\left(\frac{\alpha^2 - \alpha^2 - \alpha^3 + \alpha^3}{1+\alpha}\right)\sum_{t=0}^{\tau-1}\alpha^{3t} \\ &= 0 \end{aligned}$$

$$\begin{aligned} D^{(\ell)}(v_j, v_i) &= D^{(3\tau+1)}(v_j, v_i) \\ &= D^{(3\tau)}(v_j, v_i) + \frac{1-\alpha}{n}\alpha^{3\tau+1}\frac{\alpha}{1+\alpha} \\ &= \frac{1-\alpha}{n}\frac{\alpha^{3\tau+2}}{1+\alpha} \\ &> 0 \end{aligned}$$

$$\begin{aligned} D^{(\ell)}(v_j, v_i) &= D^{(3\tau+2)}(v_j, v_i) \\ &= D^{(3\tau+1)}(v_j, v_i) - \frac{1-\alpha}{n}\alpha^{3\tau+2} \\ &= \frac{1-\alpha}{n}\left(\frac{\alpha^{3\tau+2}}{1+\alpha} - \alpha^{3\tau+2}\right) \\ &< 0 \end{aligned}$$

To prove that nodes $v_1, \ldots, v_k$ are the top k ranked *at any iteration* by the brute force algorithm, note that the score of each child z of v_i,

$$P(z) = \frac{1-\alpha}{n}\left(1 + \frac{\alpha}{2} + \frac{\alpha^2}{2}(1+w) + \alpha^3 + \frac{\alpha^4}{2} + \ldots\right)$$

is term-by-term not larger (and iteration-by-iteration smaller) than the scores of both v_i and $v_{i+k/2}$. Finally, nodes of set W_i are orphans and thus ranked lower than both v_i and $v_{i+k/2}$. □

By Theorem 3.2, 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 α, p and q, the size of the graph in Figure 3.2 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 3.1 and 3.2 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 is not breadth-first: for example, visiting highly-contributing ancestors earlier. Second, they could compute the approximate PageRank scores more accurately: for example, taking into account the cycles in the graph to compute "in one shot" the total contribution of an ancestor appearing in an infinite number of layers, thus eliminating potential rank oscillations. 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.

Thus, it is necessary to spend some more effort on a more rigorous definition of the “local ranking” problem; which we shall do in the next Section.

3.3 Local ranking of nodes

This section formally restates the local ranking problem, that was informally introduced in Section 3.1, taking into account the observations raised at the end of Section 3.2.

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 algorithm A to correctly rank u and v.

In this work, the local ranking problem is considered in a PageRank context. Formally, given a graph G, a damping factor α, and k *target* nodes $v_1, \ldots, v_k \in G$, one must 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 distance is not less than a given ϵ. Formally, given a graph G, a damping factor α, and k target nodes $v_1, \ldots, v_k \in G$, the *local ϵ-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 to this context with obvious modifications.

It is thus natural to ask how high the cost of local ϵ-ranking is – even considering only the top nodes of a graph. The next section proves some theoretical lower bounds on this cost, for both randomized and deterministic algorithms.

3.4 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 *ϵ-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 [6]), to ϵ-approximate the score of a node for a reasonable constant ϵ, 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 *ϵ-ranking* of two or more target nodes, as described in Section 3.3. We prove that this is indeed the case: the following Theorem 3.3 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 3.4 takes the bound to $\Omega(n)$ for deterministic algorithms.

Theorem 3.3. *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 $v_0, \ldots, v_{k-1}$ are ϵ-separated and, to compute their relative ranking, A performs an expected $\Omega\left(\alpha\sqrt{\frac{n}{\epsilon}}\right)$ 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 $v_0, \ldots, v_{k-1}$ are ϵ-separated and, to compute their relative ranking, A performs $\Omega\left(\alpha\sqrt{\frac{n}{\epsilon}}\right)$ 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 [45] 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 3.2. *The expected worst-case cost of any Las Vegas randomized local algorithm for* 1*-OR is at least $\frac{m}{2}$.*

Proof.[of Lemma 3.2] Any (Las Vegas) algorithm performs 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. □

Proof.[of Theorem 3.3] 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 3.2 gives a lower bound on (the expected value of) S_B, and thus on (the expected value of) S_A.

Let $\mathbf{x_1}, \ldots, \mathbf{x_{k-1}}$ be $k-1$ m-bit instances of 1-OR and b a positive integer (proper values of m and b will be computed below). We describe a link server that lets A run on a graph G consisting of k disjoint subgraphs $G_0, G_1, \ldots, G_{k-1}$. For $i = 1, \ldots, k-1$, subgraph G_i (Figure 3.3) contains the target node v_i and its self-loop, $m+1$ nodes $u_i^0, \ldots, u_i^m$ that have v_i as their sole child, ib nodes $w_i^0, \ldots, w_i^{ib-1}$ that have u_i^0 as their sole child and, for $j = 1, \ldots, m$, kb nodes $w_i^{jkb}, \ldots, w_i^{(j+1)kb-1}$ 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 $u_0^0, \ldots, u_0^m$ that have v_0 as their sole child and kb nodes $w_0^0, \ldots, w_0^{kb-1}$ whose sole child is u_0^0.

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 the bit equals 1 then it returns v_i as the sole child and $w_i^{jkb}, \ldots, w_i^{(j+1)kb-1}$ 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 the bit equals 1 then it returns $u_i^{j'}$ as the sole child and no parents, else it returns w_i^j 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

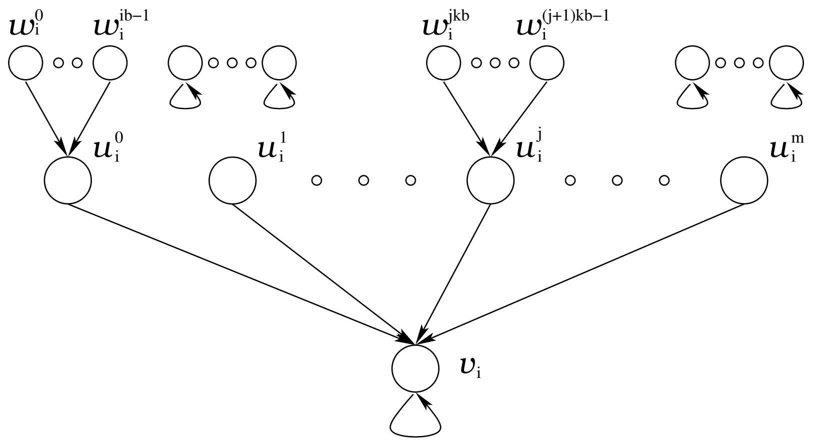

Figure 3.3: Subgraph G_i (Theorem 3.3). Node u_i^0 always has exactly ib parents; node u_i^j has kb parents if and only if $\mathbf{x_i}(j) = 1$, else it has an associated group of kb "disconnected parents".

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 $v_0, \ldots, v_{k-1}$ then B solves correctly the $k-1$ instances of 1-OR.

Since G has no dangling nodes, by Lemma 3.1 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 (3.1) these scores become:

$$P(v_0) = \tfrac{1}{n}\left(1 + \alpha(m+1) + \alpha^2 kb\right)$$
$$P(v_i) = \tfrac{1}{n}\left(1 + \alpha(m+1) + \alpha^2(i + k\|\mathbf{x_i}\|)b\right) \qquad i \geq 1$$

By hypothesis, algorithm A ranks $v_0, v_1, \ldots, v_{k-1}$ and, in particular, computes the relative ranking of v_0 and v_i for $i = 1, \ldots, k-1$. 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 $v_0, \ldots, v_{k-1}$, 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

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

It is immediate to verify that the number of nodes n in the graph is in $\Theta(mbk^2)$ which, substituting the above values for m and b, is in $\Theta(n_0)$. The hypothesis $n_0 \geq 6k^3$ guarantees also that $b \geq 1$ and $m \geq kb \geq 2$; this implies that $v_0, v_1, \ldots, v_{k-1}$ 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 ϵ-separated since one of them has the same number of parents but at least b grandparents more than the other, and thus the difference ΔP between their scores satisfy $\Delta P \geq \frac{\alpha^2 b}{n}$ which, 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:

$$\begin{aligned} \frac{\Delta P}{P} &\geq \frac{\alpha^2 b}{n} / \frac{1}{n}\left(1 + \alpha(m+1) + \alpha^2(k-1+k)b\right) \\ &\geq \frac{\alpha^2 b}{2(m+kb+1)} \geq \frac{\alpha^2 b}{5m} \end{aligned}$$

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 ϵ-separated.

We now compute the cost of locally ranking the target nodes $v_1, \ldots, v_k$. 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 3.2 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 [15] 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. □

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 3.4. *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 $v_0, \ldots, v_{k-1}$ are ϵ-separated and, to compute their relative ranking, A performs $\Omega(n)$ queries.*

Proof. The proof is based on an adversarial argument. 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 the graph of Theorem 3.3 and consisting of k disjoint subgraphs $G_0, G_1, \ldots, G_{k-1}$. Let m and b be two positive integers (their values will be computed below). For $i = 1, \ldots, k-1$, subgraph G_i (Figure 3.4) contains the target node v_i and its self-loop, $m+1$ nodes $u_i^0, \ldots, u_i^m$ that have v_i as their sole child, ib nodes $w_i^0, \ldots, w_i^{ib-1}$ that have u_i^0 as their sole child and kb nodes $w_i^{ib}, \ldots, w_i^{(i+k)b-1}$ that may either have self-loops or have u_i^j as their sole child for some $1 \leq j \leq m$. Subgraph G_0 contains the reference target node v_0 and its self-loop, $m+1$ nodes $u_0^0, \ldots, u_0^m$ that have v_0 as their sole child, and kb nodes $w_0^0, \ldots, w_0^{kb-1}$ that have u_0^0 as their sole child.

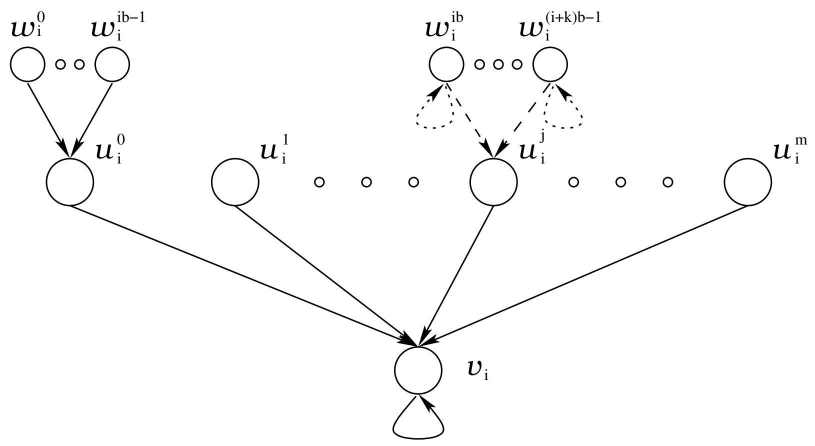

Figure 3.4: Subgraph G_i (Theorem 3.4). Node u_i^0 always has ib parents, while nodes $w_i^{ib}, \ldots, w_i^{(i+k)b-1}$ may or may not have u_i^j as their sole child.

The link server exploits the fact that the mapping of node IDs (i.e. the content of queries) to nodes in the graph can be chosen arbitrarily and can force a deterministic algorithm to query almost all the nodes before collecting enough information to compute a correct ranking. Indeed this is what the link server does, following this rule:

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

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

Since the number of nodes in graph G is $n \in \Theta(mk + k^2 b)$, these choices for m and b imply $n \in \Theta(n_0)$. The hypothesis $n_0 \geq k^2$ guarantees also that $m \geq kb$ and $b \geq 1$; this implies that $v_0, \dots, v_{k-1}$ are the top k nodes in the graph since at any level they have more ancestors than any other node.

It remains to prove that the target nodes are ϵ-separated. Since the number of parents and grandparents of v_i is the same as in the graph of Theorem 3.3, 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

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

which proves that the target nodes are ϵ-separated.

Note that the link server can invalidate *any* ranking of the target nodes that A should output *before* querying at least mk nodes. To prove this, suppose that A has performed less than mk queries and let v_i be a target node such that one of its parents u_i^j has not been queried (as stated before, one such node must exist). If A ranks v_i higher than v_0, the link server disconnects $w_i^{ib}, \dots, w_i^{(i+k)b-1}$ from u_i^j; otherwise, it connects $w_i^{ib}, \dots, w_i^{(i+k)b-1}$ to u_i^j. The expression (see proof of Theorem 3.3) of the score of v_i shows that in either case A gives an incorrect ranking. Therefore A must incur a cost of at least mk which, plugging in the above value for m and recalling that $n_0 \in \Theta(n)$, becomes:

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

which concludes the proof. □

It is easy to see that, as the sizes of the graphs of Theorems 3.3 and 3.4 increase, the multiplicative constants effectively bring the upper bound on the choice of ϵ within a small constant factor of $\frac{\alpha^2}{k}$, which takes the lower bounds on the incurred costs within a small constant factor of, respectively, $\Omega(\sqrt{kn})$ and $\Omega(n)$. Therefore, by Theorems 3.3 and 3.4 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 ϵ-separated for a constant $\epsilon \approx 0.1$, which in absolute terms can be *orders of magnitude* greater than the average PageRank score $\frac{1}{n}$, yet any "useful" (i.e. with a reasonable confidence level) local ϵ-ranking algorithm incurs $\approx$ 100k queries in the randomized case and $\approx$ 1B queries in the deterministic case.

It is now clear that, at least *in theory*, there exist pathological cases that make the local ϵ-ranking problem intractable. In practice, real graphs may behave differently, and good algorithms may be able to compute the correct relative ranking of the target nodes at a reasonable cost. The next section gives experimental results to shed light on this point.

3.5 Experiments

This section describes two experiments, both making use of known score approximation algorithms ([6, 16]), to estimate how far the theoretical lower bounds of Section 3.3 are in practice from the cost of local ranking on real graphs. Subsection 3.5.1 introduces the experimental setting, while Subsections 3.5.2 and 3.5.3 detail the two experiments and the results obtained.

3.5.1 Experimental design

We ran the experiments on two publicly available real graphs crawled by the Laboratory for Web Algorithmics at the University of Milan (graphs crawled with UbiCrawler [9] and compressed with WebGraph [11]). 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 [19], 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 whole web does not imply suppressing 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 reputes 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, it is a completely isolated graph and thus 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 [33]. 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. Surprisingly, the literature on local score approximation always neglects this important issue. We conjecture that guessing this influence requires statistical considerations similar to those used in [18] to estimate the influence of the surrounding (unknown) graph on the nodes of a (known) subgraph; however, this problem is beyond the scope of the present work. These reasons led us to eliminate all the dangling nodes by repeated pruning. For both graphs, this removed less than 15% of the nodes and less than 6% of the arcs.

For the sake of simplicity, in both experiments we restricted the problem to the ranking of *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 PageRank 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 3.3. 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 α, and from each graph we selected the top 10k nodes – about the top 1% in each graph. 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 ϵ 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.

3.5.2 Is ranking local in real graphs?

The goal of the first experiment is to estimate a lower bound on the cost of locally ranking ϵ-separated nodes as a function of ϵ. 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, a concept formalized in the following definition:

Definition 3.3. *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 a minimal 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 ϵ-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 ϵ-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, which is similar if not identical to the behaviour of algorithms proposed in literature. We conjecture that *any* local algorithm that behaves differently would fail for some input G,u,v : intuitively, if the algorithm does not know *anything* about the unexplored part of the graph, and yet it estimates the score of the target node exceeding the sum of the contributions of the visited nodes, then the rest of the graph could be adversarially built to make the PageRank score of the target node lower than the estimated score – and make the algorithm fail. Thus, we consider only algorithms that approximate the PageRank score as the sum of the contributions of the visited nodes.

Under these assumptions, 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 $\tilde{P}(v) \leq P(v)$, therefore inferring 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. Therefore the cardinality of $S_{PER}(G,u,v)$ is a lower bound to the cost of locally ranking u and v. Note that, since the contribution of an ancestor z is α 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. This implies that nodes of $S_{PER}(G,u,v)$ form a connected graph and are in principle *reachable* by those algorithms that query only the parents of already-queried nodes – i.e. virtually any "reasonable" algorithm.

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 an approximated minimal set.

Figure 3.5 illustrates the average size of the estimated minimal sets as a function of ϵ for both graphs. As expected, this size increases as ϵ decreases – ranking poorly-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, but still 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, ϵ-ranking in the LiveJournal graph is strongly non-local except for extremely separated nodes.

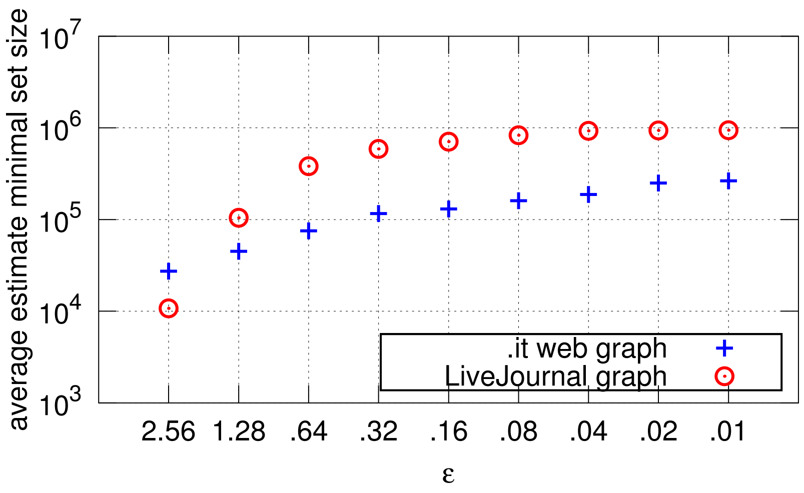

Figure 3.5: Average size of estimated minimal sets for $(\epsilon, 2\epsilon)$-separated nodes as a function of ϵ for the `.it` web graph and the LiveJournal graph

3.5.3 Local ranking with real algorithms

The second experiment evaluates the performance of two real (i.e. returning *wrong* results in some cases) algorithms, giving an 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 of the two target nodes layer by layer, computing their approximate scores according to Equation (3.1) and inferring their relative ranking. Figures 3.6 - 3.7 (`.it` graph) and 3.9 - 3.10 (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. Figures 3.8 (`.it` web graph) and 3.11 (LiveJournal graph) show, for different values of $(\epsilon, 2\epsilon)$, the competitive ratio of BF, i.e. its average cost over the average size of the estimated minimal sets, as a function of the number of layers visited. As one might expect from a naive brute force-based ranking algorithm, BF incurs overwhelmingly high costs (several million queries) in both graphs, even to guarantee a precision ≥ 0.9 for only the most separated nodes.

In the `.it` web graph, the average cost incurred by BF saturates to slightly more than ten millions, or $\approx 0.25n$, and its estimated competitive ratio (the ratio of the average cost to the average size of estimated minimal sets) to achieve a precision ≥ 0.9 is ≈ 10 for the highest separations and range from 20 to more than 100 for all the others – thus, at least in principle, there is abundant space for performance

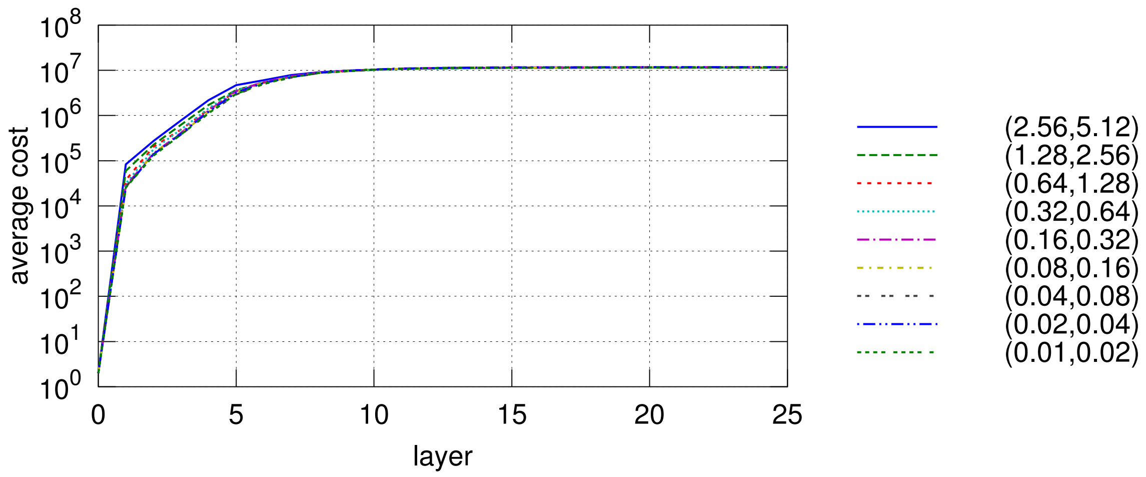

Figure 3.6: `.it` web graph, BF algorithm : average cost vs. layers visited, for different values of $(\epsilon, 2\epsilon)$

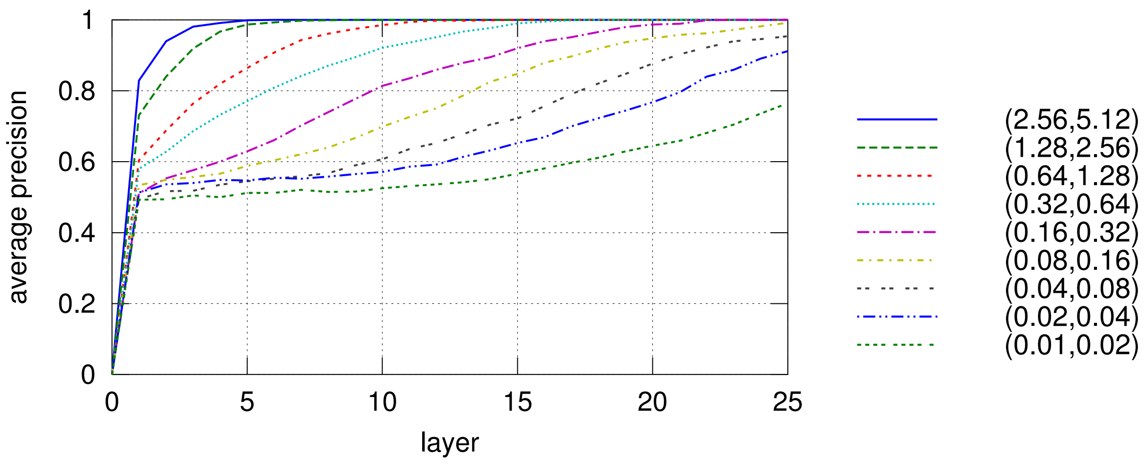

Figure 3.7: `.it` web graph, BF algorithm : average precision vs. layers visited, for different values of $(\epsilon, 2\epsilon)$

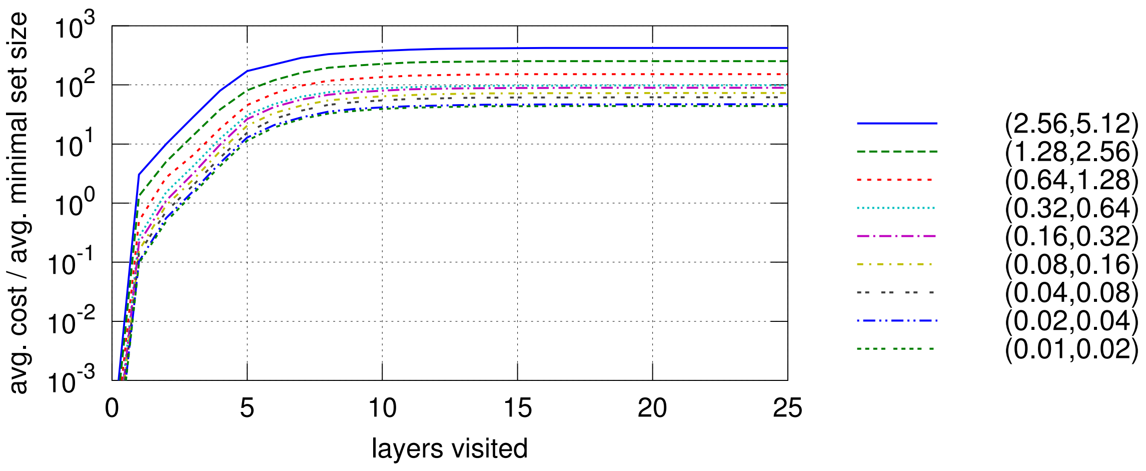

Figure 3.8: `.it` graph, BF algorithm : ratio of average cost to average size of estimated minimal sets vs. layers visited, for different values of $(\epsilon, 2\epsilon)$

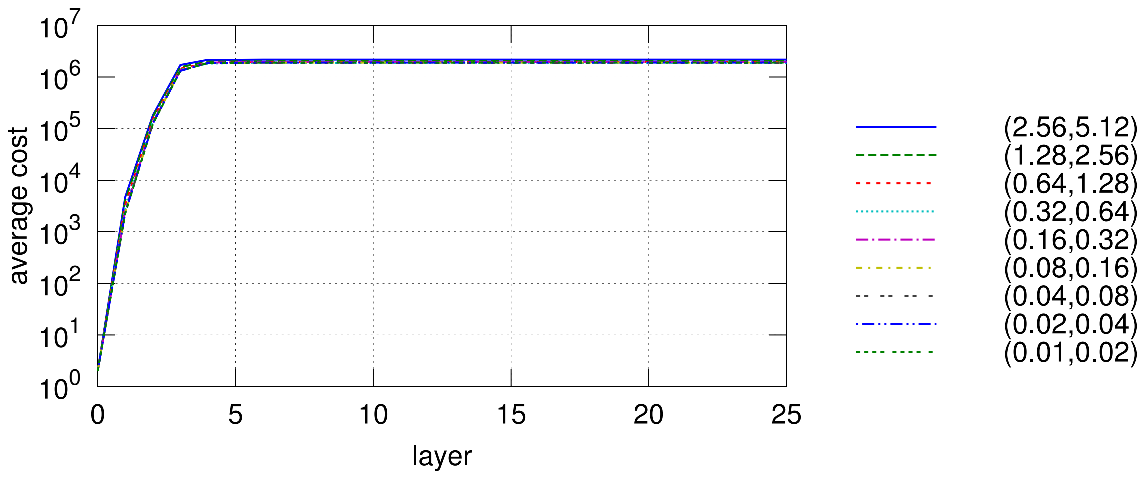

Figure 3.9: LiveJournal graph, BF algorithm : average cost vs. layers visited, for different values of $(\epsilon, 2\epsilon)$

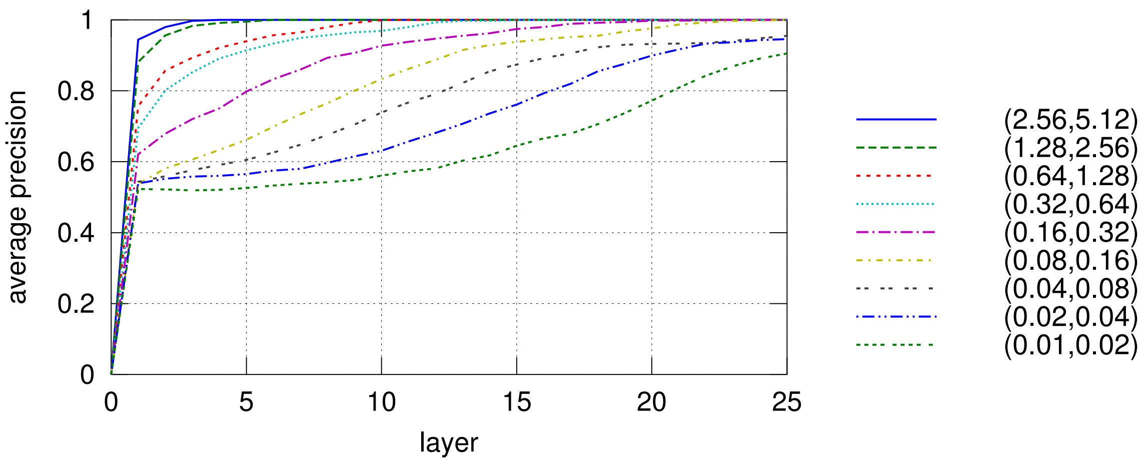

Figure 3.10: LiveJournal graph, BF algorithm : average precision vs. layers visited, for different values of $(\epsilon, 2\epsilon)$

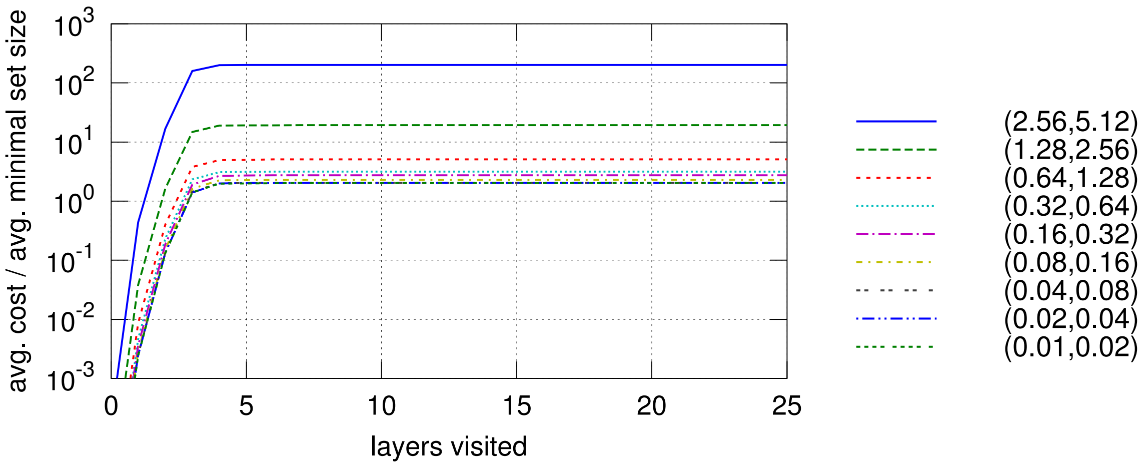

Figure 3.11: LiveJournal graph, BF algorithm : ratio of average cost to average size of estimated minimal sets vs. layers visited, for different values of $(\epsilon, 2\epsilon)$

improvements in this case. In the LiveJournal graph, the average cost of BF saturated to about two millions, or $\approx 0.4n$, while its estimated competitive ratio to achieve a precision ≥ 0.9 ranges from 2 to 5 for every separation except the two highest, dropping to 0.4 for $\epsilon = 2.56$. This points not to the (unlikely) efficiency of BF, but to the "hardness" of graph as an instance for the local ranking problem.

Observe that neither the average in-degree of the graphs (> 30 for `.it` vs < 16.5 for LiveJournal) nor the sizes of their largest strongly connected component ($\approx 0.72n$ for `.it` vs. $\approx 0.78n$ for LiveJournal , see [42]) do easily predict the number of nodes visited by the BF algorithm or the size of the estimated minimal sets illustrated in Figure 3.5.

We then tested an improved version (ImPBF) of the pruned brute force algorithm [6] (PBF). This is an intuitively efficient variation of the BF algorithm that, at each layer $\tau \geq 1$, does not visit all the parents of layer $\tau - 1$ but only those whose estimated contribution (i.e. over paths that are known at iteration τ) to the score of the target node is above a given threshold. The conjecture underpinning this heuristics is that the nodes with a small contribution over paths of length $\leq \tau$ probably give a small overall contribution, and their parents probably do the same. Counterexamples show that sometimes this is false [7], but [16] shows that in practice on real web graphs this heuristics works well; and although other algorithms for local score approximation exist (see [16]), they have less clear theoretical backgrounds, make unrealistic assumptions (such as the knowledge of the number of arcs in the graph), or give approximations that may exceed the true PageRank score – all assumptions that do not fit our general model. We terminated PBF either when no more candidates had a sufficient estimated contribution, or when the cost reached a threshold of $0.1n$, at which point it could be considered $\Theta(n)$. On the subgraph induced by the visited nodes, ImPBF runs BF for 40 iterations (while PBF typically did not visit more than 20th layers) or until the relative per-iteration increment in the estimated score drops below 0.1%. This "squeezes" out most of the overall contribution 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 3.12 - 3.13 (`.it` graph) and 3.15 - 3.16 (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 $(\epsilon, 2\epsilon)$; Figures 3.14 (`.it` graph) and 3.17 (LiveJournal graph) show, for different values of $(\epsilon, 2\epsilon)$, the competitive ratio of ImPBF, i.e. its average cost over the average size of the estimated minimal sets, as a function of the contribution threshold.

In the `.it` web graph, ImPBF guaranteed a precision ≥ 0.9 for any $\epsilon > 0.02$

incurring a cost within a factor 2.5 to 5 of the estimated minimal set size. Thus, its performance is reasonably close to the optimum even for very small separations, yet the cost still ranges in the hundreds of thousands, or in the order of $0.01n$ – too high in many cases.

In the LiveJournal graph, ImPBF guaranteed a precision ≥ 0.9 only for $\epsilon \geq 0.64$, incurring a cost of a few thousand nodes, or approximately 0.1 times the average estimated minimal set size. For any $\epsilon < 0.64$, 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. Note that the precision of ImPBF in the LiveJournal graph tends to decrease as the contribution threshold lowers and the cost increases. This counterintuitive behaviour is likely due to the saturation of the cost threshold set to $0.2n$, and confirms that local ranking is a definitely non-trivial problem.

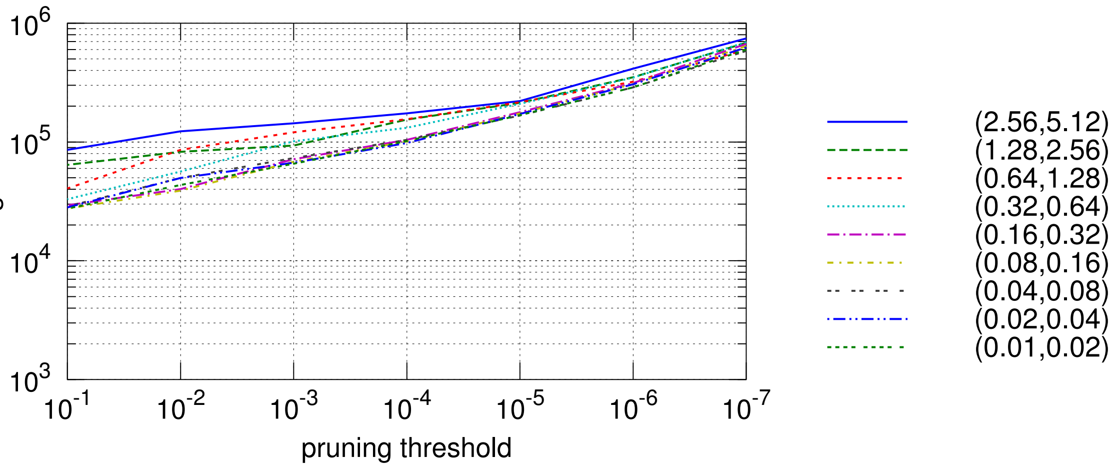

Figure 3.12: `.it` graph, ImPBF algorithm : average cost vs. contribution threshold, for different values of $(\epsilon, 2\epsilon)$

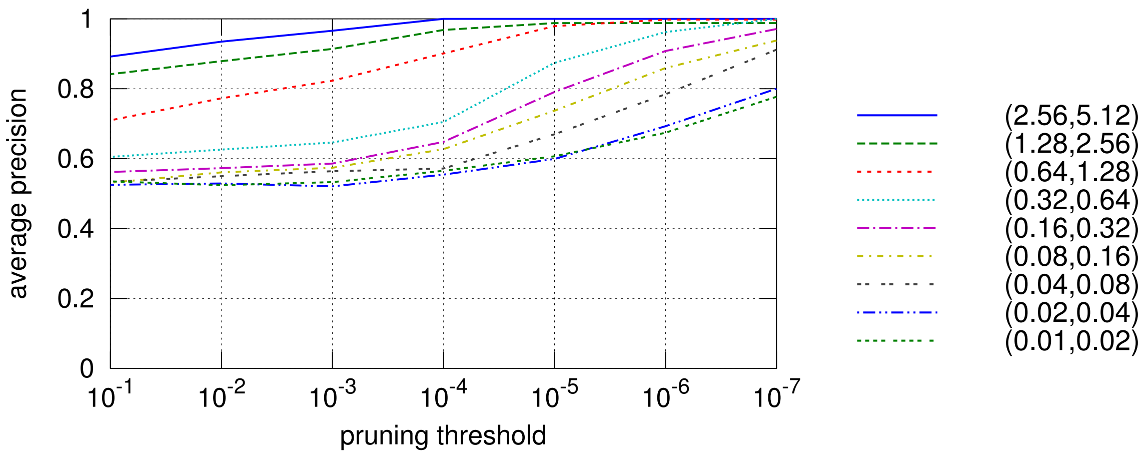

Figure 3.13: `.it` graph, ImPBF algorithm : average precision vs. contribution threshold, for different values of $(\epsilon, 2\epsilon)$

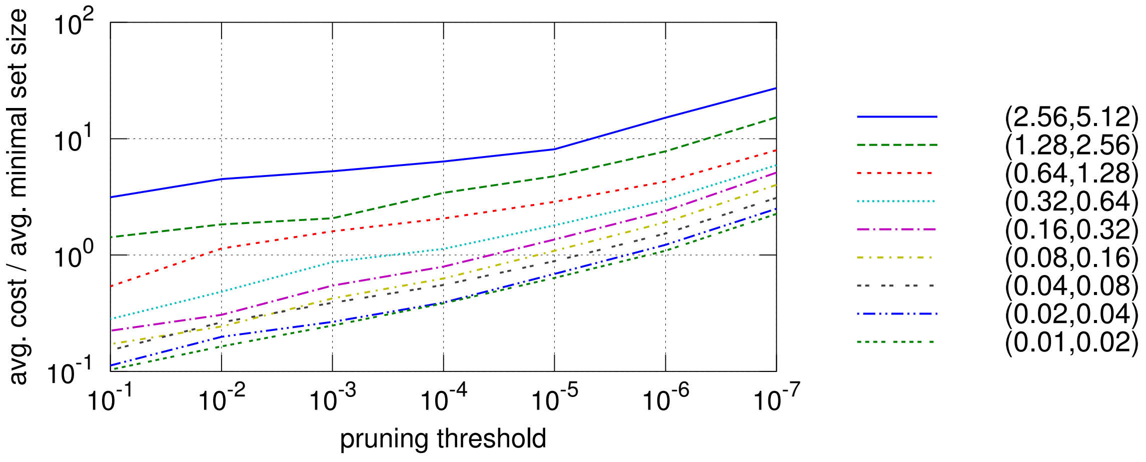

Figure 3.14: `.it` graph, ImPBF algorithm : ratio of average cost to average size of estimated minimal sets, for different values of $(\epsilon, 2\epsilon)$

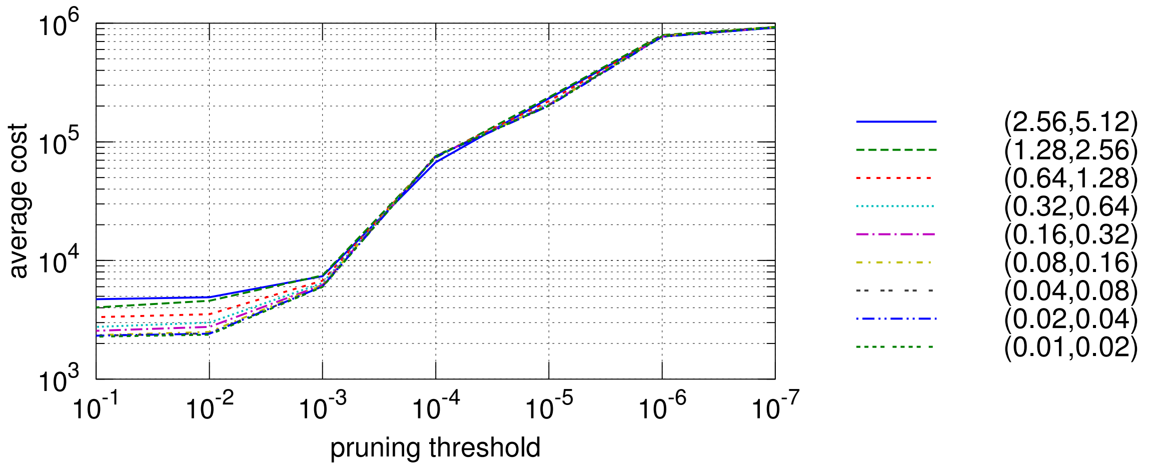

Figure 3.15: LiveJournal graph, ImPBF algorithm : average cost vs. contribution threshold, for different values of $(\epsilon, 2\epsilon)$

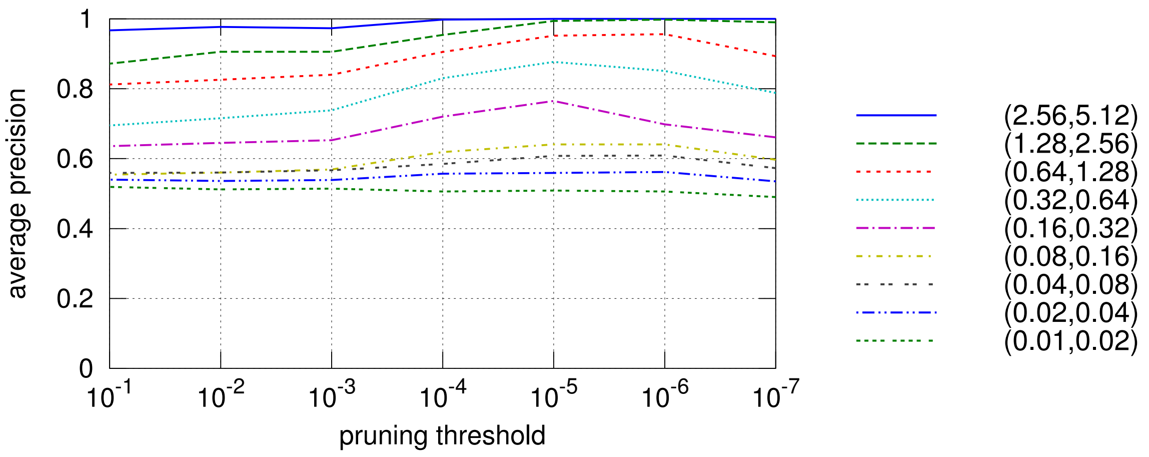

Figure 3.16: LiveJournal graph, ImPBF algorithm : average precision vs. contribution threshold, for different values of $(\epsilon, 2\epsilon)$

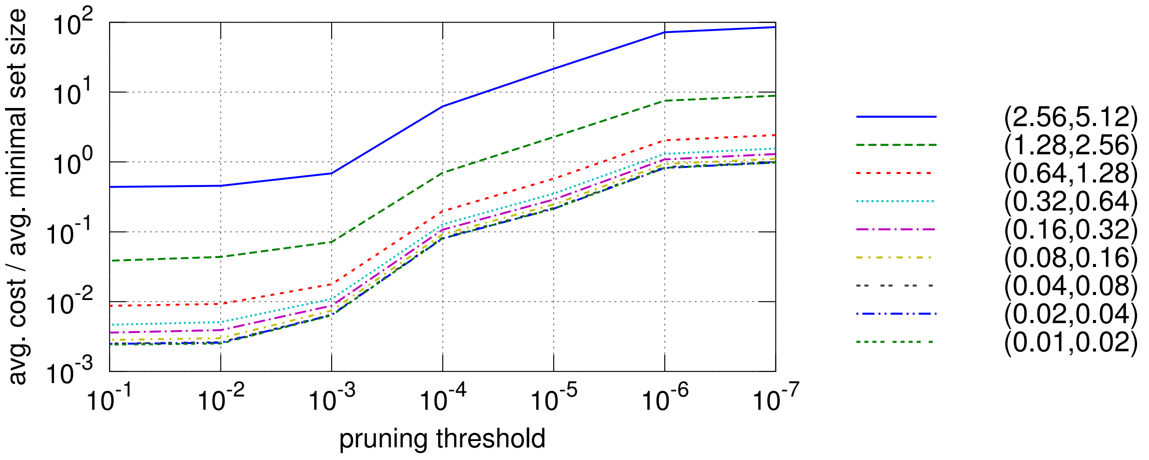

Figure 3.17: LiveJournal graph, ImPBF algorithm : ratio of average cost to average size of estimated minimal sets, for different values of $(\epsilon, 2\epsilon)$

3.6 Ranking is not local

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 the ranking by visiting only small subnetworks around the nodes that represent each candidate (i.e. by considering only the friends of the candidates, the friends of their friends and so on, but without going too far in the network)? Motivated by questions of this kind - naturally arising in the large and still growing number of IR applications which use PageRank only for its ranking capabilities - this is the first work to define and address the general problem of locally computing the ranking induced by PageRank on k target nodes of a graph. Our work shows that locally computing the ranking is in general infeasible by proving that, in an n-node graph, every deterministic algorithm may fail to produce the correct ranking when visiting less than $\Omega(n)$ nodes and every randomized algorithm may fail to produce the correct ranking when visiting less than $\Omega(\sqrt{kn})$ nodes in expectation.

Our research provides the notion of *minimal set*, which characterizes and allows to compute, on a given graph, the minimum number of nodes any correct algorithm must visit – leaving open the issue for different or more specific classes of algorithms. Experiments carried out on large, publicly available crawls of the web and of a social network show that also in practice the size of minimal sets may be considerably large, even if the ranking algorithms lever on efficient local score approximations. Indeed, (simple) variations of the naive (and highly inefficient) brute force algorithm incur costs close to the minimum cost even to guarantee a 90% rate of correctness, definitively confirming that ranking is non-local. Furthermore, the size of minimal sets may be surprisingly higher in smaller, more sparse graphs than in larger, more dense ones; their characterization in terms of properties of the underlying graph deserves more study.

The experiments raised a major flaw in existing literature on local approximation of PageRank scores, regarding the lack of a proper treatment of the (frequent) case of graphs with dangling nodes. Investigating this question is certainly an interesting direction for future research.

Chapter 4

Conclusions

This thesis addresses the fundamental question of how robust is the ranking induced by PageRank to variations in parameters which should, intuitively, have little weight.

In terms of variations in the damping factor we show that, in some cases, PageRank is extremely unstable. Indeed, arbitrarily small perturbations may completely reverse the ranking of the top nodes in the graph. One would like to predict these situations but, as we prove, the intuitive method of sampling rank for a discrete set of different damping factors does not give any definitive information about its stability. Thus, previous experimental results only suggest, but not prove, that in real graphs rank is insensitive to variations of the damping factor. Fortunately, there exist mathematical tools to analyze these situations. The novel concept of lineage analysis provides a simple and natural interpretation of rank stability, and allows one to assess whether the ranking of the nodes of a particular graph is "robust" to variations of the damping factor. It turns out that theoretical worst-case graphs do not arise in practice: in real graphs, ranking appears to be considerably but not totally stable with respect to the damping factor. This marginal sensitivity can be exploited for tuning purposes. In this sense, we develop two ranking metrics that estimate the best damping factors under different user scenarios, and provide a further motivation for the empyrical choice of some "classic" values of the damping factor.

Another factor that should, intuitively, have little weight on the relative ranking of two nodes is the link structure in remote portions of the graph. Unfortunately, we show this is not generally the case for the ranking induced by PageRank - both in theory and in practice. The first step is investigating if it is possible to compute the relative ranking of a set of nodes taking into account only small "local" subgraphs. We first show that the classic brute force, breadth first algorithm is impractical: it may both be inefficient and return a wrong ranking. However, other algorithms could in theory perform better, especially when ranking nodes whose PageRank *scores* are significantly separated. It turns out that, even in this case, no algorithm (randomized or deterministic) can give a correct ranking without visiting a large fraction of the graph – at least for some graphs. However, one could guess that real graphs are not as "hard" as the worst cases predicted by theory, and that the minimum number of nodes to take into account to provide a correct ranking is reasonably small. Surprisingly, it turns out that, in real graphs, one must take into account an impractically high number of nodes to provide a correct ranking even with a modest confidence level. Furthermore, this property of "intractability" does not appear to be easily predictable from the general properties of a graph. Our experiments expose a major flaw in the existing literature on the local approximation of PageRank scores, regarding the lack of proper treatment of the (frequent) case of graphs with dangling nodes.

This work leaves open several directions of future research, both theoretical and experimental. Perhaps the most promising involves a generalization of our results to other ranking algorithms that infer ranks from scores, such as HITS or SALSA. Previous work addresses (at least partially) the issue for HITS [37, 38], but there is still a wide gap between known results and a general theory that covers the whole class of ranking algorithms. Another promising direction is the study of robustness with respect to other factors. For example, it woould be interesting to investigate how the ranking vector changes in response to very small perturbations in the input graph, such as the deletion or addition of few nodes or arcs. As we remark in Section 3.5, there is also a major flaw in the existing literature on local score approximation: previous research ignores the presence of dangling nodes, which may considerably affect the final score (and rank) of each node. Investigating the impact of dangling nodes - both in theory and in practice - also deserves more effort. Finally, we did not address issues of numerical stability, which may have great importance especially when dealing with (variations of) the damping factor, yet another crucial direction of future research.

List of Figures

Bibliography

[1] *CiteSeer metadata.* `http://citeseer.ist.psu.edu/oai.html`.

[2] Brian Amento, Loren Terveen, and Will Hill. Does “authority” mean quality? Predicting expert quality ratings of web documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 296–303, 2000.

[3] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcroft, Vahab Mirrokni, and Shang-Hua Teng. Local computation of PageRank contributions. *Internet Mathematics*, 5(1–2):23–45, 2008.

[4] Konstantin Avrachenkov, Nelly Litvak, and Kim Son Pham. A singular perturbation approach for choosing PageRank damping factor. *ArXiv Mathematics e-prints*, 2006.

[5] Ricardo Baeza-Yates, Paolo Boldi, and Carlos Castillo. Generalizing PageRank: Damping functions for link-based ranking algorithms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 308–315, 2006.

[6] Ziv Bar-Yossef and Li-Tal Mashiach. Local approximation of PageRank and reverse PageRank. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM)*, pages 279–288, 2008.

[7] Ziv Bar-Yossef and Li-Tal Mashiach. Local approximation of PageRank and reverse PageRank. Technical report, Israel Institute of Technology, 2008.

[8] Paolo Boldi. Totalrank: ranking without damping. In *Proceedings of the 14th ACM International World Wide Web Conference (WWW) (Special interest tracks and posters)*, pages 898–899, 2005.

[9] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[10] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. PageRank as a function of the damping factor. In *Proceedings of the 14th ACM International World Wide Web Conference (WWW)*, pages 557–566, 2005.

[11] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proceedings of the 13th ACM International World Wide Web Conference (WWW)*, pages 595–601, 2004.

[12] Marco Bressan and Enoch Peserico. Choose the damping, choose the ranking? In *Proceedings of the 6th International Workshop on Algorithms and Models for the Web Graph (WAW)*, pages 76–89, 2009.

[13] Marco Bressan and Enoch Peserico. Choose the damping, choose the ranking? *Journal of Discrete Algorithms (JDA)*, 8(2):199–213, 2010.

[14] Sergey Brin and Lawrence Page. The anatomy of a large scale hypertextual web search engine. In *Proceedings of the 7th ACM International World Wide Web Conference (WWW)*, 1998.

[15] Harry Buhrman and Ronald De Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288:2002, 1999.

[16] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. Local methods for estimating PageRank values. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 381–389, 2004.

[17] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through URL ordering. *Computer Networks*, 30(1–7):161–172, 1998.

[18] Jason V. Davis and Inderjit S. Dhillon. Estimating the global PageRank of web communities. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 116–125, 2006.

[19] Stephen Dill, Ravi Kumar, Kevin S. McCurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. *ACM Transactions on Internet Technology*, 2(3):205–223, 2002.

[20] Güneş Erkan and Dragomir R. Radev. LexRank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research (JAIR)*, 22:457–479, 2004.

[21] Andrea Esuli and Fabrizio Sebastiani. PageRanking WordNet synsets: An application to opinion mining. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 424–431, 2007.

[22] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 28–36, 2003.

[23] Floris Geerts, Heikki Mannila, and Evimaria Terzi. Relational link-based ranking. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 552–563, 2004.

[24] David Gleich, Peter W. Glynn, Gene H. Golub, and Chen Greif. Three results on the PageRank vector: eigenstructure, sensitivity, and the derivative. In *Web Information Retrieval and Linear Algebra Algorithms*, 2007.

[25] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with TrustRank. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 576–587, 2004.

[26] D. G. Hook and P. R. McAree. Graphics gems. chapter Using Sturm sequences to bracket real roots of polynomial equations, pages 416–422. Academic Press Professional, Inc., 1990.

[27] X. M. Jiang, G. R. Xue, H. J. Zeng, Z. Chen, W.-G. Song, and W.-Y. Ma. Exploiting PageRank at different block level. In *Proceedings of the ACM Workshop on Wireless Security (WISE)*, 2004.

[28] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th ACM International World Wide Web Conference (WWW)*, pages 508–516, 2003.

[29] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46:604–632, 1999.

[30] Donald E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley Professional, third edition, November 1997.

[31] Oren Kurland and Lillian Lee. PageRank without hyperlinks: Structural reranking using links induced by language models. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 306–313, 2005.

[32] Amy N. Langville and Carl D. Meyer. Deeper inside PageRank. *Internet Mathematics*, 1(3):335–380, 2004.

[33] Amy N. Langville and Carl D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, Princeton, 2006.

[34] Massimo Melucci and Luca Pretto. PageRank: When order changes. In *Proceedings of the 29th European conference on IR research (ECIR)*, pages 581–588, 2007.

[35] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: bringing order to the Web. Technical report, Stanford InfoLab, 1998.

[36] Enoch Peserico and Luca Pretto. What does it mean to converge in rank? In *Proceedings of the 1st International Conference on Theory of Information Retrieval (ICTIR)*, pages 239–245, 2007.

[37] Enoch Peserico and Luca Pretto. HITS can converge slowly, but not too slowly, in score and rank. In Hung Q. Ngo, editor, *Computing and Combinatorics - 15th Annual International Conference, COCOON 2009*, volume 5609 of *Lecture Notes in Computer Science*, pages 348–357, Berlin Heidelberg, 2009. Springer.

[38] Enoch Peserico and Luca Pretto. Score and rank convergence of HITS. In Mark Sanderson, ChengXiang Zhai, Justin Zobel, James Allan, and Javed A. Aslam, editors, *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 770–771, New York, 2009. ACM.

[39] Luca Pretto. A theoretical analysis of Google's PageRank. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 131–144, 2002.

[40] Dragomir R. Radev and Rada Mihalcea. Networks and natural language processing. *AI Magazine*, 29(3):16–28, 2008.

[41] Paul Tarau, Rada Mihalcea, and Elizabeth Figa. Semantic document engineering with WordNet and PageRank. In *Proceedings of the 20th Annual ACM Symposium on Applied Computing (SAC)*, pages 782–786, 2005.

[42] University of Milan - DSI. *Laboratory for Web Algorithmics*. `http://law.dsi.unimi.it/`.

[43] James H. Wilkinson. *Rounding Errors in Algebraic Processes.* Dover Publications, Incorporated, 1994.

[44] Rebecca S. Wills and Ilse C. F. Ipsen. Ordinal ranking for Google's PageRank. *SIAM Journal on Matrix Analysis and Applications*, 30(4):1677–1696, 2009.

[45] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.