LOCAL APPROXIMATION OF PAGE CONTRIBUTIONS IN THE PAGERANK ALGORITHM

by

Shabnam Shariaty

B.Sc., Sharif University of Technology, Tehran, Iran, 2009

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

APPROVAL

Name:	Shabnam Shariaty
Degree:	Master of Science
Title of Thesis:	Local Approximation of Page Contributions in the PageRank Algorithm
Examining Committee:	Prof. Ramesh Krishnamurti, Chair
	Prof. Andrei Bulatov, Senior Supervisor
	Prof. Petra Berenbrink, Supervisor
	Prof. Jian Pei, SFU Examiner
Date Approved:	1 December 2011

Abstract

Search engines are a key factor shaping the way people interact with today's worldwide web. It is therefore of critical importance for web masters to understand and increase the ranking of their pages, and so is for search engines to inspect how a page obtains its ranking in order to detect possible malicious activities for manipulating the natural ranking of pages. PageRank is a popular algorithm used by search engines such as Google to rank web pages returned as search results. This algorithm assigns a score to each web page (i.e., graph node) reflecting its importance, which is a function of the score of other pages having a hyperlink or a path of hyperlinks (graph links) to that page. In addition to the web graph, PageRank is also used for ranking graph nodes in other contexts such as the citation network of the scholarly literature. In this thesis, we take a closer look at the PageRank algorithm on graphs and analyze how each graph node collects its PageRank score from other nodes. We develop a systematic method for calculating the contribution that individual nodes make to each other's PageRank score, i.e., the difference that it would make in the PageRank score of node v if node u did or did not exist. We then present an approximation algorithm with guaranteed error bounds for approximating page contributions to any given target node v, which operates in a local neighborhood of v in the graph since real-world graph such as the web is too large to be computed on as a complete graph. We evaluate our algorithm on a web graph and a citation graph dataset. Our experimental results indicate that we can estimate the page contribution values with good accuracy. Moreover, our results show that we can find higher-contribution supporter nodes for a given target node in a shorter time than previous works.

Keywords: PageRank algorithm, page contribution, path contribution, local approximation.

Acknowledgments

First, I am heartily and deeply thankful to my senior supervisor, Dr. Andrei Bulatov, who has supported me thoughout my thesis from the initial to the final level with his patience and knowledge. Anytime I needed help, he made his time available for me and helped me with valuable advice and guidance whilst allowing me the room to work in my own way.

I would like to express my gratitude to Dr. Petra Berenbrink, my supervisor, and Dr. Jian Pei, my thesis examiner, for being on my committee and reviewing this thesis. I also would like to thank Dr. Ramesh Krishnamurti for taking the time to chair my thesis defense.

Last but certainly not least, I owe my deepest gratitude to my parents, friends and family for their love, encouragement, and endless support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Search engines play a critical roll in the way people interact with the worldwide web. Given a search query, search engines return a set of pages ordered according to a score for each page, which is a combination of a static ranking score pre-calculated for the page as well as a dynamic relevance score for the page with respect to the search query. Intuitively, the static ranking score of a page reflects the importance of the page on the web. PageRank is the most popular algorithm for this score which is used by Google, the most widely used search engine today. PageRank is a link analysis algorithm which assigns a numeric value to each page v, based on the number of pages that have hyperlinks to v as well as their own PageRank scores. In other words, pages that are referred to by many other important pages receive a higher score. Therefore, the PageRank score of a page depends on other pages which have a direct link or a sequence (path) of links to the target page.

The application of the PageRank algorithm is not limited to the web graph only. This algorithm is also used for ranking graph nodes in other contexts such as scholarly citation networks and social networks. In a citation graph, nodes and links represent papers and citations between papers, respectively. Similar to the web, a static ranking score is calculated for each paper which is used in the ordering of search results. Google Scholar, the most widely used scholarly search engine today, uses a PageRank-based algorithm for this purpose [5, 15]. Similarly in social networks, PageRank-based algorithms are used for assigning various importance metrics (e.g., trust and reputation) to social entities [14, 21, 24].

For the owners of many commercial and personal web sites, it is very important to appear at the top of search results. This is because pages that rank higher in search results get a much higher user traffic. Therefore, web masters always try to obtain higher PageRank scores for their web sites through various techniques and tricks such as legal Search Engine Optimization techniques. Some of these techniques even involve malicious activities such as link-spamming, which we will discuss shortly. It is thus important for both web masters and search engines to be able to inspect the PageRank score of a page and analyze how it is obtained. This information will allow a web master to understand how her pages interact with each other and with other pages on the web for collecting PageRank scores, and thus to optimize her pages for higher ranking. In addition, this information will help search engines in finding and counteracting link-spamming activities on the web [11, 29, 31] and possibly on the scholarly network [5, 6, 15]. These issues, detecting spamming and optimizing page rankings, have both been hot topics of research and highly demanded by the industry in recent years.

1.1 Problem Statement and Contributions

In this thesis, we study the problem of PageRank contribution, and analyze the contribution that nodes of the graph make to each other's PageRank score—the increase in the PageRank score of node v that is brought by node u. This is a fundamental problem for the study of PageRank-related issues discussed earlier, and therefore has been studied by various previous works in the literature. [1, 31].

The contribution values calculated by most previous works measures the contribution of node u to the PageRank score of node v according to the paths (i.e., sequence of links) from u to v. However, an important factor in the contribution made by node u to node v is the PageRank of node u itself. That is, it can make a significant difference for the PageRank of node v to have a path from a low-PageRank node u or to have the same path from a high-PageRank node u'. The contribution of u and u', however, will be considered the same by most previous works. We will refer to this type of PageRank contribution as *path contribution*. On the other hand, we consider the contribution of node u to node v as the difference that it would make in the PageRank score of v if node u did or did not exist. We will refer to this contribution, which is our focus in this thesis, as *page contribution*.

We develop a systematic method for finding the page contribution of nodes to each other. While this page contribution value has been considered by an earlier work [31] as well, to the best of our knowledge, no systematic algorithm for estimating these contribution values have been proposed in the literature before. Moreover, we design approximation algorithms with guaranteed running time and worst-case approximation ratio for finding these contribution values locally in a graph. We also demonstrate through experiments on real-world datasets that our algorithms can estimate page contributions with small errors and reasonable running time. Moreover, our algorithms can find supporting sets with the higher page contribution in a shorter running time than the previous works.

1.2 Applications

Given the crucial importance of search engine rankings for web pages, various techniques have been developed for web masters to improve the ranking of their pages in search results, enabling them to attract more user traffic. Therefore, *Search Engine Optimization* (SEO) has become a very successful market in recent years. As an example, consider a user study indicating that 92% of users prefer using natural search results (as opposed to advertised results) when looking to buy a product [28].

Therefore, different SEO methods have been developed, including various search engine spamming techniques. In particular, to deceive the PageRank algorithm and give a boost to the ranking of a particular page, link-spamming techniques often create a group of web pages linking to each other and to the target pages. Rather than representing a natural hyperlink from page A to page B, these spam links are added only for the purpose of misleading the PageRank algorithm. These techniques are not approved by search engines, and search engines constantly try to detect and ban such web pages from search results.

Our algorithms provide an effective tool for this detection. Most link-spam detection techniques are based on finding and analyzing the *supporting set* of a given web page, that is a group of pages that contribute the most to the ranking of the target page. Then, various algorithms and heuristics are applied on supporting sets to decide if it is legitimate or spam. Our algorithms can efficiently find these supporting sets for any given target page. In particular, we demonstrate that the supporting sets found by our algorithms include pages with higher page contributions than the supporting sets found by previous works. In addition to finding the supporting sets, our algorithms report the contribution of each supporting page to the target page. Revealing such fine-grained information can provide further advantages for the analysis of spam activities.

Aside from search engine spamming, which are known as *black hat* SEO techniques, legitimate SEO methods have had a prosperous market in recent years. These methods, which are referred to as *white hat* SEO, try to increase the ranking of a page legitimately and are even recommended by search engines as part of good design. Although SEO services are being widely traded today, it is not clear how one can quantize the success of an SEO company in increasing the ranking of its page. While a simple way is to check the ranking of the pages before and after getting SEO support [23], this naive approach can only be used once at the beginning; it cannot help in assessing the effectiveness of existing SEO support, or that of a particular SEO company among multiple ones. However, this is an important question for any web master paying one or more SEO companies, e.g., to determine how much it is worth to pay an SEO company or to extend/discontinue a contract.

In addition to link-spam detection, our algorithms also have applications in the white hat SEO domain. These algorithms can compute the contribution that a page makes to any other page's PageRank score. Therefore, web masters can easily use these algorithms to estimate the effectiveness of a link-based SEO technique in boosting their pages' scores.

Our work is not limited to the web only, and is applicable in any network on which PageRank-based algorithms are applied, such as citation and social networks discussed earlier. Our algorithms can be used to find the contribution of nodes (scholarly papers) to each other's ranking score. Similar to the web graph, this information can assist in the detection of link spamming in scholarly search engines such as Google Scholar [5,6,15]. Moreover, they can also help in extracting useful information from the citation network, such as the papers that have the highest page contributions to each given target paper. This can indicate some level of similarity between the papers' topics and their influence from the corresponding target paper, which are useful information for further analysis of citation networks such as finding a number of key, influential papers in a given scholarly literature.

Furthermore, our algorithms can be used in social networks where PageRank-based algorithms are used to assign different importance scores to nodes [14,21,24]. Our algorithms enable new analyses on these networks, such as how individual social entities influence each other's trust or reputation score. Moreover, similar to the web, our algorithms are useful against the potential threat of spamming in social rankings, where on may add extra entities and/or relationships to the network just to increase the trust score of a malicious user.

1.3 Thesis Organization

We review the related work in Chapter 2. Chapter 3 provides a systematic method for calculating the page contribution, followed by an approximation algorithm in Chapter 4 to estimate these values locally. In chapter 5, we evaluate the accuracy of our algorithm for approximating page contributions as well as its running time. Finally, we conclude the thesis and highlight a number of future research directions in Chapter 6.

Chapter 2

Related Work

In this chapter, we review previous work related to calculating contributions in the PageRank algorithm and finding supporting sets (also known as page farms in some works). Moreover, since in many previous works page farms are used for the purpose of detecting link spamming, we also review related papers on spam detection in which page farms are studied as well. We put the previous work in two categories as follows.

The papers in the first category study the concept of page farms and supporting sets, and in some cases present methods for extracting such sets. In an early work in this direction, Lempel et al. [19] have shown that the PageRank algorithm is not rank-stable. That is, the ranking of web pages can significantly change by small modifications in the link structure of the web graph. Later, several papers have studied the relationship between PageRank scores and the structure of the corresponding supporting sets. Sydow [27] has shown that the PageRank score of a web page can be significantly increased by adding carefully chosen outgoing links to the page itself. The optimal set of such links for the page is analyzed in [2]. Kerchove et al. [17] study linkage strategies for a given set of pages so that their total PageRank score is maximized. Du et al. [12] focus on a similar problem for spam farms, where the structure of a page farm is optimized for boosting the ranking of a target page. The same problem is considered in [13], which also investigates how groups of farms can collaborate and form spam farm alliances that benefit all participants. While these works conduct an in-depth analysis of potential structures of page farms, they do not provide any methods for finding these farms.

In order to find page farms for web spam detection, Saito et al. [26] have proposed to decompose the web graph into strongly connected components (SCC). Then, they empirically showed (on sample web data sets) that large components are spam farms with high probability. In a later work by the same authors, a recursive SCC decomposition is introduced in [11] to extract such farms. This is an efficient approach for finding farms given that we can find SCCs. However, we note that generally finding SCCs in the web graph takes a time proportional to the size of the whole graph, which is infeasible for the web. Largillier et al. [18] have proposed to simulate random walks of a random surfer up to length l. Then the authors analyze the patterns observed in the visited paths (e.g., repeated pages) to find page farms and those pages benefiting from the farms. The proposed algorithm is designed for finding some random farms in the web, and it does not provide a method for finding the supporting set of a given target page.

Becchetti et al. [3, 4] assumed that the supporting set of a target page is within a short distance d of it (moving backwards on incoming links), and proposed to ignore these pages in the PageRank score of the target page so that the effect of spam page farms is neutralized. Moreover, a probabilistic method for estimating the size of the supporting set of a target page is introduced. A random bit vector is assigned to each page. Then, several paths of length d that end at the target page are visited, and the bit vector of each page is updated according to its incoming neighbor on the path: it is bitwise-ORed with the bit vector of the neighbor. Then, intuitively, if a target page has a larger number of supporters than another, more distinct bit vectors will be ORed with its vector, and more 1s will appear in its final bit vector. The number of 1s therefore provides an estimate on the number of supporters. This work allows to flexibly adjust the running time and accuracy through the parameter d and the number of paths to visit. Nevertheless, it does not find the supporting set of the target page; it only provides an estimate on its size.

Benczur et al. [7] propose to perform a Monte Carlo simulation of random walks from random pages on the web (based on the algorithm in [19]), and then analyze the PageRank distribution of the supporters of each page to decide spamicity. However, this practice is computationally expensive since it cannot find the supporting set of just a given target page.

Wu et al. [30] presented a two-step algorithm for finding page farms in order to detect web spam. First, based on the observation that pages in such farms have many incoming and outgoing neighbors in common, pages whose incoming and outgoing set are overlapping beyond a threshold are considered as an initial seed farm. Then, this farm is expanded by adding new pages to it that have many links to pages already in the farm. In [29] the same authors proposed to identify page farms by performing a random walk from a few manually detected pages of the farm. This is based on the fact that a farm (specifically a spam farm) has a dense link structure, hence a random walk is likely to remain in the farm for a while. These two algorithms can be effective methods for finding spam farms, given a good initial seed. On the other hand, it is not clear which pages are analyzed at the beginning to form the initial seed. Also we note that these algorithm, specially the one in [30], is based on a specific farm structure and predefined thresholds, which can make it easy to bypass (and make the farm undetectable) by a careful spammer.

The second category of papers, besides finding page farms, study the problem of calculating contributions in the PageRank algorithm. The first algorithm in this category has been proposed by Jeh and Widom [16]. This algorithm is essentially designed for computing personalized PageRank values, and can be used to calculate the PageRank contribution that node u makes to other nodes in the graph. While an efficient solution for the personalized PageRank problem, this algorithm cannot find the PageRank contributions of all nodes to a given target node v, which is the problem of our interest. This is because we need to repeat the algorithm for many (possibly all) nodes in the graph, which is infeasible. Moreover, the contribution values calculated by this work are path contributions as opposed to our desired page contributions (Section 1.1).

Contributions received by a target page v from other pages of the graph has been investigated by Andersen et al. [1]. The authors presented a local approximation algorithm for finding a PageRank contribution vector representing contributions made to v, as well as an algorithm for finding the supporting set of page v. The former algorithm employs the idea of [16] in reverse direction: the score of the target page is pushed back repeatedly to its incoming neighbors, then to the neighbors of the neighbors and so on, until the value associated with each page becomes smaller than the desired error ϵ. This provides an approximate on the contribution of each page u to page v. To find the smallest farm that supports v beyond a given threshold ρ (i.e., a set of pages from which a ρ fraction of the PageRank score of page v comes from), the aforementioned algorithm for finding contribution vectors is repeated in a binary search procedure to find the minimum farm size. This algorithm performs efficiently in approximating contribution values with guaranteed error ϵ. However, as discussed earlier, the contribution values considered in this paper are only path contributions, i.e., the PageRank contribution made by paths starting exactly at u and ending at v. However, we note that a significant portion of the PageRank score transferred from u to v depends on the state of u itself in the graph; a page u_1 with no incoming links can make much less contribution to v than a page u_2 with a reasonable PageRank score which itself is the destination of many paths. We capture this fact by calculating page contributions as the difference that the existence or non-existence of page u makes in the PageRank score of page v.

To the best of our knowledge, the only previous work considering page contribution values is the algorithm proposed by Zhou and Pei [31]. The algorithm heuristically searches in a local neighborhood of the target page up to a given distance k in order to reach to a given threshold θ to the PageRank of the target page. If the algorithm does not reach to the threshold, then the whole local neighbors of distance k reported as a supporting set. This work, for the first time, considers the notion of page contribution separate from the path contribution, however, no systematic method for calculating page contribution values is provided. Moreover, no approximation guarantee is provided on the size and accuracy of the returned page farms.

Our work, on the other hand, presents a systematic method for calculating page contribution values, and also includes a local approximation algorithm with guaranteed error bounds. We first formulate the page contributions suggested by Zhou and Pei in [31] as closed-form matrix equations, and then develop a new algorithm for local approximation of these values which builds upon the algorithm in [1].

Chapter 3

Page Contributions in the Graph

In this chapter, we provide a new method to calculate the page contribution of nodes to each other. We first review basic preliminaries from the literature, and then present a systematic method for obtaining page contributions. We then formulate this contribution value as a series of matrix equations, which forms the basis of our algorithms in the next chapter.

3.1 Preliminaries

We model the web with a directed graph $G = (V, E)$ in which each node represents a page and a directed edge from node u to node v represents a hyperlink from page u to page v; similarly, in a citation graph each node represents a paper and a link from node u to node v exists iff paper u cites paper v. N denotes the number of nodes of the graph.

The PageRank algorithm and the concept of page contribution discussed shortly are primarily defined for web pages, but they are also used in other contexts such as citation networks. We use both terms page and node depending on the context to refer to the nodes of the graph. We also refer to graph nodes simply by their indices, e.g., node i where $1 \leq i \leq N$. Let $I(v)$ denote the set of incoming neighbors of node v, $O(v)$ the set of outgoing neighbors of node v, and $ODeg(v) = |O(v)|$ and $IDeg(v) = |I(v)|$ be the number of its outgoing and incoming links, respectively.

Table 3.1: Notations used in the thesis.

Notation	Description
G, V, E, N	Graph $G = (V, E)$. $N = \|V\|$.
$G(U)$	Induced subgraph created by voiding (removing all outgoing links) of nodes $u \notin U$ in the original graph.
$I(v), IDeg(v)$, $O(v), ODeg(v)$	$I(v), O(v)$: the set of incoming and outgoing neighbors of node v, respectively. $IDeg(v) = \|I(v)\|$, $ODeg(v) = \|O(v)\|$.
α, d	α is the random jump probability. $d = 1 - \alpha$ is the damping factor.
$L_{N \times N}$	Left-stochastic matrix of the graph (each column sums to 1).
$SPcont(v, u)$	Sum of the contribution of all paths from u to v.
$M_{N \times N}$	Matrix of total path contribution: $M[v, u] = SPcont(v, u)$.
$\mathbf{e}_v$	Unit vector whose v-th entry is 1 and all others are 0.
$\mathbf{pr}(u)$	PageRank vector of the graph ($\sum_{u \in V} \mathbf{pr}(u) = N$).
$\mathbf{cv}_v$	Path contribution vector of v where $\mathbf{cv}_v(u) = SPcont(v, u)$.
$\mathbf{cpv}_v$	Augmented path contribution vector of v; $\mathbf{cpv}_v(u) = \mathbf{cv}_v(u) \times \mathbf{pr}(u)$.
$\mathbf{gcv}_v$	Page contribution vector of v where $\mathbf{gcv}_v(u)$ is the page contribution of node u to node v.

3.1.1 Stochastic Matrices

A *stochastic matrix*, also called a probability or transition matrix, represents transition probabilities in a graph. There are different types of stochastic matrices. A *right stochastic matrix*, is a square matrix in which each row consists of nonnegative real numbers, and sum up to 1. A *left stochastic matrix*, is a square matrix in which each column consists of nonnegative real numbers, and sum up to 1. A *doubly stochastic matrix* is a square matrix where all entries are nonnegative and all rows and all columns sum up to 1.

For our problem, we use a left stochastic matrix $L = \{l_{i,j}\}_{N \times N}$ where $l_{i,j} = 1/ODeg(i)$, if there is an edge from i to j. That is, $l_{i,j}$ represents the probability of going from node i to node j assuming a uniform distribution among all outgoing neighbors of i.

3.1.2 The PageRank Algorithm

The PageRank algorithm was first formally described by Brin et al. [8], and is used for ranking pages by the Google search engine. It is an iterative algorithm for measuring the relative importance of web pages. Consider a random surfer on the web that can start from any page and keeps following the outgoing links of pages. The surfer may also get bored at some point and jump to some random page on the web (not necessarily a page linked to by the current page). The probability of this random jump is denoted by α. Assuming the surfer is currently on page v which has $ODeg(v)$ outgoing links, in the next step the surfer either jumps to one of the $ODeg(v)$ neighbors of v with probability $(1-\alpha)/ODeg(v)$ for each neighbor, or to any other of the N page on the web (including itself) with probability α for each page. We also note that the random surfing model in the literature is sometimes defined based on d, the damping factor which is equal to $d = 1 - \alpha$.

The PageRank score of a node is defined as the probability of the random surfer being at node v in the steady state, i.e., in the long term, where the probability assigned to each node is consistent with the probabilities assigned to its neighbors. This can be formally defined as follows.

Definition 1 (PageRank Score)**.** *The PageRank score of node v on graph G is given as:*

$$PageRank(v,G) = \alpha + (1-\alpha) \sum_{u \in I(v)} \frac{PageRank(u,G)}{ODeg(u)}. \tag{3.1}$$

The PageRank algorithm is often implemented in an iterative manner. First, an arbitrary value (e.g., 1) is assigned as the PageRank score of each node, and then Equation (3.1) is repeatedly applied until the PageRank vector converges.

3.1.3 Induced Subgraph

We use a definition of induced subgraph suggested in [31], which is different from the conventional definition of induced subgraph in graph theory.

Definition 2 (Induced Subgraph)**.** *For a set of vertices U, the induced subgraph of U is given by $G(U) = (V, E')$, where $E' = \{v \to u | (v \to u \in E) \wedge (v \in U)\}$. That is, in $G(U)$, all the vertices that are not in U are voided—its outgoing links are removed.*

3.1.4 Path and Page Contribution

The definition of page contribution as well as path contribution, and also the relationship between them is reviewed in this part.

An intuitive method for calculating the contribution of a node to the PageRank score of a given target node, is to neutralize the impact of the contributor node by removing its outgoing links. Then, we can measure the difference of the PageRank score of the target node with and without the neutralized node. The formal definition of page contribution based on this idea is as follows.

Definition 3. *(Page Contribution [31]). Consider graph $G = (V, E)$ and target node $v \in V$. The contribution of node $u \in V$ to the PageRank score of v is:*

$$PageCont(v, u) = PageRank(v, G) - PageRank(v, G(V - \{u\})) \tag{3.2}$$

Although this definition is simple and intuitive, for every node u we need to calculate PageRank score, which makes it inefficient especially for large graphs. An alternative way suggested in [31] is to calculate the page contribution through path contributions.

Definition 4. *(Path Contribution [31]). Consider graph $G = (V, E)$ and target node $v \in V$. Let $P = v_0 \to v_1 \to ... \to v_n \to v$ be a directed path from v_0 to v in the graph. The path contribution to the PageRank of v from P is defined as*

$$PathCont(P, v) = \alpha \prod_{i=0}^{n} \frac{1-\alpha}{ODeg(v_i)} \tag{3.3}$$

We show the sum of all path contributions from node u to v with $SPcont(v, u)$ as follows:

$$SPcont(v, u) = \sum_{\text{path } P \text{ from } u \text{ to } v} PathCont(P, v) \tag{3.4}$$

Also, based on a proposition in [9], the PageRank score of node v can be calculated using the path contributions as follows.

$$PageRank(v) = \sum_{x \in V} SPcont(v, x) \tag{3.5}$$

The following lemma states the relationship between page contribution and path contribution.

Lemma 1. *([31]) The page contribution of a node u to node v can be calculated through path contributions as follows:*

$$\begin{aligned} PageCont(v, u) &= \sum_{\textit{path } P \textit{ from } u \textit{ to } v} PathCont(P, v) \\ &+ \sum_{x \in V} \Big(\sum_{\textit{path } P \textit{ from } x \textit{ to } v \textit{ through } u} PathCont(P, v) \Big). \end{aligned} \tag{3.6}$$

Proof. According to Definition 3.2, for nodes v and u in graph $G = (V, E)$ we have:

$$PageCont(v, u) = PageRank(v, G) - PageRank(v, G(V - \{u\}))$$

Using Equation 3.5, we have:

$$PageCont(v,u) = \sum_{x \in V} SPcont(v,x) - \sum_{x \in V - \{u\}} SPcont(v,x). \tag{3.7}$$

Because the induced subgraph $G(V - \{u\})$ is obtained by removing the outgoing links of u, the set of paths remaining after the above subtraction consists of all paths from u to v as well as any path from another node x ($x \neq u$) to v that passes through u. $\square$

3.2 A Systematic Method For Calculating Page Contributions

A simple way of estimating the page contribution values defined in the previous section (Equation (3.6)) is to exhaustively enumerate all paths involved in $PageCont(v,u)$ and sum up their contributions. We propose an alternative method for this task in this section, which provides a closed form equation for $PageCont(v,u)$.

Our goal is to first calculate $PageCont(v,u)$ based on $SPcont(\cdot)$.

Lemma 2. *For $u, v, x \in V$ ($u \neq v$), let $\mathbf{S}$ denote the set of all paths from x to v that go through u, let $\mathbf{S}_1$ denote the set of all paths from x to u that visit u only once (i.e., no cycles over u), and let $\mathbf{S}_2$ denote the set of all paths from u to v that can visit u any number of times. The sum of the contribution of all paths from x to v that go through u can be found as:*

$$\sum_{P \in \mathbf{S}} PathCont(P,v) = \sum_{P_1 \in \mathbf{S}_1} \sum_{P_2 \in \mathbf{S}_2} PathCont(P_1 \cdot P_2, v), \tag{3.8}$$

where $P_1 \cdot P_2$ denotes concatenation of paths P_1 and P_2; clearly it always happens at u.

Proof. Let $\mathbf{S}' := \{P_1 \cdot P_2 \mid P_1 \in \mathbf{S}_1, P_2 \in \mathbf{S}_2\}$. We show that $\mathbf{S} = \mathbf{S}'$, and that no path is counted more than once in Equation (3.8).

First, it is easy to verify that $\mathbf{S} \subseteq \mathbf{S}'$: if there is a path P such that $P \in \mathbf{S}, P \notin \mathbf{S}'$, we consider the first time node u is visited in the path $P = x, \ldots, u, \ldots, v$ and we break it into two paths $P_A = x, \ldots, u$ and $P_B = u, \ldots, v$. From the definition of $\mathbf{S}_1$ and $\mathbf{S}_2$, it immediately follows that $P_A \in \mathbf{S}_1$ and $P_B \in \mathbf{S}_2$, and hence $P \in \mathbf{S}'$.

Next, we similarly verify that $\mathbf{S}' \subseteq \mathbf{S}$: each path $P \in \mathbf{S}'$ is a concatenation of a path from $\mathbf{S}_1$ and one from $\mathbf{S}_2$, so it would start at x, end at v, and visit u at least once. Hence, it is by definition one of the paths in $\mathbf{S}$. This leads to $\mathbf{S} = \mathbf{S}'$.

Finally, we must show that the right hand side summation in Equation (3.8) counts the paths in $\mathbf{S}'$ once and only once. Assume that there is a path $P \in \mathbf{S}'$ that is counted more than once (at least twice) as $P = P_A \cdot P_B = P_C \cdot P_D$, where $P_A, P_C \in \mathbf{S}_1$ and $P_B, P_D \in \mathbf{S}_2$. We have $P_A \neq P_C$ and $P_B \neq P_D$. This is because $P_A \cdot P_B$ and $P_C \cdot P_D$ both constitute the same path (i.e., the same sequence of nodes); hence either of the equalities $P_A = P_C$ or $P_B = P_D$ would immediately result in the other one being true as well. Therefore, the equality $P_A \cdot P_B = P_C \cdot P_D$ means that either P_A has to be a prefix of P_C or the other way around. Also since both paths P_A and P_C are in $\mathbf{S}_1$, they both end at u. This indicates that either of P_A or P_C has to visit u more than once so that $P_A \neq P_C$ can hold, which is in contradiction with the definition of $\mathbf{S}_1$. Hence, Equation (3.8) holds. $\square$

Lemma 3. *The sum of the contributions of all paths in* $\mathbf{S}_1$*, which is the set of all paths from* x *to* u *that visit* u *only once, can be calculated as:*

$$\sum_{P \in \mathbf{S}_1} PathCont(P, u) = \frac{SPcont(u, x)}{SPcont(u, u)} \cdot \alpha. \tag{3.9}$$

Proof. Let $\mathbf{S}_{x \to u}$ denote the set of all paths from x to u (can visit u any number of times), and $\mathbf{S}_{u \to u}$ the set of all paths starting at u and ending at u itself (including the path of length 0 at u). We have:

$$\mathbf{S}_{x \to u} = \{ \ P_1 \cdot P_2 \mid P_1 \in \mathbf{S}_1, P_2 \in \mathbf{S}_{u \to u} \ \}.$$

Therefore, the sum of the contributions of all paths in $\mathbf{S}_{x \to u}$ can be calculated as:

$$\begin{aligned}
SPcont(u, x) &= \sum_{P \in \mathbf{S}_{x \to u}} PathCont(P, v) \\
&= \sum_{P_1 \in \mathbf{S}_1} \sum_{P_2 \in \mathbf{S}_{u \to u}} PathCont(P_1 \cdot P_2, u) \\
&= \sum_{P_1 \in \mathbf{S_1}} \sum_{P_2 \in \mathbf{S_{u \to u}}} PathCont(P_1, u) \cdot \frac{1}{\alpha} \cdot PathCont(P_2, u) \\
&= \sum_{P_1 \in \mathbf{S}_1} PathCont(P_1, u) \cdot \frac{1}{\alpha} \cdot SPcont(u, u),
\end{aligned}$$

where the equality $PathCont(P_1 \cdot P_2, u) = PathCont(P_1, u) \cdot \frac{1}{\alpha} \cdot PathCont(P_2, u)$ comes from the definition of path contribution: $PathCont(u_0 \to u_1 \to \cdots \to u_l \to u, u) =$

$\alpha \prod_{i=1}^{l} \dfrac{1-\alpha}{D_o(u_i)}$. From the above equation it simply follows that:

$$\sum_{P \in \mathbf{S}_1} \mathit{PathCont}(P, u) = \frac{\mathit{SPcont}(u, x)}{\mathit{SPcont}(u, u)} \cdot \alpha.$$

□

Using the above lemmas, we can prove the following theorem.

Theorem 1. *The sum of the contributions of all paths from x to v that go through u ($u \neq v$), i.e., all paths in $\mathbf{S}$, can be calculated as:*

$$\sum_{P \in \mathbf{S}} \mathit{PathCont}(P, v) = \frac{\mathit{SPcont}(u, x)}{\mathit{SPcont}(u, u)} \cdot \mathit{SPcont}(v, u)$$

Proof. This is easy to verify using Lemmas 2 and 3.

$$\begin{aligned}
\sum_{P \in \mathbf{S}} \mathit{PathCont}(P, v) &= \sum_{P_1 \in \mathbf{S}_1} \sum_{P_2 \in \mathbf{S}_2} \mathit{PathCont}(P_1 \cdot P_2, v) \\
&= \sum_{P_1 \in \mathbf{S}_1} \sum_{P_2 \in \mathbf{S}_2} \mathit{PathCont}(P_1, u) \cdot \alpha \cdot \mathit{PathCont}(P_2, v) \\
&= \sum_{P_1 \in \mathbf{S}_1} \mathit{PathCont}(P_1, u) \cdot \alpha \cdot \mathit{SPcont}(v, u) \\
&= \frac{\mathit{SPcont}(u, x)}{\mathit{SPcont}(u, u)} \cdot \mathit{SPcont}(v, u) \qquad (3.10)
\end{aligned}$$

□

Finally, we can rewrite Equation 3.6 using $\mathit{SPcont}(\cdot)$ and PageRank.

Theorem 2. *The page contribution of node u to node v equals:*

$$\mathit{PageCont}(v, u) = \mathit{SPcont}(v, u) \cdot \left(\frac{\mathit{PageRank}(u)}{\mathit{SPcont}(u, u)} \right) \qquad (3.11)$$

Proof. Using Lemma 1 and 3, and Theorem 1, we have:

$$\begin{aligned}
\mathit{PageCont}(v,u) &= \sum_{\text{path } P \text{ from } u \text{ to } v} \mathit{PathCont}(P,v) + \\
&\quad \sum_{x \in V-\{u\}} \; \sum_{\text{path } P \text{ from } x \text{ to } v \text{ through u}} \mathit{PathCont}(P,v) \\
&= \mathit{SPcont}(v,u) + \sum_{x \in V-\{u\}} \frac{\mathit{SPcont}(u,x)}{\mathit{SPcont}(u,u)} \cdot \mathit{SPcont}(v,u) \\
&= \mathit{SPcont}(v,u) \cdot \Big(1 + \frac{1}{\mathit{SPcont}(u,u)} \sum_{x \in V-\{u\}} \mathit{SPcont}(u,x)\Big) \\
&= \mathit{SPcont}(v,u) \cdot \Big(\frac{\sum_{x \in V} \mathit{SPcont}(u,x)}{\mathit{SPcont}(u,u))}\Big) \\
&= \mathit{SPcont}(v,u) \cdot \Big(\frac{\mathit{PageRank}(u)}{\mathit{SPcont}(u,u)}\Big)
\end{aligned}$$

And the last equality holds based on Equation 3.5. □

3.3 Calculation of Page Contribution Using Matrix Equations

In this section, we develop a method to calculate path and page contributions based on the stochastic matrix of the graph.

Theorem 3. *Let L be the stochastic matrix of a graph $G = (V, E)$. Matrix M such that $M[v, u]$ represents the sum of the contributions of all paths from node u to node v, i.e., $M[v, u] = \mathit{SPcont}(v, u)$, can be calculated as follows:*

$$M = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t \cdot L^t. \tag{3.12}$$

Proof. In order to obtain matrix M, we note that according to the definition of path contribution (Equation (3.3)), the sum of the path contributions from node u to node v (i.e., $M[v, u]$) is equal to the sum of the path contribution of u's outgoing neighbors $x \in O(u)$ to v (i.e., $M[v, x]$), times the probability of going from u to each of these neighbors (i.e.,

$\frac{1-\alpha}{ODeg(u)}$). More specifically, the following equations hold for all entries of M:

$$M[v,u] = \sum_{x \in\ O(u)} M[v,x] \frac{1-\alpha}{ODeg(u)} \qquad \text{if } \ u \neq v \tag{3.13}$$

$$M[v,u] = \sum_{x \in\ O(u)} M[v,x] \frac{1-\alpha}{ODeg(u)} + \alpha \quad \text{if } \ v = u \tag{3.14}$$

The difference between Equation (3.13) and Equation (3.14) is that when $u = v$, we are interested in all paths starting from v and ending on itself, including the path of length 0, which is not captured in Equation (3.13)—this is the probability of starting at v (which is 1) and immediately staying at v forever (with the probability of α).

Equations (3.13) and (3.14) can be summarized as:

$$M = (1-\alpha) \cdot ML + A, \tag{3.15}$$

where A is a scalar matrix with a scalar factor α. Inserting the power series of Equation (3.12) into the right hand side of Equation (3.15) leads to:

$$\begin{aligned} A \cdot (1-\alpha) \sum_{t=0}^{\infty} (1-\alpha)^t \cdot L^t L + A &= A\Big((1-\alpha) \sum_{t=0}^{\infty} (1-\alpha)^t \cdot L^t L + I\Big) & (3.16) \\ &= A\Big(\sum_{t=1}^{\infty} (1-\alpha)^t \cdot L^t + (1-\alpha)^0 \cdot L^0\Big) \\ &= \alpha \sum_{t=0}^{\infty} (1-\alpha)^t \cdot L^t \\ & & (3.17) \end{aligned}$$

Hence, the power series is a solution to Equation (3.15). We now show that this power series is the only solution. Assume that this is not the case, and M_1 and M_2 are two solutions of Equation (3.15):

$$\begin{aligned} & M_1 = (1-\alpha) \cdot M_1 L + A \\ & M_2 = (1-\alpha) \cdot M_2 L + A \\ & \Rightarrow M_1 - M_2 = (1-\alpha) \cdot (M_1 - M_2) L. \end{aligned} \tag{3.18}$$

For simplicity, let $M_1 - M_2 = K$, $K[i,j] = k_{ij}$, and $L[i,j] = l_{ij}$. Since $K \neq 0$, we select one of its non-zero rows, assume it is row i, and we choose the entry in row i which has the largest absolute value, denoted by k_{ij}:

$$\begin{aligned}
|k_{ij}| &= (1-\alpha)\cdot(|k_{i1}\cdot l_{1j} + k_{i2}\cdot l_{2j} + \cdots + k_{in}\cdot l_{nj}|) \\
&\le (1-\alpha)\cdot(|k_{i1}\cdot l_{1j}| + |k_{i2}\cdot l_{2j}| + \cdots + |k_{in}\cdot l_{nj}|) \\
&\le (1-\alpha)\cdot(|k_{ij}\cdot l_{1j}| + |k_{ij}\cdot l_{2j}| + \cdots + |k_{ij}\cdot l_{nj}|) \\
&= (1-\alpha)\cdot(|k_{ij}|\cdot|l_{1j}| + |k_{ij}|\cdot|l_{2j}| + \cdots + |k_{ij}|\cdot|l_{nj}|) \\
&= (1-\alpha)\cdot(|k_{ij}|\cdot(|l_{1j}| + |l_{2j}| + \cdots + |l_{nj}|)) \\
&= (1-\alpha)\cdot|k_{ij}| \qquad (3.19)
\end{aligned}$$

We have $|k_{ij}| = (1-\alpha)\cdot|k_{ij}|$ which is a contradiction for $1-\alpha \neq 1$. Hence, $M = \alpha\sum_{t=0}^{\infty}(1-\alpha)^t\cdot L^t$ is the only solution of Equation (3.15). It just remains to prove the convergence of the series which we do as follows. □

Lemma 4. *All powers of a left stochastic matrix L are left stochastic matrices.*

Proof. This can be simply proven by induction. For $t = 1$, $L^t = L$ is a stochastic matrix by assumption. We show that for every $t > 0$, if L^t is an stochastic matrix, L^{t+1} is stochastic as well. The entries of an stochastic matrix are non negative by definition, then for each column c of L^{t+1}, the sum of all entries, which we need to show is equal to 1, is calculated as:

$$\sum_{i=1}^{N} L^{t+1}[i,c] = \sum_{i=1}^{N}\sum_{j=1}^{N} L^t[i,j]L[j,c] = \sum_{j=1}^{N}\Big(L[j,c]\sum_{i=1}^{N} L^t[i,j]\Big) = \sum_{j=1}^{N} L[j,c] = 1.$$

□

Lemma 5. *The power series in Equation (3.12) converges for any stochastic matrix L.*

Proof. Let $\mathbf{1}_{N\times N}$ denote an $N\times N$ matrix whose entries are all 1, and the relation $\le$ between two matrices as $A \le B$ stand for $A[i,j] \le B[i,j]$ for all $1 \le i,j \le N$. By Lemma 4 we have:

$$L^t \le \mathbf{1}_{N\times N}\ (t \ge 0) \Rightarrow \alpha\sum_{t=0}^{\infty}(1-\alpha)^t\cdot L^t \le \alpha\sum_{t=0}^{\infty}((1-\alpha)^t\cdot\mathbf{1}_{N\times N}) \le \mathbf{1}_{N\times N}\cdot\frac{1}{\alpha}.$$

Also, since $(1-\alpha)\cdot L \ge 0$, the series is ascending, and the lemma is proved. □

Having calculated matrix M using Theorem 3, the next and last step to formulate the page contribution value by a matrix equation is simply substituting *SPcont*(.) values by the entries of M.

Corollary 1. *The page contribution of node u to node v ($u \neq v$), can be formulated by a matrix equation as follows. $PageCont(v, v) = \alpha$ by definition.*

$$PageCont(v, u) = PageRank(u) \cdot \frac{M[v, u]}{M[u, u]}, \tag{3.20}$$

where matrix M is defined in Theorem 3.

Proof. Since $SPcont(v, u) = M[v, u]$ from Theorem 3, it is enough to substitute $SPcont(v, u)$ and $SPcont(u, u)$ in Theorem 2 with $M[v, u]$ and $M[u, u]$. $\square$

The complexity of calculating page contribution for every pair of nodes using matrix multiplications is of $O(K \times N^3)$, where K is the number of iterations until convergence. For example, if in the implementation of Equation 3.12 we consider convergence as when the values added to the series are all smaller than a threshold ϵ, we have $K = O(\log_{(1-\alpha)} \epsilon)$; this is because the value added to each cell of matrix M in iteration t is no larger than $(1 - \alpha)^t$. Moreover, by using Strassen's algorithm for matrix multiplications, this complexity can be reduced to $O(\log_{(1-\alpha)} \epsilon \times N^{2.8})$.

Chapter 4

Local Calculation of Page Contribution Vector

As we have shown in the previous chapter, the page contribution of node u to node v can be calculated as $\mathit{PageCont}(v,u) = \mathit{SPcont}(v,u) \cdot \mathit{PageRank}(u)/\mathit{SPcont}(u,u)$, where $\mathit{SPcont}(v,u)$ is the sum of the contribution of all paths from u to v. We would like to find a vector that represents the page contribution of all nodes of the graph to a particular target node. The exact value of such page contribution vector can be obtained using the matrix calculations in the previous chapter. However, it needs repeated multiplications of the stochastic matrix of the whole graph in itself, even if we are looking for the vector only for one target node (i.e., one row of the page contribution matrix), which is very inefficient and even infeasible when we are working with large datasets. In this chapter, we first formally define the page contribution vector in 4.1, and then present an efficient local algorithm to approximate it.

4.1 Page Contribution Vector

Let $\mathbf{cv}_v$ be a row vector where $\mathbf{cv}_v(u) = \mathit{SPcont}(v,u)$. We can obtain $\mathbf{cv}_v$ easily using Theorem 3 as follows:

$$\mathbf{cv}_v = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t \cdot L^t \mathbf{e}_v, \tag{4.1}$$

where L is the stochastic matrix of the graph, α is the random jump probability, and $\mathbf{e}_v$ is the row unit vector whose v-th entry is equal to 1.

Based on the path contribution vector $\mathbf{cv}_v$, we are going to find a page contribution vector $\mathbf{gcv}_v$, where $\mathbf{gcv}_v(u) = \mathit{PageCont}(v, u)$. Let $\mathbf{pr}$ be a pagerank vector so that $\mathbf{pr}(u) = \mathit{PageRank}(u)$, then based on Theorem 2 and using the path contribution vector we have:

$$\mathbf{gcv}_v(u) = \mathbf{pr}(u) \cdot \mathbf{cv}_v(u)/\mathbf{cv}_u(u) \tag{4.2}$$

4.2 Approximation Algorithms

Our goal is to find a local algorithm to approximate page contribution vector. A local algorithm for approximating a contribution vector has been proposed by Andersen et al. in [1] as discussed in Section 2, however it only approximates the path contribution vector. Based on this algorithm, we present a new algorithm to approximate the page contribution vector.

Following Equation (4.2), our algorithm consists of two steps. In the first step, we follow the same idea as in Andersen's algorithm with a modification that we add in PageRank scores into the computation of path contribution vectors. In this way, the search algorithm selects nodes with higher pagerank scores, resulting in more efficient discovery of most contributing nodes. In the second step, we refine the obtained vector by removing the self-contribution factor from page contributions as in Equation (4.2).

4.2.1 Step 1 - Approximation of Augmented Path Contribution Vector

In this step, we are going to approximate the augmented path contribution vector whose u-th entry is equal to the path contribution of node u to a given target node, multiplied by the pagerank of node u. This is formally defined as follows.

$$\mathbf{cpv}_v := \mathbf{cv}_v \Box \mathbf{pr},$$

where $\Box$ for two vectors $v = (v_1, v_2, \dots, v_n)$ and $u = (u_1, u_2, \dots, u_n)$ is defined as follows:

$$v \Box u = (v_1.u_1, v_2.u_2, \dots v_n.u_n)$$

Our algorithm for approximating the augmented path contribution vector is shown in Fig. 4.1. The input to this algorithm consists of the target node v, the random jump probability α ($0 < \alpha < 1$), and the desired error bound ϵ ($0 < \epsilon < 1$). The algorithm computes a vector $\hat{\mathbf{cpv}}_v$ which is an approximation of $\mathbf{cpv}_v$.

Approximation of Augmented Path Contribution

```
ApproxAugPathCont ($v$, $\alpha$, $\epsilon$)
// $v$: target node; $\epsilon$: max absolute error
// $\alpha$: random jump probability (i.e., 1 - damping factor)
1. $\mathbf{p} = \mathbf{0}$, $\mathbf{r} = \mathbf{e}_v$
2. while $\mathbf{r}(u) \cdot \mathbf{pr}(u) \geq \epsilon$ for some node $u$ do
3.    Pick a node $u$ where $\mathbf{r}(u) \cdot \mathbf{pr}(u) \geq \epsilon$
4.    pushback($u$, $\mathbf{p}$, $\mathbf{r}$)
5. done
6. return $\mathbf{p}$
pushback ($u$, $\mathbf{p}$, $\mathbf{r}$)
1. $\mathbf{p}(u) \leftarrow \mathbf{p}(u) + \alpha \cdot \mathbf{r}(u) \cdot \mathbf{pr}(u)$
2. $r_0 \leftarrow \mathbf{r}(u)$; $\mathbf{r}(u) \leftarrow 0$
3. for $w \in I(u)$ do
4.    $\mathbf{r}(w) \leftarrow \mathbf{r}(w) + (1 - \alpha) \cdot r_0 / ODeg(w)$
5. done
```

Figure 4.1: The proposed algorithm for approximating the augmented path contribution vector.

The algorithm keeps two nonnegative vectors $\mathbf{p}$ and $\mathbf{r}$, starting with $\mathbf{p} = \mathbf{0}$ and $\mathbf{r} = \mathbf{e}_v$. The key idea is to maintain the invariant $\mathbf{p} + \mathbf{cpv_r} = \mathbf{cpv}_v$ throughout the procedure. The algorithm applies a series of pushback operations (starting from the target node v). Each pushback operation takes one node u of the graph at a time with $\mathbf{r}(u)\mathbf{pr}(u) \geq \epsilon$, and increases $\|\mathbf{p}\|_1$ by moving an α fraction of $\mathbf{r}(u)\mathbf{pr}(u)$ to $\mathbf{p}(u)$. This is done such that the invariant $\mathbf{p} + \mathbf{cpv_r} = \mathbf{cpv}_v$ is maintained. The algorithm keeps performing the series of pushback operations until all entries of the vector $\mathbf{r}$ become less than ϵ. Then, according to the invariant, $\mathbf{p}$ will contain an ϵ-approximation of $\mathbf{cpv}_v$ as proven shortly.

The fact that the equation $\mathbf{p} + \mathbf{cpv_r} = \mathbf{cpv}_v$ holds throughout the algorithm for all pushback operations is the key point for proving the correctness of the algorithm. Before proceeding with a proof of this equation, we first need to establish a number of preliminaries.

Using Equation 4.1, we can define the path contribution vector for a specified subset S of vertices as follows:

$$\mathbf{cv}_S = \sum_{v \in S} \mathbf{cv}_v$$

Let $\mathbf{cv}_S = \sum_{v \in S} \cdot \mathbf{e_v}$. Then,

$$\mathbf{cv}_S = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t \cdot L^t \cdot \mathbf{e}_S$$

For the sake of convenience, for any non-negative vector $\mathbf{s}$, we define

$$\mathbf{cv_s} = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t \cdot L^t \cdot \mathbf{s} \tag{4.3}$$

We can now prove the following equations using Equation 4.3. For any vector $\mathbf{s}$ we have:

$$\mathbf{cv_s} \cdot L = \mathbf{cv}_{\mathbf{s}L} \tag{4.4}$$

$$\mathbf{cv_s} = \alpha \cdot \mathbf{s} + (1-\alpha) \cdot \mathbf{cv}_{\mathbf{s}L} \tag{4.5}$$

Lemma 6. *Let $\mathbf{p}'$ and $\mathbf{r}'$ be the result of performing pushback(u) on $\mathbf{p}$ and $\mathbf{r}$. If $\mathbf{p}$ and $\mathbf{r}$ satisfy the invariant $\mathbf{p} + \mathbf{cpv_r} = \mathbf{cpv}_v$, then $\mathbf{p}'$ and $\mathbf{r}'$ satisfy the invariant $\mathbf{p}' + \mathbf{cpv}_{\mathbf{r}'} = \mathbf{cpv}_v$.*

Proof. After the pushback operation, we have:

$$\begin{aligned} \mathbf{p}' &= \mathbf{p} + \alpha \cdot \mathbf{r}(u) \cdot \mathbf{pr}(u) \cdot \mathbf{e}_u \\ \mathbf{r}' &= \mathbf{r} - \mathbf{r}(u) \cdot \mathbf{e}_u + (1-\alpha) \cdot \mathbf{r}(u) \cdot \mathbf{e}_u \cdot L \end{aligned} \tag{4.6}$$

We are going to show that $\mathbf{p} + \mathbf{cpv_r} = \mathbf{p}' + \mathbf{cpv}_{\mathbf{r}'}$. Using Equation 4.5 we have:

$$\begin{aligned} \mathbf{cpv_r} &= \mathbf{cpv}_{(\mathbf{r}-\mathbf{r}(u)\cdot\mathbf{e}_u)} + \mathbf{cpv}_{\mathbf{r}(u)\cdot\mathbf{e}_u} \\ &= \mathbf{cpv}_{(\mathbf{r}-\mathbf{r}(u)\cdot\mathbf{e}_u)} + \alpha \cdot \mathbf{r}(u) \cdot \mathbf{e}_u + \mathbf{cpv}_{((1-\alpha)\cdot\mathbf{r}(u)\cdot\mathbf{e}_u\cdot L)} \\ &= \mathbf{cpv}_{(\mathbf{r}-\mathbf{r}(u)\cdot\mathbf{e}_u+(1-\alpha)\cdot\mathbf{r}(u)\cdot\mathbf{e}_u\cdot L)} + \alpha \cdot \mathbf{r}(u) \cdot \mathbf{e}_u \\ &= \mathbf{cpv}_{\mathbf{r}'} + \alpha \cdot \mathbf{r}(u) \cdot \mathbf{e}_u \\ &\Rightarrow \mathbf{cpv_r} \square \mathbf{pr} = \mathbf{cpv}_{\mathbf{r}'} \square \mathbf{pr} + \alpha \cdot \mathbf{r}(u) \cdot \mathbf{e}_u \square \mathbf{pr} \\ &\Rightarrow \mathbf{cpv_r} = \mathbf{cpv}_{\mathbf{r}'} + \mathbf{p}' - \mathbf{p} \end{aligned} \tag{4.7}$$

□

We are now ready to present our approximation algorithm in full details, and analyze its running time. We describe the running time of our algorithm based on the number of pushback operations; we guarantee an upper bound on the total number of pushback operations performed by the algorithm. The running time for each pushback operation is also proportional to the incomming degree of the node which is being pushbacked.

Theorem 4. *The algorithm ApproxAugPathCont(v, α, ϵ) returns a vector* $\hat{\mathbf{cpv}}_v$ *such that* $\hat{\mathbf{cpv}}_v \geq 0$ *and for every vertex u we have:*

$$\mathbf{cpv}_v(u) - \epsilon \leq \hat{\mathbf{cpv}}_v \leq \mathbf{cpv}_v(u)$$

Let T be the total number of pushback operations performed by the algorithm, then:

$$T \leq \frac{\|\mathbf{cpv}_v\|_1}{\alpha\epsilon} + 1, \tag{4.8}$$

where for any vector $X = (x_1, \cdots, x_n)$, $\|X\|_1 = \sum_{i=1}^n |x_i|$.

Proof. Let $\mathbf{p}_t$ and $\mathbf{r}_t$ be the states of the vectors $\mathbf{p}$ and $\mathbf{r}$ after t pushes. The algorithm starts with $\mathbf{p}_0 = 0$ and $\mathbf{r}_0 = \mathbf{e}_v$ which satisfies the invariant $\mathbf{p}_0 + \mathbf{cpv}_{\mathbf{r}_0} = \mathbf{cpv}_v$, and also using Lemma 6, we know that $\mathbf{p}_t + \mathbf{cpv}_{\mathbf{r}_t} = \mathbf{cpv}_v$ for every $t \geq 1$. Thus $\mathbf{cpv}_{\mathbf{r}_t}$ is the error term, and since $\mathbf{r}_t$ is nonnegative at each step the error is also nonnegative. When the algorithm terminates, for every vertex u we must have $\mathbf{r}_T(u)\mathbf{pr}(u) \leq \epsilon$. Also for every vertex u we have:

$$\mathbf{cpv}_{\mathbf{r}_T}(u) \leq \mathbf{r}_T(u) \cdot \mathbf{pr}(u) \leq \epsilon.$$

This is true because $\mathbf{r}_T$ is nonnegative, $\mathbf{cpv}_{\mathbf{r}_T}(u) = \mathbf{cv}_{\mathbf{r}_T}(u) \cdot \mathbf{pr}(u)$, and $\mathbf{cv}_{\mathbf{r}_T}(u) \leq \mathbf{r}_T(u)$ since each column of L sums to 1. Also each pushback operation increases $\|\mathbf{p}\|_1$ by at least $\alpha\epsilon$, so we have:

$$\begin{aligned} &\alpha\epsilon(T-1) \leq \|\mathbf{p}_{T-1}\|_1 \leq \|\mathbf{cpv}_v\|_1 \\ &\Rightarrow T \leq \frac{\|\mathbf{cpv}_v\|_1}{\alpha\epsilon} + 1 \end{aligned} \tag{4.9}$$

□

We also note that the nodes visited are at most at a distance of $\lceil \log_{(1-\alpha)}(\epsilon/\mathbf{pr}_{max}) \rceil$ from the target node, where $\mathbf{pr}_{max} = \max_u \mathbf{pr}(u)$. This is because a term of $(1-\alpha)/ODeg(w)$ (i.e., at most $1-\alpha$) is multiplied in the $\mathbf{r}(\cdot)$ value of nodes as we get far from the target node (see Line 4 of the **pushback** function in the pseudocode of Figure 4.1). Nevertheless, this is only a loose upper bound on the distance (tens to hundreds in our experiments) which the visited nodes most often do not reach; they were usually within a few hops from the target node in our experiments.

4.2.2 Step 2 - The Approximation of Page Contribution (APC) Algorithm

In this step, we complete the calculation of the page contribution vector by adding in the self contribution factor $\mathbf{cv}_u(u)$ for each node. We present two alternatives for this purpose. First, a straightforward way is to estimate $\mathbf{cv}_u$ using Andersen's algorithm in [1]. We show the approximation error of this method in Theorem 5. In addition, we obtain an upper and lower bound for $\mathbf{cv}_u(u)$, and show that the page contribution vector can be approximated with reasonable accuracy by just substituting $\mathbf{cv}_u(u)$ values with a fixed constant.

In order to prove an upper bounds for error of our algorithm, we first need to see the following lemma.

Lemma 7. *Let v and u be two nodes in graph $G = (V, E)$. Then:*

$$\begin{cases} 0 \le \mathbf{cv}_v(u) \le 1-\alpha & (u \ne v) \\ \alpha \le \mathbf{cv}_v(u) \le 1 & (u = v) \end{cases} \tag{4.10}$$

Proof. Recall that $M = \alpha \sum_{t=0}^{\infty}(1-\alpha)^t \cdot L^t$ where L is the stochastic matrix of the graph, and $M[v,u] = \mathbf{cv}_v(u)$. Also recall that we have proven that all powers of a stochastic matrix are stochastic, hence no entry of matrix L^t can be greater than 1. Then:

$$(\text{for any x} \in V)\ M[v,x] = \alpha \sum_{t=0}^{\infty}(1-\alpha)^t \cdot L^t[v,x] \le \alpha \sum_{t=0}^{\infty}(1-\alpha)^t = 1$$

Also, note that the minimum path contribution of one node to itself ($\mathbf{cv}_v(v)$) is the random jump probability α by definition. Thus the lemma is proved for $u = v$.

Then for $u \ne v$ we have:

$$\begin{aligned} \mathbf{cv}_v(u) &= \sum_{x \in O(u)} \Big(\frac{1-\alpha}{ODeg} \cdot \mathbf{cv}_v(x)\Big) \\ &\le ODeg(u) \cdot \max_{x \in O(u)} \Big\{\frac{1-\alpha}{ODeg(u)} \cdot M[v,x]\Big\} \\ &\le 1-\alpha \end{aligned}$$

Moreover, obviously the minimum path contribution of node u to v when $(u \neq v)$ is 0, since it might not exist any sequence of links from node u to v. □

Theorem 5. *Let ϵ be the absolute error of approximating $\mathbf{cpv}_v$ using The Approximation of Page Contribution (**APC**) algorithm, and also the absolute error of approximating $\mathbf{cv}_u$ based on Andersen's algorithm in [1]. Then the absolute approximation error of approximating $\mathbf{gcv}_v(u)$, is* $\max\{\frac{\epsilon}{\alpha},\ \mathbf{pr}(u)\frac{\epsilon(1-\alpha)}{\alpha\ (\alpha-\epsilon)}\}$.

Proof. According to the definition, the u-th entry of the page contribution vector is

$$\mathbf{gcv}_v(u) = \frac{\mathbf{cpv}_v(u)}{\mathbf{cv}_u(u)}$$

Let $\hat{\mathbf{cv}}_u(u)$ denote the estimate value for $\mathbf{cv}_u(u)$ using [1], then:

$$\begin{aligned} &\mathbf{cv}_u(u) - \epsilon \le \hat{\mathbf{cv}}_u(u) \le \mathbf{cv}_u(u) \\ &\Rightarrow (1-\frac{\epsilon}{\alpha}) \cdot \mathbf{cv}_u(u) \le \hat{\mathbf{cv}}_u(u) \le \mathbf{cv}_u(u) \end{aligned} \tag{4.11}$$

Also based on the Theorem 4 we have:

$$\mathbf{cpv}_v(u) - \epsilon \le \hat{\mathbf{cpv}}_v(u) \le \mathbf{cpv}_v(u)$$

Then first:

$$\begin{aligned} \frac{\mathbf{cpv}_v(u)}{\mathbf{cv}_u(u)} - \frac{\hat{\mathbf{cpv}}_v(u)}{\hat{\mathbf{cv}}_u(u)} &\le \frac{\mathbf{cpv}_v(u)}{\mathbf{cv}_u(u)} - \frac{\mathbf{cpv}_v(u)-\epsilon}{\mathbf{cv}_u(u)} \\ &\le \frac{\epsilon}{\mathbf{cv}_u(u)} \\ &\le \frac{\epsilon}{\alpha} \end{aligned} \tag{4.12}$$

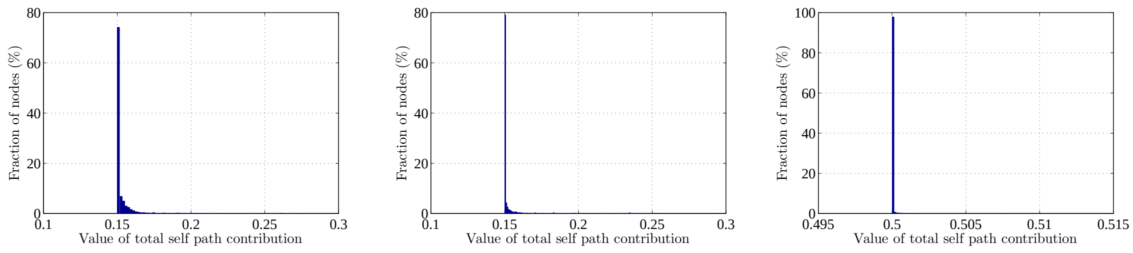
And the last inequality holds, because the minimum amount of the self contribution is α by definition.

Second:

$$\begin{aligned} \frac{\hat{\mathbf{cpv}}_v(u)}{\hat{\mathbf{cv}}_u(u)} - \frac{\mathbf{cpv}_v(u)}{\mathbf{cv}_u(u)} &\le \frac{\mathbf{cpv}_v(u)}{(1-\frac{\epsilon}{\alpha})\cdot \mathbf{cv}_u(u)} - \frac{\mathbf{cpv}_v(u)}{\mathbf{cv}_u(u)} \\ &\le \mathbf{pr}(u) \cdot \frac{\epsilon(1-\alpha)}{\alpha\ (\alpha-\epsilon)} \end{aligned} \tag{4.13}$$

And again the last inequality holds because based on Lemma 7, maximum value of $\mathbf{cpv}_v(u)$ is $\mathbf{pr}(u) \cdot (1-\alpha)$.

Thus, the total absolute error is $\max\{\frac{\epsilon}{\alpha},\ \mathbf{pr}(u) \cdot \frac{\epsilon(1-\alpha)}{\alpha\ (\alpha-\epsilon)}\}$. □



(a) On the web graph ($\alpha = 0.15$). (b) On the host graph ($\alpha = 0.15$). (c) On the citation graph ($\alpha = 0.5$).

Figure 4.2: Distribution of $\mathbf{cv}_u(u)$ values of different data sets measured on 10,000 nodes (uniformly selected from the ranked sequence of all nodes). $\alpha = 0.15, 0.5$ on the web/host and citation graph, respectively.

4.2.3 The Fast Approximation of Page Contribution (FAPC) Algorithm

Since $\mathbf{gcv}_v(u) = \mathbf{cpv}_v(u)/\mathbf{cv}_u(u)$, using APC algorithm, we need to apply the Augmented Path Contribution algorithm to obtain all the necessary $\mathbf{cpv}_v(u)$ values, and then apply Andersen's algorithm [1] several times to find $\mathbf{cv}_u(u)$ for each contributor node u individually. Although the approximation error of APC algorithm is small (as proven in the previous subsection and confirmed in our evaluations), it can take a long time to complete especially on large graphs. This is because for each nonzero element of $\mathbf{cpv}_v$ Andersen's algorithm must be run.

However, we have observed that in all datasets we have looked at (web, host, and citation graphs), the $\mathbf{cv}_u(u)$ value (i.e., the self path contribution) is around a constant value α, for the vast majority of nodes. In other words, there are not so many loops with significant path contributions from node u to itself, and the main factor in $\mathbf{cv}_u(u)$ is just a contribution value of α corresponding to the 0-length path on node u.

This observation is illustrated in Figure 4.2, which plots the distribution of $\mathbf{cv}_u(u)$ values for the web, host and citation graphs datasets; the description of the datasets is given in details in the next section. The $\mathbf{cv}_u(u)$ values in this figure are found using Andersen's algorithm [1] with $\epsilon = 10^{-6}$, i.e., an absolute error of at most 10^{-6} which is negligible. The values are calculated on 10,000 nodes from each data set, which are picked with equal distances ($N/10000$) from the ranked (based on PageRank) sequence of all nodes; this is to ensure that representative nodes from a wide range of PageRank scores are included. As Figure 4.2 confirms, the vast majority of nodes have a self path contribution close to α. In particular, in the web graph data set the $\mathbf{cv}_u(u)$ value equals α (0.15) for 58% of the nodes, it is between 0.15 and 0.16 for 93% of the nodes, and less than 2% of the nodes have a $\mathbf{cv}_u(u)$ value greater than 0.20. These percentages are 70%, 94%, and again less than 2% for the host graph. In the citation graph, as expected the self path contribution factor is smaller since cycles are very unlikely. In particular, less than 0.02% of the nodes have a $\mathbf{cv}_u(u)$ value greater than 0.51 (α =0.5 for the citation graph).

Therefore, we can significantly reduce the running time of our APC algorithm by skipping the calculation of $\mathbf{cv}_u(u)$ values and just assume a fixed value of $\mathbf{cv}_u(u) = \alpha$, without introducing much error. We call this version of the APC algorithm the Fast APC (**FAPC**) algorithm. In fact, the running time of FAPC algorithm is same as the running time for approximation of augmented path contribution vector. We now provide some upper bounds for the error of this approximation algorithm.

Theorem 6. *By assigning $\mathbf{cv}_u(u) = \alpha$ (for every $u \in V(G)$), the approximation error of the FAPC algorithm is at most* $\max\{\frac{\epsilon}{\alpha}, \mathbf{pr}(u) \cdot \frac{(1-\alpha)^2}{\alpha}\}$.

Proof. Let $\hat{\mathbf{cpv}}_v(u)$ denote the estimate value for $\mathbf{cpv}_v(u)$. The value approximated for $\mathbf{gcv}_v(u) = \mathbf{cpv}_v(u)/\mathbf{cv}_u(u)$ by the FAPC algorithm is $\hat{\mathbf{cpv}}_v(u)/\alpha$. We have:

$$\frac{\mathbf{cpv}_v(u)}{\mathbf{cv}_u(u)} - \frac{\hat{\mathbf{cpv}}_v(u)}{\alpha} \le \frac{\mathbf{cpv}_v(u)}{\alpha} - \frac{\hat{\mathbf{cpv}}_v(u)}{\alpha} \le \frac{\epsilon}{\alpha}.$$

$$\frac{\hat{\mathbf{cpv}}_v(u)}{\alpha} - \frac{\mathbf{cpv}_v(u)}{\mathbf{cv}_u(u)} \le \mathbf{cpv}_v(u) \cdot \Big(\frac{1}{\alpha} - \frac{1}{\mathbf{cv}_u(u)}\Big) \le \mathbf{pr}(u) \cdot (1-\alpha)\Big(\frac{1-\alpha}{\alpha}\Big)$$

The maximum approximation error is therefore: $\max\{\frac{\epsilon}{\alpha}, \mathbf{pr}(u) \cdot \frac{(1-\alpha)^2}{\alpha}\}$. □

Note that α is not the best value for $\mathbf{cv}_u(u)$ to minimize the worst-case approximation bound of the FAPC algorithm. Rather, the choice of $\mathbf{cv}_u(u) = \alpha$ is to minimize the approximation error for nodes whose $\mathbf{cv}_u(u)$ value is close to α, which actually are the vast majority of nodes. To make our study of the FAPC algorithm comprehensive, in the following theorem we also show the best fixed value for $\mathbf{cv}_u(u)$, which can minimize the worst-case approximation error. Nevertheless, in our experiments we use $\mathbf{cv}_u(u) = \alpha$ since it results in much smaller average-case errors (see the observation in Figure 4.2).

Lemma 8. *By assigning a fixed value I_0 ($\alpha \le I_0 \le 1$) to all $\mathbf{cv}_u(u)$ ($u \in V(G)$), the approximation error of the FAPC algorithm is at most:*

$$\max\Big\{\mathbf{pr}(u) \cdot (1-\alpha)\Big(\frac{1}{\alpha} - \frac{1}{I_0}\Big) + \frac{\epsilon}{I_0},\ \mathbf{pr}(u) \cdot (1-\alpha)\Big(\frac{1}{I_0} - 1\Big)\Big\}. \tag{4.14}$$

Proof. Let $\hat{\mathbf{cpv}}_v(u)$ denote the estimate value for $\mathbf{cpv}_v(u)$. The value approximated for $\mathbf{gcv}_v(u) = \mathbf{cpv}_v(u)/\mathbf{cv}_u(u)$ is $\hat{\mathbf{gcv}}_v(u) = \hat{\mathbf{cpv}}_v(u)/I_0$.

$$\begin{gathered}
\mathbf{gcv}_v(u) - \hat{\mathbf{gcv}}_v(u) = \frac{\mathbf{cpv}_v(u)}{\mathbf{cv}_u(u)} - \frac{\hat{\mathbf{cpv}}_v(u)}{I_0} = \\
\frac{(I_0 - \mathbf{cv}_u(u)) \cdot \mathbf{cpv}_v(i) + \mathbf{cv}_u(u) \cdot \mathbf{cpv}_v(u) - \mathbf{cv}_u(u) \cdot \hat{\mathbf{cpv}}_v(u)}{I_0 \cdot \mathbf{cv}_u(u)} = \\
\frac{\mathbf{cpv}_v(u)}{I_0} \cdot \Big(\frac{I_0}{\mathbf{cv}_u(u)} - 1\Big) + \frac{\mathbf{cpv}_v(u) - \hat{\mathbf{cpv}}_v(u)}{I_0}.
\end{gathered} \tag{4.15}$$

Since $0 \le \mathbf{cpv}_v(u) \le 1 - \alpha$ and $\alpha \le \mathbf{cv}_u(u), I_0 \le 1$, using Equation (4.15) we have:

$$\begin{aligned}
\mathbf{gcv}_v(u) - \hat{\mathbf{gcv}}_v(u) &\le \frac{\mathbf{pr}(u) \cdot (1-\alpha)}{I_0}\Big(\frac{I_0}{\alpha} - 1\Big) + \frac{\epsilon}{I_0} \\
&\le \mathbf{pr}(u) \cdot (1-\alpha)\Big(\frac{1}{\alpha} - \frac{1}{I_0}\Big) + \frac{\epsilon}{I_0}.
\end{aligned}$$

Moreover:

$$\begin{aligned}
\hat{\mathbf{gcv}}_v(u) - \mathbf{gcv}_v(u) &= \frac{\mathbf{cpv}_v(u)}{I_0} \cdot \Big(1 - \frac{I_0}{\mathbf{cv}_u(u)}\Big) + \frac{\hat{\mathbf{cpv}}_v(u) - \mathbf{cpv}_v(u)}{I_0} \\
&\le \frac{\mathbf{pr}(u) \cdot (1-\alpha)}{I_0}\Big(1 - \frac{I_0}{1}\Big) + 0 \\
&\le \mathbf{pr}(u) \cdot (1-\alpha)\Big(\frac{1}{I_0} - 1\Big).
\end{aligned}$$

Hence, the worst-case approximation error of the FAPC algorithm where $\mathbf{cv}_u(u) = I_0$ equals: $\max\Big\{\mathbf{pr}(u) \cdot (1-\alpha)\Big(\frac{1}{\alpha} - \frac{1}{I_0}\Big) + \frac{\epsilon}{I_0},\ \mathbf{pr}(u) \cdot (1-\alpha)\Big(\frac{1}{I_0} - 1\Big)\Big\}$.

□

Finally, we find the appropriate value for I_0 that minimizes the worst-case approximation error of $\mathbf{gcv}_v(u)$. First, we note that the second term of the $\max\{\cdot\}$ operator in Equation (4.14) is a decreasing function of I_0, meaning that the highest possible value for I_0 (i.e., $I_0 = 1$) would minimize it. The first term, on the other hand, can be an increasing or decreasing function of I_0 depending on the sign of $\mathbf{pr}(u) \cdot (1-\alpha) - \epsilon$; for some nodes this error is less for lower values of I_0 while for some other nodes it is the opposite.

To find a fixed value for I_0, which minimizes the worst-case approximation error in the general case (independent of u), we denote by P_{max} the maximum PageRank score in the graph, and rewrite the worst-case error in Equation (4.14) as follows:

$$\max\Big\{P_{max}(1-\alpha)\Big(\frac{1}{\alpha}-\frac{1}{I_0}\Big)+\frac{\epsilon}{I_0},\ P_{max}(1-\alpha)\Big(\frac{1}{I_0}-1\Big)\Big\} =$$
$$\max\Big\{\frac{X}{\alpha}-\frac{X-\epsilon}{I_0},\ \frac{X}{I_0}-X\Big\}\ ;\ X=P_{max}(1-\alpha). \tag{4.16}$$

We can now find the best value for I_0 in the general case.

Theorem 7. *Assume $P_{max}(1-\alpha)>\epsilon$. To minimize the worst-case approximation error of the FAPC algorithm, the fixed value I_0 to assign to all $\mathbf{cv}_u(u)$ $(u\in V(G))$ must be $I_0^*=\frac{2\alpha}{1+\alpha}-\frac{\alpha\epsilon}{P_{max}(1-\alpha^2)}$, where $P_{max}=\max_{u\in V(G)}\mathbf{pr}(u)$.*

Remark 1. *We assume $P_{max}(1-\alpha)>\epsilon$ because P_{max} is usually a large number, e.g., $P_{max}=$ 65,172, and 34,000 in the web, and citation graph datasets, respectively.*

Proof. The first term of the max{·} operator in Equation (4.16) is an increasing function of I_0. This means that the smallest possible I_0 value, $I_0=\alpha$, will minimize it. As with the second term of the max{·} operator, on the other hand, the largest possible value $I_0=1$ will minimize it. Therefore, the optimal I_0 value that minimizes the maximum error in Equation (4.16) can be found as:

$$\frac{X}{\alpha}-\frac{X-\epsilon}{I_0}=\frac{X}{I_0}-X$$
$$\Rightarrow\ I_0^*=\frac{2\alpha}{1+\alpha}-\frac{\alpha\epsilon}{P_{max}(1-\alpha^2)}. \tag{4.17}$$

Also the minimum worst-case error for approximating each $\mathbf{gcv}_v(v,u)$ value can be achieved by replacing I_0^* from Equation (4.17) in Equation (4.14).

It is easy to verify that the I_0^* value obtained from Equation (4.17) is within the valid range $\alpha\le I_0^*\le 1$:

$$I_0^*<\frac{2\alpha}{1+\alpha}<1.$$
$$I_0^*=\alpha\frac{2P_{max}(1-\alpha)-\epsilon}{P_{max}(1-\alpha^2)}\ge\alpha\Leftrightarrow 2P_{max}(1-\alpha)-\epsilon\ge P_{max}(1-\alpha^2)$$
$$\Leftrightarrow P_{max}(1-\alpha)^2\ge\epsilon,$$

□

For example, for $\alpha=0.15$ which is the commonly used α value on the web graph, the worst-case approximation error can be minimized by $I_0^*\simeq 0.26$. For $\alpha=0.5$ which is used on the citation graph, we obtain $I_0^*\simeq 0.67$. Nevertheless, as discussed earlier we use $I_0=\alpha$ since it dramatically reduces the average-case error.

4.3 Supporting Sets

A *supporting set* of a target node v is a set U of nodes which make a nonzero contribution to the PageRank score of v. For example, $U = \{v\}$ is a minimum-size supporting set and $U = \{u \mid v \text{ can be reached from } u\}$ is the maximum size supporting set of node v.

The supporting set of interest is generally a set of nodes that contribute the most to v's PageRank score. For example, Andersen et al. [1] consider the supporting set of target node v as the k nodes with highest entries in the $\mathbf{cv}_v$ vector, or the nodes whose sum (i.e., total path contributions) becomes at least a θ fraction of the PageRank score of v. Either k (the supporting set size) or θ (the supporting set's contribution) can be given as input. Then, the other variable, contribution or size, is approximately maximized or minimized respectively.

Another example is the work by Zhou and Pei [31] in which a similar supporting set is defined in the form of a (θ, l)-farm for target node v: the minimum-size set of nodes whose total path contribution to v is at least $\theta \cdot \mathbf{pr}(v)$, and only consists of nodes with distance at most l to node v. A heuristic algorithm is then proposed for finding these farms.

We return the top k nodes with highest entries in the $\mathbf{gcv}_v$ vector as the supporting set of node v; if no k is given, we return all nodes with non-zero entries in $\mathbf{gcv}_v$.

As discussed in Section 1.1, the influence that two nodes u and u' with equal path contributions to v can make to v's PageRank score can be significantly different depending on the PageRank score of u and u' themselves. Therefore, we find page contribution (rather than path contributions) as a better measure for capturing the contributions in the PageRank algorithm. We will assess in Section 5.3 the quality of supporting sets as the page contribution of its nodes; a set of nodes with higher page contributions is a stronger supporter of the target node than a set with lower page contributions.

Chapter 5

Evaluation

In this chapter, we present the results of our experiments with the proposed page contribution algorithms. The setup for our experiments including the datasets that we use are described in Section 5.1. Then, in Section 5.2 we evaluate our algorithms and report the results. Finally, in Section 5.3, we compare our work with two related previous works.

5.1 Setup

Our page contribution algorithm is general and applicable in any context where the PageRank algorithm is used. We select two different categories of graphs to conduct our experiments on: a web and its host graph, and a citation graph.

We use the Webspam-uk2007 dataset [25] from Yahoo! Research for our web graph experiments. This dataset is based on a crawl of the `.uk` domain in May 2007, and consists of two graphs: a web graph of over 105 million nodes (pages) and approximately 3.7 billion edges (hyperlinks), and a host graph of about 115,000 nodes and 1.8 million edges. The host graph is a summary of the web graph in which each *host* (which can include several pages) is represented by a single node; a link exists from host A to host B iff a page on host A links to a page on host B. The Webspam-uk2007 dataset [25] also includes a spam/nonspam label for about 6000 labeled pages.

As the citation graph, we use the High-energy physics theory dataset [20]. The dataset consists of 27,770 papers (nodes) with 352,807 edges which covers all papers in the period from January 1993 to April 2003 (124 months) on this topic. If a paper i cites paper j, the graph contains a directed edge from node i to node j. The graph does not contain any information from the links to or from the papers outside the dataset.

We run our experiments on the High Performance Computing (HPC) cluster of the Faculty of Applied Science, Simon Fraser University[1]. The nodes of the cluster have Intel Xeon CPUs from 2.40 to 2.80 GHz with up to 16 cores. Each of our experiments is run single-threaded on a single core of a machine. The machines have up to 32 GB of memory.

In our experiments with the web and host graphs, we use a value of 0.15 for random jump probability α (see Chapter 3), i.e., a damping factor of $d = 0.85$ which is the common value used in the PageRank algorithm on the web graph. In our citation graph experiments, however, we need to use a different α value. This is because the behavior of a random surfer in jumping from paper to paper is different than a random web surfer jumping from web page to web page. We use a random jump probability of $\alpha = 0.5$ for the citation graph as has been suggested before in the literature [10, 22]. In order to find an appropriate value, they assign different values to α, calculate PageRank scores, compare the results with the other ranking methods, and then choose the best value for α that matches the previous ranking methods.

We run each of our experiments on a set of target nodes. We use two such sets for each graph. The first set of target nodes is a sample of the set of all graph nodes, chosen in a way that it ensures that we consider nodes in the whole range of node rankings (based on the PageRank algorithm). Consider the sequence of all graph nodes sorted in decreasing order of PageRank score. From this sequence in the host and citation graphs, we pick 1000 nodes at uniform distances; for example, from the 27,000 nodes of the citation graph, nodes ranked 1st, 28th, 55th, ... are selected. From the web graph, 100 such nodes (at uniform distances) are taken. This set of target nodes is indicated as *U.P. 1000 nodes* (standing for uniformly picked) in the following figures. On the other hand, we also would like to evaluate our algorithm specifically on high-ranked nodes, which have many more contributors with a more diverse range of page contributions. Moreover, these nodes are of higher importance and may be more likely to be a target in practice. Therefore, as the second target set we take the top-ranked 1000 nodes from the host and citation graphs, and the top 100 nodes from the web graph.

[1]`https://wiki.cs.sfu.ca/HPC/`

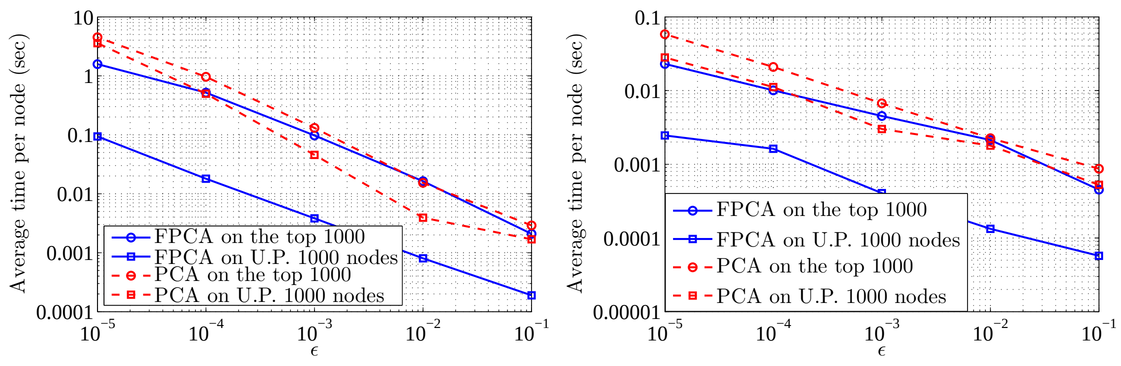
5.2 Results

We conduct several experiments with our algorithms. First, we report the running time of our algorithms. Then, we evaluate the absolute errors in the approximated page contribution values. We also compare the observed errors with the theoretical upper bounds obtained in Section 4.2 to show that: (i) our analysis of the worst-case approximation error of our algorithms is reasonably tight, and (ii) our algorithms can achieve orders of magnitude smaller average-case errors compared to the theoretical worst-case errors. Moreover, we evaluate the relative error of the approximated values to illustrate the approximation accuracy with respect to the actual page contribution values using Definition 3. We also evaluate the accuracy of the estimated page contribution ratios, i.e., page contribution divided by the PageRank score of the corresponding target node.

We note that to evaluate an estimated page contribution value for (v, u), we first need the actual value of $PageCont(v, u)$, which can be obtained using Equation 3.2 in Section 3.1. We used this equation and calculated the actual page contribution values for host and citation graphs by repeatedly running the PageRank algorithm N times on these graphs. However, for the web graph dataset this is not feasible to calculate. This is because such task involves 105 million times computation of PageRank on the graph (each taking up to hours). To work around this problem, we use the values obtained by the APC algorithm (with a small ϵ) as reference page contribution values for the web graph. We note that this may not be a 100% accurate measure for our evaluations, especially where we need to analyze the observed errors with exact theoretical expectations. We therefore present our main evaluation results on the host and citation graphs (where exact reference values are available) in Sections 5.2.1 through 5.3, and then report the results of our experiments on the web graph dataset in Section 5.2.5.

5.2.1 Running Time

For each given target node, our algorithm returns a set of contributing nodes as well as their approximated page contribution values. We plot in Figure 5.1 the average time taken by our algorithm for each target node. The figure shows that when target nodes represent the whole graph (i.e., U.P. 1000 nodes), the FAPC algorithm takes between 1 to 95 ms on the host graph and between 0 to 3 ms on the citation graph. On the other hand, as expected, a longer time is taken for the top-ranked 1000 nodes. This is because those are the nodes



(a) On the host graph.

(b) On the citation graph.

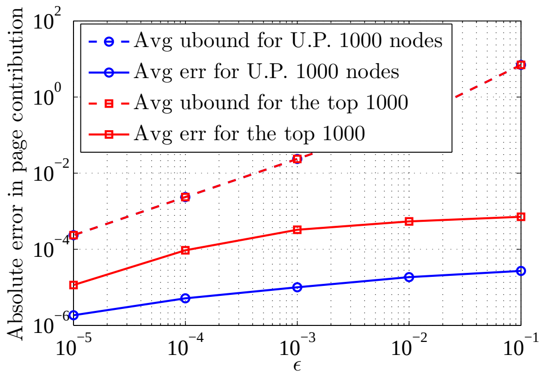
Figure 5.1: Average time taken per target node to find its contributor nodes and their approximated page contributions.

with the highest PageRank score, and therefore with the highest number of contributor nodes to take into account. For the top 1000 nodes, the FAPC algorithm takes up to 1.6 s for the host graph and up to 23 ms for the citation graph. The time for the median value $\epsilon = 10^{-3}$, which provides reasonable accuracy as shown in the following, is about 100 ms for the FAPC algorithm on the top 1000 nodes of the host graph, and it is under 4 ms for all other experiments with FAPC.

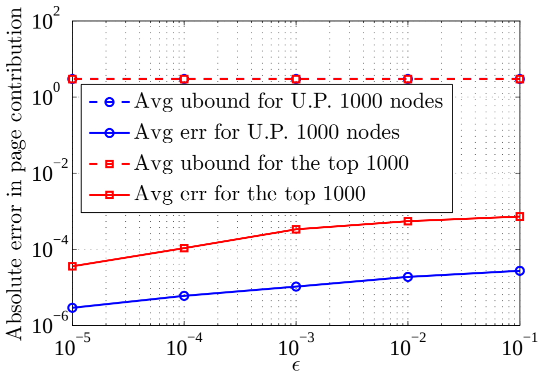
The APC algorithm takes longer times as shown by the dashed lines in Figure 5.1. This is expected due to the additional calculations done for approximating $\mathbf{cv}_u(u)$ values. For the median value $\epsilon = 10^{-3}$, this time on the host graph is about 50 and 130 ms for U.P. 1000 nodes and the top 1000 nodes, respectively. In the most time-consuming case, which is with $\epsilon = 10^{-5}$, the time taken per target node reaches 4.5 s. For the citation graph (Figure 5.1(b)), the APC algorithm takes between 0 to 30 ms for the U.P 100 nodes, and between 0 and 60 ms for the top 1000 nodes.

5.2.2 Absolute Errors and Theoretical Upper Bounds

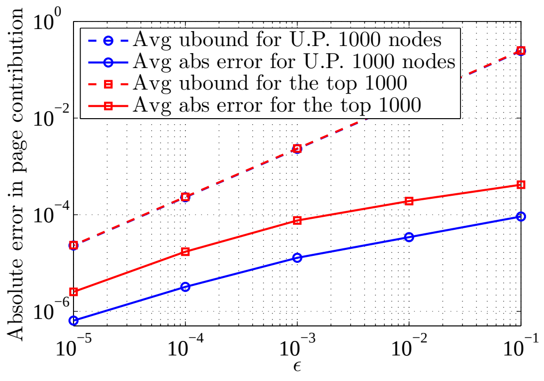
We first evaluate the accuracy of the algorithms in terms of the absolute errors. For this purpose, we need the actual value of *PageCont*(v, u), which we obtained by running the PageRank algorithm N times on the host and citation graphs. Each time, one of the nodes is voided and PageRank is applied, which gives us the page contribution of the voided node to other nodes of the graph. Note that this is an intensive procedure that has taken several



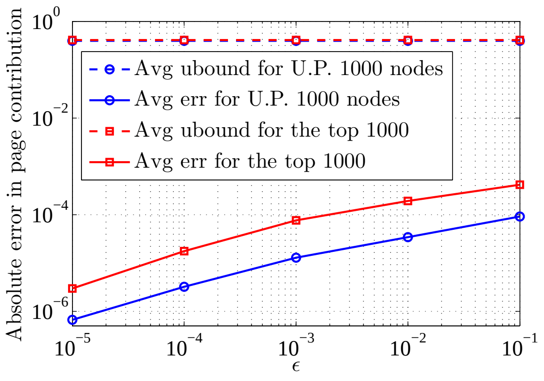
(a) On the host graph with APC algorithm.



(b) On the host graph with FAPC algorithm.



(c) On the citation graph with APC algorithm.



(d) On the citation graph with FAPC algorithm.

Figure 5.2: Absolute error in estimated page contributions and its theoretical upper bound.

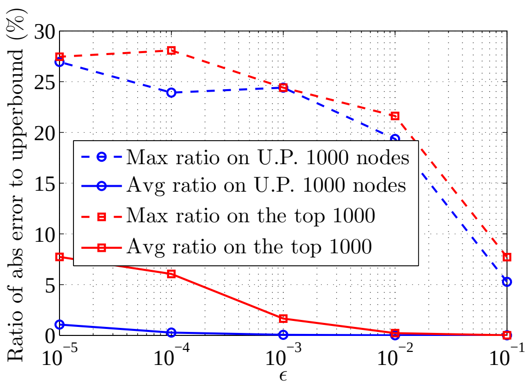
days on our host graph and citation graph datasets, and is only performed to obtain the reference $PageCont(v,u)$ values for comparison. Moreover, as discussed in Section 5.2 this procedure is not feasible on the web graph dataset.

The absolute error of the values estimated by our algorithms are shown in Figure 5.2. Consider a particular (v,u) pair where we approximate the page contribution of node u to node v. The average of the absolute error of our approximation (averaged over all (v,u) pairs) is plotted as the solid lines in Figure 5.2. The theoretical worst-case error is plotted with dashed lines, which almost overlap in the figures.

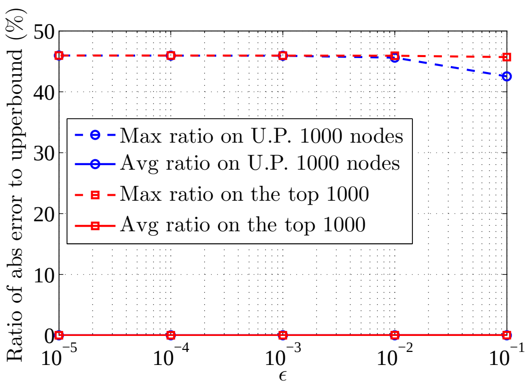
The small values of the average errors show the accuracy of our algorithms in practice. For example, even with $\epsilon = 0.01$ which is a relatively large error bound, the absolute error does not exceed 4×10^{-5} for the 1000 uniformly picked nodes of the host and citation graphs, which is a small value. For the top 1000 nodes, which are more challenging target nodes as discussed earlier, the absolute error still does not exceed 6×10^{-4}, which is again a small value compared to actual page contributions; the scale of the obtained approximation errors with respect to actual values are further illustrated through analysis of relative errors in the next subsections.

Although we have proven theoretical upper bounds on the absolute error of our algorithms, they actually perform much better in practice in the average case. To demonstrate this, for each (v,u) pair we consider the worst-case error bound on $PageCont(v,u)$, see Section 4. The average of this bound over all (v,u) pairs is plotted in Figure 5.2 as the dashed lines. This bound is a function of ϵ for the APC algorithm, but it does not depend on ϵ in the FAPC algorithm; the dominating error factor in that case is the fixed value assumed for $\mathbf{cv}_u(u)$. In all cases, the actual average absolute errors are orders of magnitude smaller than the worst-case error bounds (note the log-log scale of the figures).

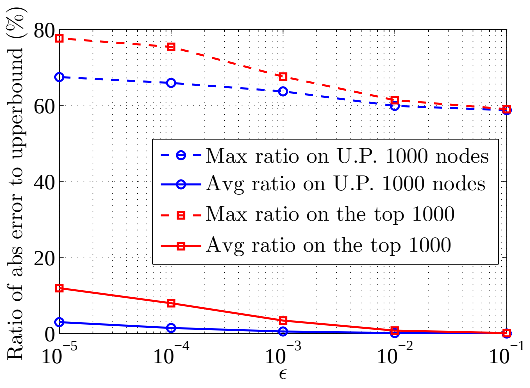
To further demonstrate the accuracy of our algorithms for the majority of nodes, we consider the ratio of the obtained absolute error to the worst-case error bound for each (v,u) individually. We plot the average and the maximum of this ratio over all considered (v,u) pairs in Figure 5.3. The average ratio barely exceeds 10% for the APC algorithm, which confirms the significantly smaller error in the average case compared to the worst-case error bound. For the FAPC algorithm, the average ratio is almost 0 (lines overlapping the horizontal axis). This is due to the special distribution of $\mathbf{cv}_u(u)$ values which is equal or very close to the minimum value of α in the different datasets (see Section 4.2.3). This property makes the average-case accuracy of the FAPC algorithm much higher than the



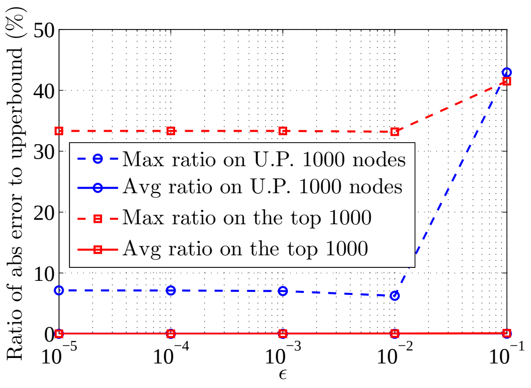
(a) On the host graph with APC algorithm.



(b) On the host graph with FAPC algorithm.



(c) On the citation graph with APC algorithm.



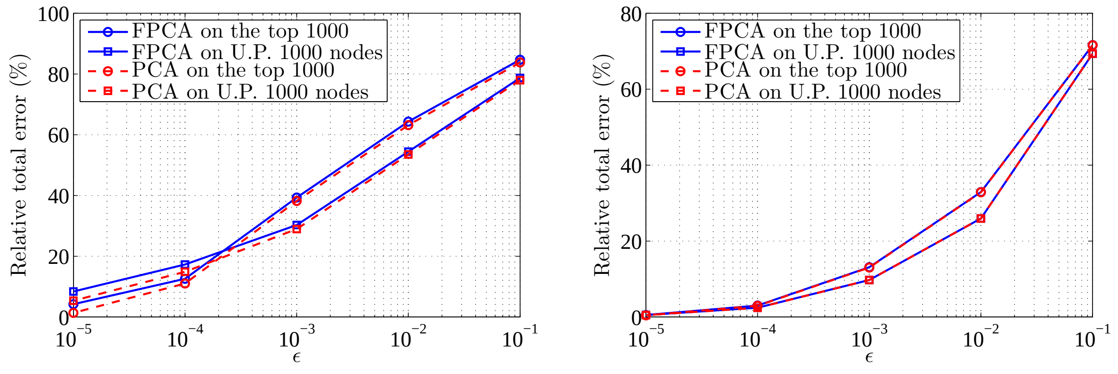
(d) On the citation graph with FAPC algorithm.

Figure 5.3: Average and maximum ratio of absolute error to the worst-case error bound.

unlikely worst case, which also shows the loose theoretical upper bound.

Another interesting finding in this experiment is that the maximum ratios, which are the dashed lines in Figure 5.3, reach up to 80% for APC algorithm on the citation graph (Figure 5.3(c)). This indicates that our error bound analysis is reasonably tight.

We also note that for the FAPC algorithm, this ratio does not exceed 50% even with large ϵ values. This is because of the special distribution of $\mathbf{cv}_u(u)$ values on the graphs as shown earlier (see Section 4.2.3). This distribution makes the estimated values more accurate than expected. The majority of $\mathbf{cv}_u(u)$ values are close to the fixed value we assumed, α, whereas a more uniform distribution of $\mathbf{cv}_u(u)$ in [0, 1] would have resulted in larger ratios in Figures 5.3(b) and 5.3(d).



(a) On the host graph. (b) On the citation graph.

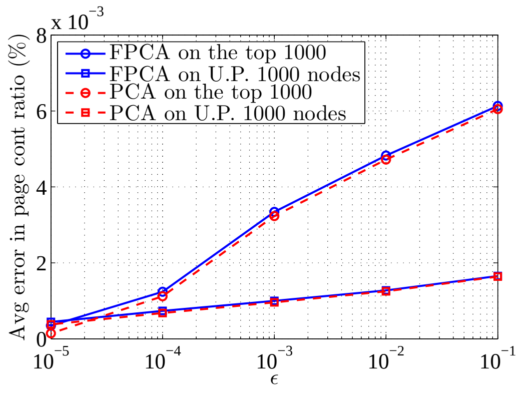
Figure 5.4: Overall relative error in estimated page contributions: the sum of the absolute errors divided by the sum of the actual page contributions.

5.2.3 Relative Error of the Approximated Values

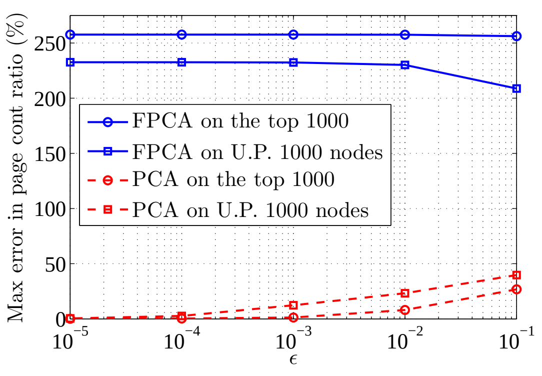
In addition to the absolute error, we also evaluate the accuracy of our algorithms in terms of the overall relative error of the estimated page contributions. This error is measured as follows:

$$\frac{\sum_{v \in \text{target nodes}} \sum_{u \in \text{contributors to } v} \left| \mathit{EstimatedPageCont}(v,u) - \mathit{ActualPageCont}(v,u) \right|}{\sum_{v \in \text{target nodes}} \sum_{u \in \text{contributors to } v} \mathit{ActualPageCont}(v,u)} \tag{5.1}$$

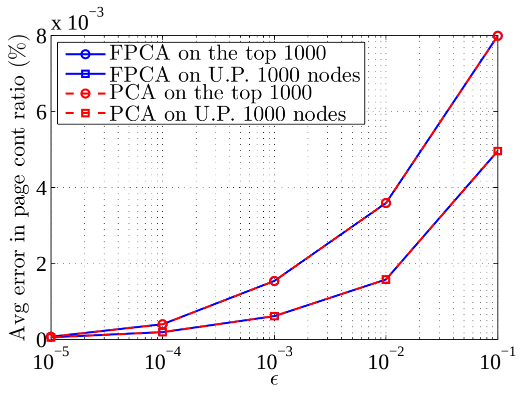
Note that we do not calculate per-node relative errors since $\mathit{ActualPageCont}(v, u)$ may be 0 in many cases, resulting in an undefined relative error value. Instead, we draw the overall relative error from Equation (5.1) in Figure 5.4. The figure shows that, for instance, when ϵ has its median value (10^{-3}) this error is less than 40% and 20% on the host and citation graphs, respectively. This suggests that a choice of $\epsilon = 10^{-3}$ provides reasonable accuracy while also resulting in fast running times as discussed in Section 5.2.1. We also note that the relative error of the FAPC algorithm (solid lines) is only slightly higher than that of the APC algorithm (dashed lines) for the host graph. For the citation graph, the two curves are actually overlapping. This is because the fixed value assumed for $\mathbf{cv}_u(u)$ in Section 4.2.3 is a close estimate for most of the nodes according to the special distribution of $\mathbf{cv}_u(u)$ values in the different datasets (Figure 4.2).



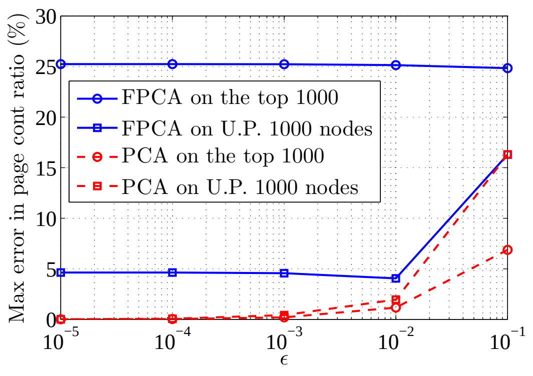
(a) Average on the host graph.



(b) Maximum on the host graph.



(c) Average on the citation graph.



(d) Maximum on the citation graph.

Figure 5.5: Average and maximum relative error in page contribution ratio: the fraction of page contribution over the PageRank score of the target node.

5.2.4 Page Contribution Ratios

We analyze the accuracy of the estimated page contributions with respect to the PageRank score of the corresponding target nodes. We consider the *page contribution ratio* as the page contribution divided by the PageRank score of the target node, i.e., what fraction of node v's PageRank score would have been lost if node u did not exist. These ratios are normalized as percentages in $[0, 100]$. We plot in Figure 5.5 the average and the maximum error in page contribution ratios obtained in our experiments. We find that the error in the estimated page contribution ratios (as percentages) is on average less than 0.01% (Figures 5.5(a) and 5.5(c)). Moreover, similar to the observation in Figure 5.4(b), the average error for the two algorithms (the dashed vs. the solid line) is very close; it is actually overlapping for the citation graph in Figure 5.5(c).

The maximum of these errors across all (v, u) pairs is shown in Figures Figures 5.5(b) and 5.5(d). This value for the APC algorithm is reasonably small: 12% and 0.4% with the median value of ϵ ($\epsilon = 0.001$) for the host and citation graphs, respectively. The notable smaller value on the citation graph is due to the smaller deviation of the citation graph nodes from the fixed value assumed for $\mathbf{cv}_u(u)$ compared to those of the host graph nodes. We also note that the maximum error in page contribution ratio reaches over 250% for the FAPC algorithm on the host graph (Figure 5.5(b)). This phenomenon happens on a small minority of nodes whose actual $\mathbf{cv}_u(u)$ value is close to 1—about 7 times higher than the assumed value of 0.15. Nevertheless, the error for the vast majority of nodes is a small value as confirmed by the below-0.01% average error in Figure 5.5(a).

5.2.5 Web Graph Results

As discussed in Section 5.2, reference page contribution values for the web graph dataset are not available to us since they are infeasible to compute. Thus, we use $\mathit{PageCont}(v, u)$ values obtained by the APC algorithm as an approximate reference. We also note that the APC algorithm needs to calculate a $\mathbf{cv}_u(u)$ value for nearly 40 million nodes in our experiments; recall that $\mathbf{gcv}_v(u) = \mathbf{cpv}_v(u)/\mathbf{cv}_u(u)$. In other words, if we denote by $\mathbf{C}_v$ the set of nodes for which APC needs to find $\mathbf{cv}_u(u)$ (i.e., any node u such that $\mathbf{cpv}_v(u) \geq \epsilon$), we have $|\sum_{v \in \text{ target nodes}} \mathbf{C}_v| \simeq 40,000,000$ (given that the smallest ϵ value in our experiments is 10^{-5}). Therefore, the APC algorithm itself may take long and not finish in a reasonable amount of time for very small ϵ values.

Therefore, we decided to use use a fixed value of $\epsilon = 10^{-2}$ for the calculation of all necessary $\mathbf{cv}_u(u)$ values, which is the most intensive part of the job. On the other hand, to increase the accuracy of page contribution estimates $\mathbf{gcv}_v(u) = \mathbf{cpv}_v(u)/\mathbf{cv}_u(u)$, we use a smaller ϵ value for finding $\mathbf{cpv}_v(u)$ values—$\epsilon = 10^{-5}$. Having obtained the estimates for reference $PageCont(v, u)$ values, we measure the absolute error, the overall relative error, and the error in page contribution rations in the same manner as in earlier experiments with host and citation graphs.

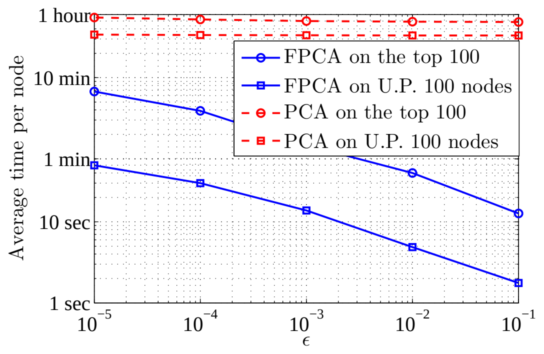
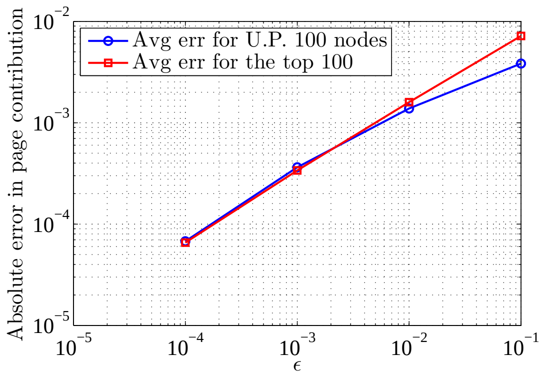


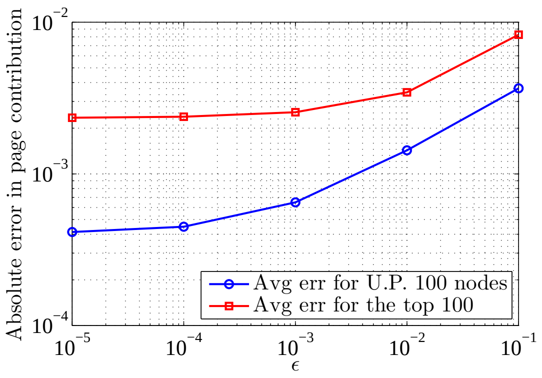
Figure 5.6: Average time taken per target node to find its contributor nodes and their approximated page contributions in the web graph dataset.

We first measure the running time of the APC and FAPC algorithms on the web graph, and plot it in Figure 5.6. The FAPC algorithm on the U.P. 100 nodes has taken from 2 to 50 seconds per node on average for different ϵ values. On the top 100 nodes, which are the most time-consuming ones in the graph, this algorithm has taken from 13 seconds to up to 7 minutes per node; it is 40 seconds for $\epsilon = 10^{-2}$ which is enough for providing a reasonable accuracy, as demonstrated shortly.

The running time of the FAPC algorithm shows a different behavior; it is almost a fixed amount independent of ϵ. This is because given an ϵ value, we only use it for finding $\mathbf{cpv}_v(u)$ values. For the calculation of $\mathbf{cv}_u(u)$ for all necessary nodes (eventually 40 million), on the other hand, we use a fixed value $\epsilon = 10^{-2}$ as discussed earlier. Therefore, the majority of the running time, which is the calculation of $\mathbf{cv}_u(u)$ values, is independent of the variable ϵ, and looks almost constant in Figure 5.6: 33 minutes per node for U.P. 100 nodes and 50 minutes for the top 100 nodes. Thus, the FAPC algorithm is the preferred choice only on small graphs (thousands of nodes), and the APC algorithm is recommended for large graphs



(a) APC algorithm.



(b) FAPC algorithm.

Figure 5.7: Absolute error in estimated page contributions.

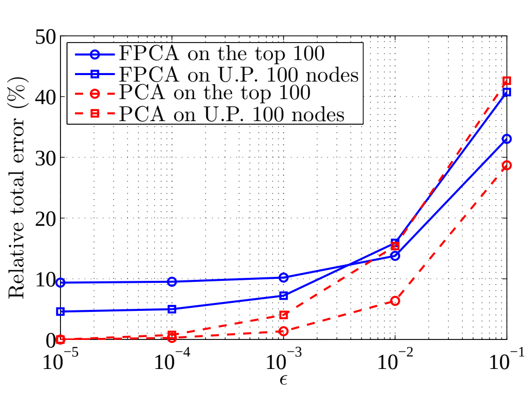
(millions of nodes) with reasonable accuracy that is shown next.

The absolute error of the APC and FAPC algorithm are shown in Figure 5.7. There is no absolute error reported for APC with $\epsilon = 10^{-5}$, since that is the reference value; the absolute error is thus 0 which is not shown in the log-scale plot in Figure 5.7. The error for the FAPC algorithm is between 4×10^{-4} and 4×10^{-3} for the U.P. 100 nodes, and between 2×10^{-3} and 8×10^{-3} for the top 100 nodes, which are small values compared to actual page contributions; the relative errors better illustrate this shortly. For the APC algorithm, the error values are even smaller.

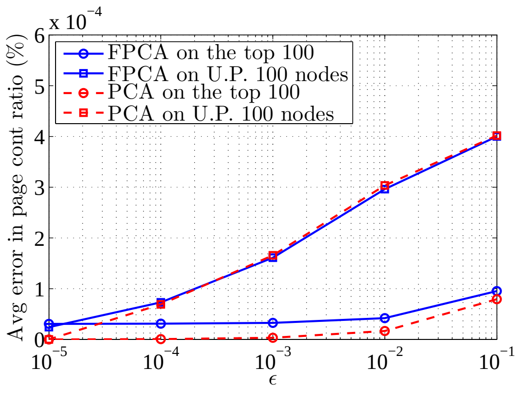
Next, the overall relative error of our algorithms (similar to Section 5.2.3) is shown in Figure 5.8(a). The figure confirms that, for instance, with $\epsilon = 10^{-2}$ which runs the algorithm fast, the relative error is below 20% in all cases. We also plot in Figure 5.8(b) the error in page contribution ratios, i.e., page contribution divided by the PageRank score of the target page. Similar to the experiments on the host and citation graphs (Section 5.2.4), a small error of at most 0.0004% is observed.

5.3 Comparison with Previous Works

As discussed in Chapter 2, the work by Zhou and Pei [31] and the one by Andersen et al. [1] are the most related works to our work, to the best of our knowledge. In this section, we compare the supporting sets found by these algorithms, which is the only common part among the three works.



(a) Overall relative error.



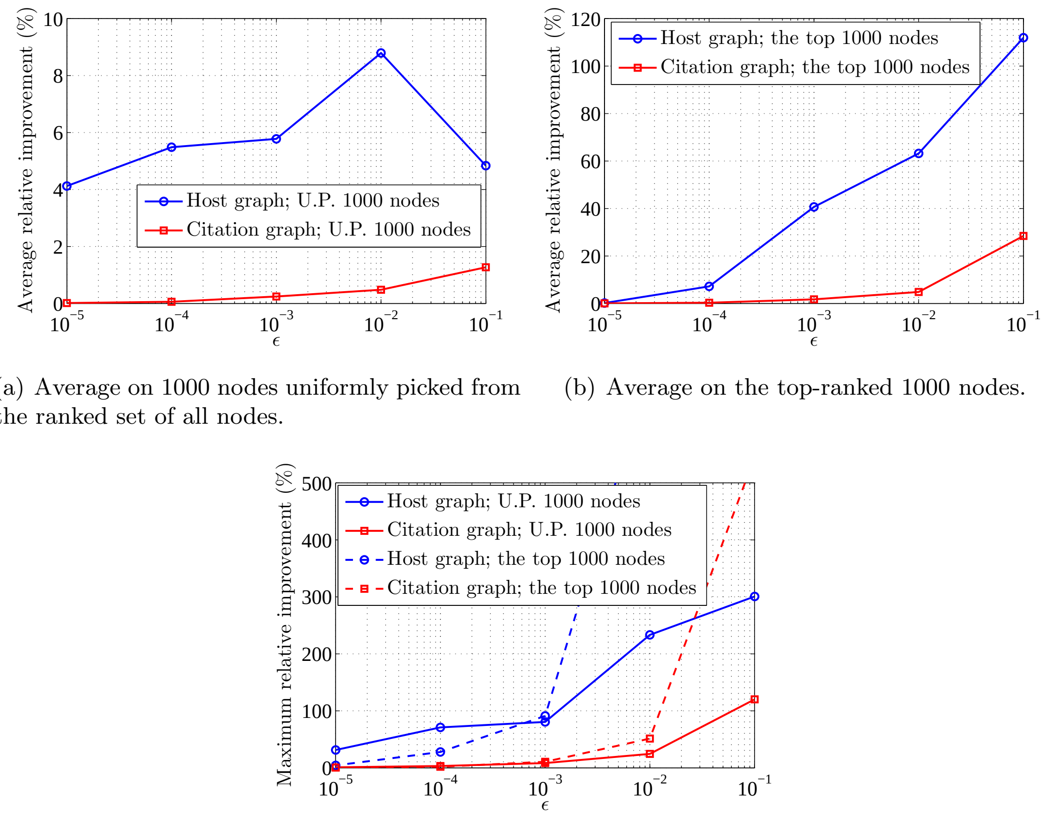
(b) Average error in page contribution ratios.

Figure 5.8: Overall relative error in estimated page contributions (the sum of the absolute errors divided by the sum of the actual page contributions), and average error in page contribution ratios (the fraction of page contribution over the PageRank score of the target node).

5.3.1 Comparison with Andersen's Algorithm

In this part, we compare the page contribution of the nodes returned by the FAPC algorithm and those of the algorithm by Andersen et al. in [1]. We note that this algorithm does not approximate page contribution values, and instead works with path contributions. Nevertheless, in this experiment we are interested in the contribution of the returned nodes to the target node's PageRank score. As discussed in Sections 1.1 and 4.3, the contribution of node u to node v's PageRank score depends on the PageRank score of u and on the paths from u to v, which are both aggregated in page contribution values that we calculate. We compare the page contribution of nodes returned by FAPC and Andersen's algorithm, and we show that those returned by FAPC have higher page contributions to the corresponding target nodes.

More specifically, to compare two returned sets A (containing nodes returned by Andersen's algorithm) and B (containing those returned by the FAPC algorithm) as contributors to a given target node, we take the average page contribution made by nodes in A and by those in B to the target node. The page contribution values used for the comparison are the actual values using from Definition 3 in order to ensure a sound comparison between the returned sets. Moreover, to ensure fairness in the comparison of sets A and B of different sizes (which typically results in the smaller set having a higher average contribution), we sort both sets in the decreasing order of the associated (path or page) contributions, and take the top $\min\{|A|, |B|\}$ elements of each set. We then take the average page contribution of the nodes remaining in A and B as x_A and x_B, respectively.



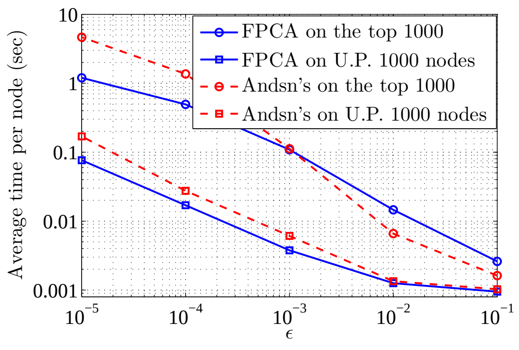
(a) Average on 1000 nodes uniformly picked from the ranked set of all nodes.

(b) Average on the top-ranked 1000 nodes.

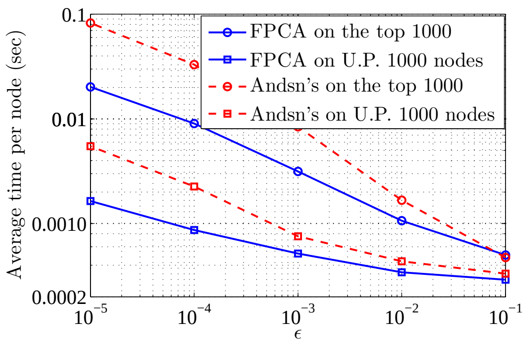
(c) Maximum on 1000 nodes (from both classes).

Figure 5.9: Relative increase in page contribution of returned nodes by the FAPC algorithm and Andersen's algorithm [1]; average and maximum on the host and citation graphs.

To show that the nodes returned by our algorithm have higher page contributions, we measure the *relative improvement* of FAPC over Andersen's algorithm as $(x_B - x_A)/x_A$. This improvement is plotted in Figure 5.9 for different target sets from the host and citation graphs. For the U.P. 1000 nodes, the average improvement (Figure 5.9(a)) is between 4% to 10% for the host graph, but it is insignificant on the citation graph. For the top 1000 nodes, on the other hand, the average relative improvement has much higher values (Figure 5.9(a)) specially for the host graph.



(a) On the host graph.



(b) On the citation graph.

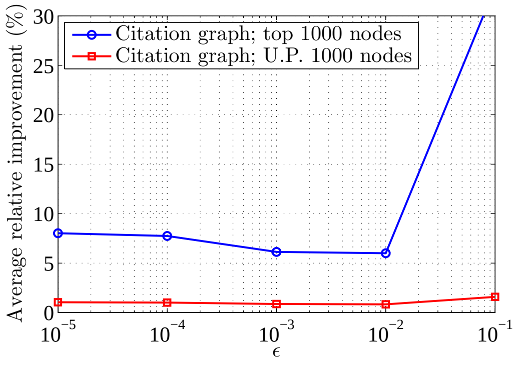
Figure 5.10: Average time taken per target node by the FAPC algorithm and Andersen's algorithm [1].

One major reason for this difference is that nodes of higher PageRank scores have many more contributors to take a set A or B from, which better demonstrates the ability of our algorithm in picking those with higher page contributions. In addition to the average relative improvement, we also plot in Figure 5.9(c) the maximum of this value observed in our experiments. The figure shows that there are target nodes for which the returned contributors by our algorithm have many times higher page contributions than those returned by Andersen's algorithm.

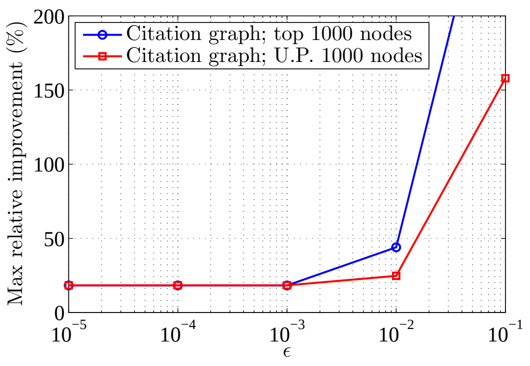
We also measure the time taken by the FAPC algorithm and Andersen's algorithm [1] in this experiment. The obtained values are plotted in Figure 5.10. As confirmed by this figure, in most cases the FAPC algorithm runs, sometimes multiple times faster (note the log-log scale of Figures 5.10(a) and 5.10(b)).

5.3.2 Comparison with Zhou and Pei's Algorithm

Next, we compare the page contribution of the nodes returned by FAPC with that of nodes returned by Zhou and Pei's algorithm [31] reviewed in Chapter 2. This work proposes a heuristic algorithm for finding a supporting set for a given target node. We conduct the same comparison as in the previous subsection; comparison of the supporting sets in terms of the page contribution of their nodes. The results are reported in Figure 5.11, which shows the relative increase in the page contribution of returned nodes (similar to Figure 5.9) in the citation graph. The figure shows that for the U.P. 1000 nodes, the average increase is



(a) Average increase.



(b) Maximum increase.

Figure 5.11: Relative increase in page contribution of returned nodes by the FAPC algorithm and the algorithm in [31]; average and maximum on the citation graph.

always positive, though marginal; we shortly show that we also found these supporting sets in a much shorter time. The average increase for the top 1000 nodes is more significant: between 6% to 34%. The maximum increase (Figures 5.11(b)), on the other hand, shows that there are target nodes for which the average page contribution of the supporting set found by FAPC is much higher (up to over 150%) than the one found by the algorithm in [31].

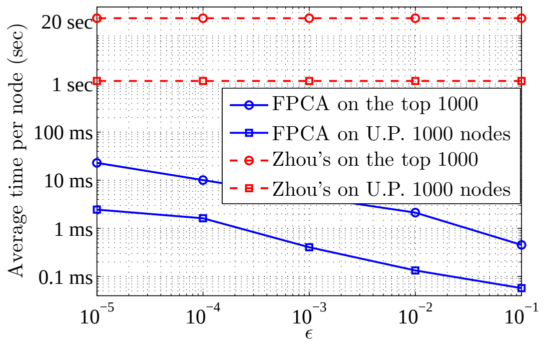


Figure 5.12: Average time taken per target node by the FAPC algorithm and the algorithm in [31].

We also report the time taken by these algorithms in Figure 5.12. As this figure confirms, FAPC has taken orders of magnitude shorter times for finding supporting sets (note the log scale of the figure).

Chapter 6

Conclusions and Future Work

In this thesis, we have taken a closer look at the PageRank algorithm and analyzed how a node collects its PageRank score from other nodes in a graph. We developed a systematic method for finding the contribution of any nodes u in a given target node v, as the different that it would make in the PageRank score of node v if node u did not exist. We also developed two approximation algorithms for estimating these values locally, i.e., without the knowledge of the whole graph. The first algorithm, the Page Contribution Approximation (PCA), provides estimates that quickly approach the actual page contribution values as we reduce the input error parameter ϵ. While an efficient solution for small scale graphs (thousands of nodes), this algorithm can have a high running time on larger graphs (hundreds of millions of nodes). Therefore, based on an observation about cyclic paths on different real-world datasets, we also developed the Fast PCA (FPCA) algorithm. FPCA trades a slight approximation error for a significantly better running time than the PCA algorithm. It is therefore the recommended choice for large graphs. We evaluated our algorithms on an partial web graph dataset, on the equivalent host graph, and on a scholarly citation graph. The experimental results demonstrate the accuracy of our algorithms in estimating page contribution values and in finding better supporting sets than similar previous works.

Our work can be used as an effective underlying tool for link-spam detection algorithms, as well as a method for measuring the success of link-based SEO techniques. As one of the directions for future research, we will employ the designed algorithms for a new link-spam detection technique, which can benefit from a better supporting set found for each target node as well as from fine-grained information such as the true contribution of individual supporting nodes to the target node.

In addition, we would like to explore the application of our work in other domains, such as social networks where PageRank-based algorithms are used for analyzing the trust/credibility of users.

Furthermore, we want to extend our page contribution approximation algorithms by enabling it to estimate the total page contribution of a group of nodes to a given target node, since it is not simply equal to the sum of the individual page contributions.

Bibliography

[1] R. Andersen, C. Borgs, J. Chayes, J. Hopcroft, V. Mirrokni, and S. Teng. Local computation of PageRank contributions. *Internet Mathematics*, 5(1):23–45, 2008.

[2] K. Avrachenkov and N. Litvak. The effect of new links on Google PageRank. *Stochastic Models*, 22(2):319–331, September 2006.

[3] L. Becchetti, C. Castillo, D. Donato, R. Baeza-Yates, and S. Leonardi. Link analysis for web spam detection. *ACM Transactions on the Web*, 2(1):2:1–2:42, March 2008.

[4] L. Becchetti, C. Castillo, D. Donato, S. Leonardi, and R. Baeza-Yates. Using rank propagation and probabilistic counting for link-based spam detection. In *Proc. of ACM SIGKDD Workshop on Web Mining and Web Usage Analysis (WebKDD'06)*, Philadelphia, PA, August 2006.

[5] J. Beel and B. Gipp. Academic search engine spam and Google scholar's resilience against it. *Journal of Electronic Publishing*, 13(3), December 2010.

[6] J. Beel and B. Gipp. On the robustness of Google scholar against spam. In *Proc. of the ACM Conference on Hypertext and Hypermedia (HT'10)*, pages 297–298, Toronto, ON, Canada, June 2010.

[7] A. Benczur, K. Csalogany, T. Sarlos, and M. Uher. SpamRank – fully automatic link spam detection. In *Proc. of the International Workshop on Adversarial Information Retrieval on the Web (AIRWEB'05)*, pages 25–38, Chiba, Japan, May 2005.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(11):107–117, April 1998.

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

[10] P. Chen, H. Xie, S. Maslov, and S. Redner. Finding scientific gems with Google's PageRank algorithm. *Journal of Informetrics*, 1(1):8–15, January 2007.

[11] Y. Chung, M. Toyoda, and M. Kitsuregawa. A study of link farm distribution and evolution using a time series of web snapshots. In *Proc. of the International Workshop*

on Adversarial Information Retrieval on the Web (AIRWEB'09), pages 9–16, Madrid, Spain, April 2009.

[12] Y. Du, Y. Shi, and X. Zhao. Using spam farm to boost PageRank. In *Proc. of the International Workshop on Adversarial Information Retrieval on the Web (AIRWEB'07)*, pages 29–36, Banff, AB, Canada, May 2007.

[13] Z. Gyongyi and H. Garcia-Molina. Link spam alliances. In *Proc. of the International Conference on Very Large Databases (VLDB'05)*, pages 517–528, Trondheim, Norway, August 2005.

[14] T. Hogg and L. Adamic. Enhancing reputation mechanisms via online social networks. In *Proc. of ACM Conference on Electronic Commerce (EC'04)*, pages 236–237, New York, NY, 2004.

[15] P. Jacso. Google scholar duped and deduped - the aura of robometrics. *Online Information Review*, 35(1):154–160, 2011.

[16] G. Jeh and J. Widom. Scaling personalized web search. In *Proc. of the International Conference on World Wide Web (WWW'03)*, pages 271–279, Budapest, Hungary, May 2003.

[17] C. Kerchove, L. Ninove, and P. Dooren. Maximizing PageRank via outlinks. *Linear Algebra and its Applications*, 429(5-6):1254–1276, September 2008.

[18] T. Largillier and S. Peyronnet. Using patterns in the behavior of the random surfer to detect webspam beneficiaries. In *Proc. of the International Symposium on Web Intelligent Systems & Services (WISS'10)*, Hong Kong, China, December 2010.

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

[20] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'05)*, pages 177–187, Chicago, IL, August 2005.

[21] R. Levien. *Attack resistant trust metrics.* PhD thesis, Iniversity of California at Berkeley, 2002.

[22] N. Ma, J. Guan, and Y. Zhao. Bringing PageRank to the citation analysis. *Information Processing & Management*, 44(2):800–810, March 2008.

[23] R. Malaga. The value of search engine optimization: An action research project at a new e-commerce site. *Journal of Electronic Commerce in Organizations*, 5(3):68–82, 2007.

[24] A. Mtibaa, M. May, C. Diot, and M. Ammar. PeopleRank: Social opportunistic forwarding. In *Proc. of IEEE INFOCOM'10 Conference*, pages 1–5, San Diego, CA, March 2010.

[25] Yahoo! Research. Web spam collections. `http://barcelona.research.yahoo.net/webspam/datasets/uk2007/`, 2007. Crawled by the Laboratory of Web Algorithmics, University of Milan, http://law.dsi.unimi.it/.

[26] H. Saito, M. Toyoda, M. Kitsuregawa, and K. Aihara. A large-scale study of link spam detection by graph algorithms. In *Proc. of the International Workshop on Adversarial Information Retrieval on the Web (AIRWEB'07)*, pages 45–48, Banff, AB, Canada, May 2007.

[27] M. Sydow. Advances in web intelligence. In *Proc. of Advances in Web Intelligence Conference (AWIC'05)*, pages 408–414, Lodz, Poland, June 2005.

[28] Tamar. Tamar's in-depth analysis into British consumers' attitudes towards search. White Paper, 2007.

[29] B. Wu and K. Chellapilla. Extracting link spam using biased random walks from spam seed sets. In *Proc. of the International Workshop on Adversarial Information Retrieval on the Web (AIRWEB'07)*, pages 37–44, Banff, AB, Canada, May 2007.

[30] B. Wu and B. Davison. Identifying link farm spam pages. In *Proc. of Special Interest Tracks and Posters of the International Conference on World Wide Web (WWW'05)*, pages 820–829, Chiba, Japan, May 2005.

[31] B. Zhou and J. Pei. Link spam target detection using page farms. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(3):13:1–13:38, July 2009.