Technical University of Crete
School of Electronic and Computer Engineering
SOFTware Technology and NETwork Applications Laboratory

Stochastic PageRank maintenance over shared-nothing architectures

Ioakeim Perros

Submitted to the School of Electronic and Computer Engineering
in partial fulfillment of the requirements for the Master of Science

Thesis committee:
Professor Minos Garofalakis (Advisor)
Associate Professor Antonios Deligiannakis
Associate Professor Michail Lagoudakis

July 4, 2014

Research supported by: FP7-ICT-2011 Collaborative STREP
"LEADS - Large-Scale Elastic Architecture for Data-as-a-Service" Project

Abstract

PageRank is established as one of the leading ranking algorithms in the web. To keep up with the increasing size of the web graph and the high frequency of updates, recent research has focused on parallelizing PageRank algorithms as well as the continuous maintenance of PageRank scores over streaming updates. However, no work up to now has considered PageRank maintenance over distributed shared-nothing architectures, such as the large-scale clouds available in the Internet today. This work bridges this gap by proposing the first known efficient stochastic algorithm for PageRank maintenance over distributed shared-nothing architectures. The algorithm effectively minimizes the amount of state per system node, enabling a small communication and memory footprint. The algorithm's correctness is theoretically established. An extensive experimental evaluation with real web and social network data shows the efficiency and accuracy of the proposed approach, and its superiority compared to the state-of-the-art competitors.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor, Professor Minos Garofalakis. His trust and encouragement have always been a factor of strong motivation and his advice and sharp insight have assisted me through every step of this work. I would also like to seize the opportunity and thank him for his invaluable support and guidance during the process of applying for a Ph.D. candidate's position.

I would like to thank Professor Antonios Deligiannakis and Professor Michail Lagoudakis for serving on my thesis committee, as well as for their support and for our fruitful discussions concerning my future steps.

I would also like to thank Dr. Odysseas Papapetrou for his ideas and insightful comments shaping some aspects of this thesis and for his continuous support during the progress of this work.

Furthermore, I would like to thank my friends Ioannis Demertzis and Sofia Nikolakaki for encouraging and supporting me in all of my pursuits, as well as for making this two-year journey so enjoyable. I am also grateful to all my friends, E. Alibertis, A. Chionis, D. Iliou, N. Kofinas, V. Lazaratou, E. Orfanoudakis, E. Papalexakis, N. Pavlakis, S. Pavlioglou, E. Soulas, A. Tsoubelis, for their support throughout this period. Each one of them has aided my endeavors in his/her own unique way.

I would like to thank my dear Athena for constantly being there for me, for believing in my abilities and for inspiring me to follow my dreams.

Last, but definitely not least, I would like to thank my parents, Manoussos and Anastasia and my brother Iosif, for their unconditional love, and the way this is expressed.

Contents

List of Figures

1 Introduction

> *It is great happiness to be praised by them who are most praiseworthy.*
>
> **— Sir Philip Sidney**
> English poet

1.1 Motivation and Problem Statement

The abundance of data originating from DNA sequences and particle collision experiments to social and sensor networks has raised several Big Data research challenges to prominence. Many of those huge modern datasets are characterized by a network structure, thus they can naturally be represented and handled as graphs. The most representative example of those graphs is the World Wide Web, consisting of over one *trillion* links. As of examples from social network graphs, Facebook consisted of over 800 million active users, with hundreds of billion friend links in January 2012, while Twitter had over 41.7 million users with 1.47 billion social relations in July 2009 [SK12]. Apart from their enormous size, the continuous evolvement of such graphs over time (i.e. streaming nature) highlights the importance of inventing new models and algorithms, in order both to manage and extract knowledge from *huge graph streams*.

The enormous volume of graph time-evolving data, coupled with the possible physical distribution of their sources (distributed crawlers or social networks, IP-network monitoring, power-constrained wireless sensornets) dictate the use of a real-time *in-situ* processing strategy (locally at the sites where data is observed) [Gar+13]. Besides this, there is an increasing availability of Cloud computing services (e.g. Amazon EC2 [Wal08]), offering on-demand access to storage and computing services [Low+12]. As a result of both of these factors, it is crucial to devise and efficiently implement novel algorithms which will be able to handle this massive amount of *distributed graph streams*.

[1] The term comes from Larry Page, co-inventor of the algorithm and founder of Google [RU11]

PageRank [1] [Pag+99] has been characterized as one of the most prominent algorithms for graph mining. It was initially focused on web-page ranking and on combating spam web-pages. Its success in assessing a higher importance score (PageRank measure) to web-pages serving users' needs established Google as the leading search engine of the World Wide Web. The model's main intuition is that one *web-page is considered important, if other web-pages pointing to it are considered important as well*.

Besides being the core ranking measure of Google' s web search engine, the algorithm has been exploited in a variety of additional domains. For example, in the domain of social networks, it has been frequently considered as a baseline method for social influence measuring [Agg+11; Che+10]. Recently, the researchers' intuition that the PageRank measure of authority is closely connected with the measure of influence in a social network has been theoretically verified [Xia+13]. It has also been exploited for graph clustering. Andersen et al. in [And+06] implemented a modified version of the PageRank algorithm (personalized PageRank), in order to iteratively find a set of strongly related nodes, call them a group, remove them from the graph and repeat until the graph becomes empty. PageRank has even found applications in the bioinformatics field. For example, the algorithm proposed in [Win+12] combines gene expression measurements with a PageRank-based ranking of genes in a network built upon relations between them. The target of this work is the outcome prediction of a patient's clinical status. Furthermore, it has been applied to the field of immunization and epidemics prevention. For example, in [Ada+13], Prakash et al. proposed a method of allocating antidotes to hospitals, following the order of their PageRank ranking. It is therefore not surprising that the PageRank algorithm has been characterized as one of the most influential algorithms in the area of data mining [Wu+08].

Despite its success, the original PageRank algorithm relied on power iterations and was therefore difficult to scale with the expanding web graph size. Accordingly, a large number of algorithms for computing PageRank scores has been proposed that improve on efficiency and scalability of the original algorithm. The most prominent family of these algorithms focuses on scaling the PageRank computation by distribution, e.g., [IT10; Par+06]. These algorithms typically operate on web graph snapshots, i.e., static graphs. Since in real-life, the vast majority of domains where PageRank is useful produces streaming, dynamic, graphs (e.g. continuous web-crawling, online social networks), these algorithms have to be continuously re-executed in order to handle the updates. Another direction considered in the lit-

erature for increasing scalability relies on the premise that continuous maintenance of the PageRank scores is more cost-effective than periodic full re-computation. The majority of these works again incorporates linear algebraic methods, and is therefore difficult to scale, e.g., [LM04; McS05; Ton+08]. Notable recent exceptions include the Monte-Carlo PageRank algorithms [Bah+10; Sar+08], which approximate PageRank scores by simulating the random surfer model through the construction of random walks over the web graph. These algorithms are shown to be approximately an order of magnitude more efficient (on the number of web links) than prior linear algebraic methods [Bah+10], precisely due to their stochastic nature. Still, these works have two main limitations: (a) they require a static set of web pages; only web links between the web pages that already exist at the initialization of the algorithm can be added or removed, and, (b) they are still, by nature, either centralized, or parallelized over the shared-memory model, and therefore have a limited scalability.

1.2 Thesis Contributions

Interestingly, though, there currently exists no algorithm that combines distributed computation and continuous maintenance of PageRank scores over a dynamic graph. The main reason for this gap is that the algorithms enabling maintenance over dynamic web graphs typically need a holistic view of the web graph to handle the updates, which is a limiting factor for distribution. With this work, we fill the gap by proposing DSPM (Distributed Stochastic PageRank Maintenance), a novel algorithm to maintain the PageRank scores for dynamic graphs over large-scale distributed networks. Unlike previous algorithms, DSPM does not rely on the shared-memory model. It is therefore much more applicable to modern cloud-based architectures, such as Amazon's EC2 [Wal08], and to globe-scale clusters being either general-purpose (e.g. Planetlab [Spr+06]) or focused on wide-scale stream processing [LY08]. As such, it has better scalability properties, and can even be used by SMEs with small upfront investment.

The core of DSPM is a Monte-Carlo algorithm that allows approximating PageRank with minimal state maintenance. Similar to previous Monte-Carlo PageRank algorithms [Avr+07; Bah+10], DSPM constructs random walks over the web-graph, and uses the number of visits at the nodes (webpages) to estimate PageRank. The stark difference of DSPM with previous algorithms for dynamic graphs [Bah+10; Sar+08], however, is that DSPM does not require the accurate maintenance of all the random walk segments, in order to enable future updates on the web graph. This is achieved by

an advancement in the theoretical analysis, and enables us to drastically reduce both the memory and network requirements of the algorithm, increasing its scalability on the targeted architectures. Furthermore, DSPM allows streaming updates on both the set of web links and the set of web-pages, whereas previous works always assumed a static set of web pages.

The contributions of this thesis can be summarized as follows.

- This work is the first to investigate the problem of continuous maintenance of PageRank scores over a distributed shared-nothing architecture. The considered streaming web graph model enables the addition or removal of *both* nodes in the web graph (i.e. web-pages), and edges (i.e. links).

- This work proposes several extensions of the state-of-the art Monte-Carlo algorithm [Bah+10] for decentralized PageRank maintenance in order to lift its shared-memory requirement, and examine its constraints and scalability potentials.

- This work presents DSPM, the first Monte-Carlo algorithm for distributed maintenance of PageRank scores that does not require a shared memory model and can be fully integrated in any decentralized web crawling engine (even the ones that are widely distributed over the Internet). Unlike the state-of-the-art Monte-Carlo algorithms for streaming updates, DSPM does not assume a static set of web-pages, i.e., it enables adding/removing both web pages and links in the web graph. The algorithm's correctness is theoretically established.

- The performance and accuracy of DSPM is compared to the state-of-the-art counterparts. The experiments are executed on publicly available real data sets, originating from two different domains: (a) the World Wide Web, and, (b) social networks. The experimental results demonstrate the good scalability properties of DSPM, as well as its superiority compared to alternative approaches, both in terms of network requirements and memory footprint.

1.3 Thesis Outline

In Chapter 2 the necessary background is provided and in Chapter 3 the related work is discussed. Naturally, the discussion evolves more around the state-of-the-art Monte-Carlo algorithms, since these are shown to be substantially more efficient than prior linear algebraic methods. Chapter 4 shows how the state-of-the-art Monte-Carlo algorithm for continuous PageRank maintenance can be adapted to shared-nothing architectures, whereas Chapter 5 discusses our main contribution, the DSPM algorithm. A thorough experimental comparison with real data sets will be presented in Chapter 6. The conclusion as well as a brief discussion on future work is provided in Chapter 7.

2 Background

2.1 Foundations of the PageRank model

PageRank ranks web-pages by exploiting the web graph structure. Its key intuition is that the importance of each web-page is recursively determined, through the importance of the web pages pointing to it. Formally, let $G(V, E)$ denote the web graph, with V representing the set of web-pages, and E the set of links between all web-pages in V. With $outDeg(i)$ is denoted the out-degree of page i, and with n the number of web-pages in V. In its original form, the hyperlink matrix P of size $n \times n$ is defined as follows: $p_{ij} = \frac{1}{outDeg(i)}$ if $(i, j) \in E$, and $p_{ij} = 0$ otherwise.

The founders of the PageRank vector defined it as the stationary distribution of a Markov chain whose state space is the set comprising of all the web-pages. In order to ensure existence and convergence to the stationary distribution, they forced some transformations to P so as to be used as the transition matrix of the underlying Markov chain. The first such transformation is that if a page i has no outgoing links (row full of zeros), then $p_{ij} = \frac{1}{n}$, i.e. the outgoing links of page i are uniformly distributed across all web-pages. The transformed matrix is now row-stochastic (each one of its rows corresponds to a probability distribution). Another desired property of the Markov chain's transition matrix is irreducibility, which is proven to be obtained when the directed graph corresponding to this matrix is strongly-connected (i.e. every vertex is reachable from every other vertex) [LM03]. As a result, the transition matrix is transformed as follows:

$$\tilde{P} = (1 - \epsilon)P + \frac{\epsilon}{n} J_n,$$

where J_n is the all-ones matrix of size $n \times n$ and ϵ is the probability that a surfer moves to a random web-page, empirically set to 0.15 [Pag+99]. The matrix $\tilde{P}$ is both stochastic and irreducible. Thus, the Perron-Frobenius theorem states that its principal eigenvector [1] π exists and that the dominant

[1] eigenvector corresponding to the maximum eigenvalue

eigenvalue associated with it is equal to 1 [Eld07]. Thus, the following eigenvector problem is formed:

$$\pi\tilde{P} = \pi, \sum_i \pi_i = 1$$

The vector π satisfying the above set of equations is called the PageRank vector and the traditional method of acquiring it is the Power Method [Pag+99], which dictates the repetitive calculation of the vector-matrix multiplication $\pi\tilde{P}$ until convergence.

2.2 Approximating PageRank

Following the original PageRank paper, and motivated by the practical importance and success of the algorithm in combination to its scalability limitations on web-scale data, several algorithms have been proposed (cf. Chapter 3). The following simple re-formulation of the PageRank eigenvector problem gave rise to a family of those algorithms, named as Monte-Carlo, because of their randomized nature:

$$\begin{aligned}
& \pi\tilde{P} = \pi \\
\implies & 0 = \pi - \pi\tilde{P} \\
& = \pi - \pi\left[(1-\epsilon)P + \frac{\epsilon}{n}J_n\right] \\
& = \pi[I - (1-\epsilon)P] - \frac{\epsilon}{n}\pi J_n \\
& \overset{(a)}{=} \pi[I - (1-\epsilon)P] - \frac{\epsilon}{n}\underline{1}^T \\
\implies & \pi = \frac{\epsilon}{n}\underline{1}^T[I - (1-\epsilon)P]^{-1} \\
& \overset{(b)}{=} \underline{1}^T \frac{\sum_{k=0}^{\infty}(1-\epsilon)^k P^k}{\frac{n}{\epsilon}},
\end{aligned} \tag{2.1}$$

with $\underline{1}$ denoting a column vector of all ones. (a) follows from the fact that multiplying the probability distribution π of size $1 \times n$ with an all-ones matrix of size $n \times n$ results in an $1 \times n$ all-ones vector. (b) follows from the basic property of the Neumann series: $(I - \alpha S)^{-1} = \sum_{k=0}^{\infty}(\alpha S)^k$, where S is a bounded operator on a normed vector space [Cal+08].

Prior to describing the randomized manner in which PageRank can be approximated via Equation 2.1, we introduce the notion of a random walk on a graph. It is briefly defined as the process followed by a particle beginning at some vertex and at each time step moving to another vertex

$$\sum_{k=0}^{\infty}(1-\epsilon)^k P^k = \underbrace{\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}}_{P^0\text{:Initial visits}} + (1-\epsilon)\underbrace{\begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.5 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{\substack{P^1(i,j)\text{:probability that} \\ \text{the random walk starting at web-page } \mathbf{i} \\ \text{visits } \mathbf{j} \text{ at its 1st step}}}$$

$$+(1-\epsilon)^2 \underbrace{\begin{bmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.18 & 0.18 & 0.18 & 0.28 & 0.18 \\ 0.18 & 0.18 & 0.18 & 0.28 & 0.18 \\ 0 & 0.25 & 0.25 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0 & 0.5 \end{bmatrix}}_{\substack{P^2(i,j)\text{:probability that} \\ \text{the random walk starting at web-page } \mathbf{i} \\ \text{visits } \mathbf{j} \text{ at its 2nd step}}} + \dots$$

Fig. 2.1 Depiction of the probabilistic interpretation of the PageRank model

randomly with a probability distribution governed by the transition matrix of the underlying graph [Lov93].

The remarkable probabilistic interpretation of the PageRank vector introduced by Equation 2.1 draws on the correspondence of the fraction's numerator to the expected number of visits of n random walks to each web-page in the web graph and of its denominator to the expected number of total visits of all n walks (as their length is geometrically distributed with parameter ϵ).

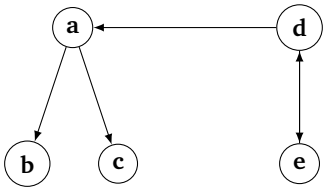


Fig. 2.2 Graph used as an example for Figure 2.1

In Figure 2.1 (the corresponding graph is depicted in Figure 2.2), the first few terms of the numerator's sum of Equation 2.1 are expanded. In order to clarify the aforementioned random-walk interpretation, we use the P^2 matrix and the walk initiated from web-page **d** as an example. The possible $1st$ step of this walk to **a** results in a 0.25 probability of propagating to each of the web-pages **b** and **c**. Likewise, its possible $1st$ step to web-page **e** results in a 0.5 probability of re-visiting **d** at its $2nd$ step (because of the bi-directional link between web-pages **d** and **e**). The $(1-\epsilon)^2$ term multiplied by the matrix P^2 corresponds to the walks' transition probability. Thus, considering that a random walk, having an ϵ probability of stopping at each step, is initiated from each page, the (i, j) position of the infinite sum's result will contain the expected number of visits to web-page **j** from the walk started at page **i**. As a result, by finally multiplying with the row vector of all-ones ($\underline{1}^T$), we extract the expected value of visit counts

to each web-page **j**, which is divided by the expected value of the total number of steps, as the length of each walk is geometrically distributed with a mean of $\frac{1}{\epsilon}$. Finally, the probabilistic interpretation described above can be trivially extended to the case when each web-page initiates $R > 1$ random walks. In this case, the normalization factor becomes $\frac{nR}{\epsilon}$.

Monte-Carlo methods have two interesting properties concerning scalability. First, they are an order of magnitude (on the number of edges in the web graph) more efficient for dense graphs than the power iteration algorithm and other linear algebraic methods [Bah+10], with only a minor decrement on the PageRank accuracy. Second, they are naturally parallelizable, i.e. they can be executed efficiently in multi-core shared-memory architectures. As such, they are the preferred algorithm for computing PageRank on Internet-scale data, like the web graph.

Even though the initial Monte-Carlo algorithms were focused on static web graphs, two recent works enable maintenance of PageRank scores over dynamic graphs, assuming that the edges of the graph (i.e. the links between the web-pages) are observed in a streaming fashion [Bah+10; Sar+08]. Here, the focus is on the latest of these works [Bah+10], which is also the state-of-the-art. The main intuition of this work will be described in the following and a more detailed explanation of its operation will be provided in Chapter 3. The key idea of FIP (Fast Incremental PageRank) is to use reservoir sampling [Vit85] over all stored random walks in order to re-distribute the random walks evenly across the new and old links of the web graph. More precisely, for a newly observed edge $(\mathbf{u}, \mathbf{w})$ (i.e. page **u** now also points to page **w**), each of the random walks passing through **u** is considered for redirection with a probability of $1/outDeg(\mathbf{u})$, with $outDeg(\mathbf{u})$ denoting the new out-degree of **u**. All selected random walks are redirected through $(\mathbf{u}, \mathbf{w})$ and the visit counts of the affected web-pages are updated accordingly (for all steps in the revoked segment of the random walk decreased by one, and for all steps in the new segment increased by one). Similar to the Monte-Carlo algorithm for static graphs [Avr+07], the expected visit counts of the web pages maintained by FIP are proportional to the true PageRank scores.

FIP is the best known algorithm for streaming PageRank maintenance, having substantially lower complexity bounds than previous alternatives. Due to its Monte-Carlo characteristics, it is also easily parallelized over shared-memory architectures [Bah+10]. However, it has two substantial limitations. First, it requires that all random walks are stored in order to

be able to revert their effect in case of redirection. This requirement can be addressed by a shared-memory model, which however severely limits the scalability of the algorithm. Second, the algorithm assumes that all nodes in the web graph, i.e. all web-pages, are available during algorithm's initialization; only the links between the web pages can change. Clearly, this is a rather unrealistic assumption for modern distributed crawlers which dynamically update both sets of web-pages and links. This argument holds for other domains of PageRank application as well (e.g. social network real-time signing up of new users). We will come back to these two requirements of FIP in Chapter 4, where we will describe an extension of the algorithm that partially addresses them.

3 Related Work

In recent years there has been a wealth of algorithms for computing PageRank. We have already described in brief the Power Iteration method and the state-of-the-art Monte-Carlo approaches in Chapter 2. For completeness, we now briefly discuss other related algorithms. Due to the vast amount of algorithms for the PageRank vector's computation, we restrict ourselves to only a few key representatives; a more complete discussion of PageRank-related literature is provided in [Ber05], and in the more recent work of [Bah+10].

3.1 Distributed PageRank computation

At first, we will refer to the subset of distributed approaches which are based on the Monte Carlo method, as our approach does. A first approach of this family of methods from the point of computing PageRank was given in [FR04]. They approximated though the PageRank measure, by exploiting no steps but the final ones of the performed walks (their destinations), thus leading to limited accuracy. Fundamental work has been published on Monte Carlo methods by Avrachenkov et al. [Avr+07] for static web graphs. This work presents various algorithms for estimating the PageRank vector via the total frequencies of the walks to each web-page and extends the theoretical analysis, in order to efficiently handle the existence of dangling pages (web-pages having no out-going edges). This approach though assumes a static web-graph and cannot be applied for continuous PageRank maintenance. Despite the fact that it has been mentioned in [Avr+07] that the PageRank scores may be updated continuously, this work neither provides any details of how exactly to perform these updates, nor give any analysis about the efficiency of doing them [Bah+10]. Since the work in [Avr+07] executes a geometrically distributed walk for each web-page, its cost is $\Omega(\frac{n}{\epsilon})$, so a naive re-computation of PageRank with this method for each edge arrival would result in a prohibitive cost $(\Omega(\frac{mn}{\epsilon}))$.

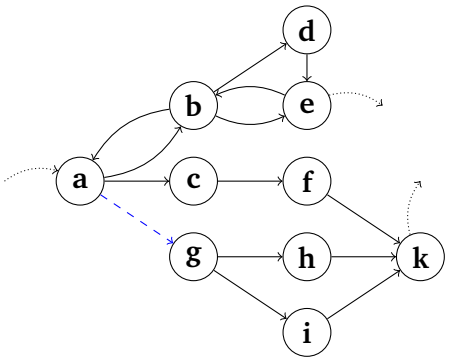
Das Sarma et al. [DS+14] recently proposed a distributed PageRank computation algorithm, based on the idea of performing a number of short walks from each node, in order to facilitate the availability of truly independent random walk segments, when those are actually computed. This computation eventually takes place by performing a stitching of the already performed short walks. This work achieves an improvement in terms of running time against the original Monte Carlo method in [Avr+07]. Despite this, it neither supports streaming updates and nor it is focused on the communication efficiency, as it initiates many walk segments that remain useless during the algorithm's execution, in order to favour the running time reduction.

Many other works attempt to tackle the distributed execution of linear-algebraic methods such as in [IT10; Par+06]. Such works though do not handle dynamic updates, apart from the ability of a trivial periodic re-execution. For completeness, we mention that the original PageRank work's [Pag+99] cost is $\Omega\left(\frac{m}{ln(\frac{1}{1-\epsilon})}\right)$, so over m edge arrivals the total cost would be $\Omega\left(\frac{m^2}{ln(\frac{1}{1-\epsilon})}\right)$.

3.2 Continuous PageRank maintenance

As concerns the continuous maintenance of the PageRank vector, theoretical work based on the Monte Carlo method under the shared memory model is published in [Sar+08]. However, the time complexity of this work is outperformed by Bahmani et al. [Bah+10] through their Fast Incremental PageRank method. They introduced a method working under the shared memory model which stores a number of initial random walk segments in a database, during the algorithm' s offline phase. During the online phase, this approach examines only a subset of the random walk segments, so as to update only those requiring a re-direction, in the case of an update edge. For each newly-added edge $(\mathbf{u}, \mathbf{w})$, each of the stored segments passing through the web-page $\mathbf{u}$ is examined and re-routed to the new edge's destination with a probability equal to $\frac{1}{outDeg(\mathbf{u})}$, where $outDeg(\mathbf{u})$ is the new out-degree of $\mathbf{u}$. This procedure is in fact analogous to the well-known Reservoir Sampling technique [Vit85], designed to randomly sample a stream, without being aware of its total size.

Figure 3.1 serves as an example of the operations performed by the FIP method. It is assumed that the number of random walks started from each web-page (parameter R) is equal to five and that during the online phase



(a) Transition from graph instance G_1 to G_2 by incorporating the edge arrival **a** $\rightarrow$ **g**

ID	Walks for G_1	Walks for G_2
a_1	**a** $\rightarrow$ **c** $\rightarrow$ **f** $\rightarrow$ **k**	**a** $\rightarrow$ **g** $\rightarrow$ **h** $\rightarrow$ **k**
a_2	**a** $\rightarrow$ **c** $\rightarrow$ **f**	**a** $\rightarrow$ **c** $\rightarrow$ **f**
a_3	**a** $\rightarrow$ **b** $\rightarrow$ **e** $\rightarrow$ **b** $\rightarrow$ **e**	**a** $\rightarrow$ **b** $\rightarrow$ **e** $\rightarrow$ **b** $\rightarrow$ **e**
a_4	**a** $\rightarrow$ **b** $\rightarrow$ **a** $\rightarrow$ **b**	**a** $\rightarrow$ **g** $\rightarrow$ **i**
a_5	**a** $\rightarrow$ **b** $\rightarrow$ **d**	**a** $\rightarrow$ **b** $\rightarrow$ **d**
$\vdots$	$\vdots$	$\vdots$
y_3	$\ldots \rightarrow$ **a** $\rightarrow$ **b** $\rightarrow$ **e**	$\ldots \rightarrow$ **a** $\rightarrow$ **g** $\rightarrow$ **h**
z_5	$\ldots \rightarrow$ **a** $\rightarrow$ **c** $\rightarrow$ **f** $\rightarrow$ **k**	$\ldots \rightarrow$ **a** $\rightarrow$ **c** $\rightarrow$ **f** $\rightarrow$ **k**

(b) Stored walk segments before and after the arrival of the update edge **a** $\rightarrow$ **g**

Fig. 3.1 Depiction of the real-time execution of the FIP method under the shared-memory model

of the algorithm, the edge **a** $\rightarrow$ **g** arrives. As such, the occurrences of web-page **a** are examined in order to update the walks stored in the database containing it. These are marked with circles in Figure 3.1. Provided that we have not selected a previous occurrence of **a** in the same walk segment for rejection, we generate a uniformly random number for each one of these occurrences. In the case of re-routing (drawn probability less than $\frac{1}{outDeg(\mathbf{a})}$), the remainder of the old random walk is rejected (i.e. the corresponding visit counters are decreased) and a new random walk is concatenated to the initial segment, starting from the new edge's destination **g**. The occurrences of **a** corresponding to the selected ones for rejection are marked with a filled circle. It is worth underlining that the second occurrence of **a** at the walk segment a_4 *is never examined for re-direction*, as the first appearance of **a** in this walk has already been selected as a start point for re-routing the same walk. A symmetric update procedure is used for handling edge deletions.

Despite its efficiency, the FIP algorithm considers a shared-memory database as the underlying infrastructure which does not permit the distribution of the updates' input sources. On the contrary, in Chapter 4, we explore how we could modify their algorithm in order to be able to support the distribution of real-time updates and in Chapter 5, a novel lightweight approach is proposed to substantially reduce both the transfer volume and the amount of memory required to achieve it.

Linear algebraic methods focused on updating the PageRank vector have been proposed as well, apart from the Monte Carlo ones, but neither of them seems suitable to real-time applications so as to consider using it as a baseline for decentralization. For example, the approach in [LM04] needs a separation of the old links from the new ones after receiving a set of updates. It also reports being susceptible to this separation of the underlying network, by stating that a bad partitioning leads to a performance as slow as the Power Iteration. Furthermore, the performance of the work in [Ton+08] heavily depends on the structural properties of the underlying graph, as it is rather inefficient ($O(mn^2)$ for m updates over n vertices) when the two parts of the bipartite graph under consideration share an equal number of nodes.

Finally, there is another line of research examining how the PageRank vector evolves as edge updates arrive (e.g. the recent work in [Bah+12]). Such works focus on an evolving graph's model, where the algorithm is unaware of the real-time updates and tries to probe the web-pages (so as

to gain knowledge about the most recent graph's instance) in a way that minimizes the total approximation error. We underline that this model is different from the streaming and distributed model of computation that we consider in this work.

4 Decentralizing streaming PageRank

As described in Chapter 2, FIP, the state-of-the-art algorithm for streaming PageRank maintenance has two key limitations: (a) it requires a shared-memory architecture for storing a repository of the random walks, and, (b) it does not allow the addition of new web-pages. In this Chapter, we show how to lift these two requirements, thereby making FIP more scalable.

4.1 Avoiding the shared-memory requirement

FIP needs to store all random walks in order to be able to revert their effects in the case of random walk re-directions. The original algorithm has assumed the existence of a distributed repository, which however was implemented over shared memory [Bah+10]. In the absence of shared memory, i.e. when the system nodes are scattered across the globe, we can keep the walk-related information in the nodes as follows. The node responsible for crawling and storing each web-page **u** also maintains triples of $< w, p, v >$, where w denotes the walk id, p the position of **u** in the random walk, and v a pointer on the next page in the walk. This information is sufficient for reproducing the random walk starting from any web-page, by exchanging of messages between the nodes. Segment rejections of previous random walks are performed by constructing *negative random walks*, which follow the original random walks, but decreasing the visit counts of the visited web-pages. Both standard and negative random walks are propagated in the network through message passing.

4.2 Enabling addition of new web-pages

Original FIP requires that the set of web-pages is static, and only the links between the pages can change. This constraint stems from the way the algorithm handles web-pages with no outgoing links; a random walk reaching a sink web-page performs a random jump to another web-page chosen with a uniform probability over all web-pages. In order to

ensure that all web-pages receive on expectation the same number of random jumps from sink nodes, all pages need to be available during the initialization of the algorithm. We avoid this requirement by changing the way we handle sink pages.

To avoid this requirement, we exploit an alternate version of the Monte Carlo PageRank approximation, proposed in [Avr+07]. This algorithm dictates that the expected value of the approximation vector is equal to the true PageRank vector, if the random walks initiated stop at dangling web-pages and the normalization factor equals the total number of steps. By emulating this technique, we freeze all random walks that reach sink pages, as proposed in [Avr+07] for static networks. In our real-time setting, these random walks will eventually continue as soon as the web-pages get at least one outgoing link. We also need to adjust the normalization factor accordingly, to account for the frozen walks: instead of dividing the visit count of each page by $\frac{nR}{\epsilon}$ to find the corresponding PageRank score, we divide it by the expected sum of total visit counts in the system. Formally:

$$\pi = \underline{1}^T \frac{\sum_{k=0}^{\infty} (1-\epsilon)^k P^k}{\sum_{i=1}^{n} X_i},$$

where X_i is the visit count of web page i. Avoiding the random jumps enables us to expand the web graph with new pages, since a newly discovered web-page is guaranteed not to have lost any randomly distributed visit counts.

Since the targeted setup is a distributed network, the above extension requires an algorithm for continuous maintenance of $\sum_{i=1}^{n} X_i$[1]. This can be inexpensively achieved by requesting each node to inform a coordinator whenever the sum of its local X_i values deviates substantially from the previously reported values. In practice, even though the visit counts of individual web-pages may vary drastically, the sum all X_i values at each node changes at a very slow pace. Therefore, as we will also demonstrate in Chapter 6, this simple way of maintaining the sum performs well and does not affect the scalability of the methods following this policy. It is straightforward to show that the estimated PageRank scores of the extended FIP algorithm are sharply concentrated around the true scores, similar to the case of the original FIP algorithm [Avr+07].

[1] Maintenance of $\sum_{i=1}^{n} X_i$ is not necessary if we are only interested in the relative web-pages' order.

Even though the extended FIP algorithm overcomes the two main constraints of FIP concerning the application domain, it still requires from

the participating nodes to exchange detailed information regarding the random walks, such that these can be fully reconstructed during re-routing. As we will also demonstrate experimentally (cf. Chapter 6), the amount of this data is significant. This contributes not only to the increased memory requirements of the algorithm (which is clearly a limitation when handling web-scale data), but also to an increase of the overall network cost of the algorithm. In the next Chapter, we show how DSPM lifts these requirements.

5 Distributed Stochastic PageRank Maintenance

5.1 Overview

The estimation of the PageRank vector with limited information available about the propagated random-walk paths is not a challenge on static web graphs, since random walks never need to be retrieved (for example, consider the algorithms proposed in [Avr+07]). However, on real-world dynamic web graphs, adding or removing links between the web-pages requires re-directing some previously executed random walks, as discussed in the previous chapters.

Very briefly, the Distributed Stochastic PageRank Maintenance (DSPM) approach re-routes the already propagated standard random walks by constructing new random walks from the redirection point (we refer to these as *negative random walks*, as discussed in Chapter 4). The challenge of this approach is twofold. First, in the absence of the previously conducted random walks, we need to find out the number of negative and standard random walks that need to be initiated after each update, such that the expected visit counts at the web-pages remain proportional to their PageRank scores. This, as we will show in the following, is not a trivial task, mainly due to the cycles, which are frequent in the web graph. A second challenge is to reduce the variance of this approach.

We now describe the crawling model targeted by DSPM, and explained how DSPM is integrated. The crawling network is composed of a set of nodes, scattered around the world (much like a P2P network). Nodes communicate via message passing. A distributed crawler runs on top of this network, assigning URLs to nodes in a consistent manner (e.g. via consistent hashing on the domain names). Approaches focused specifically on streaming graph partitioning may also be used, such as the ones proposed in [SK12]. Due to the physical distribution of the nodes, *communication between them is expensive and should be reduced*. Additionally, the crawling process may

involve low-cost commodity machines *possessing a limited amount of memory*, thus memory requirements should also be reduced. For the purpose of PageRank, each node maintains the set of crawled web-pages, and, for each web-page, the set of its out-going links. Therefore, both the web graph and the PageRank computation are fully distributed. The particular crawling model has an interesting property: it does not require a shared-memory architecture. This makes the model widely applicable, even over modern elastic architectures such as [Wal08]. Note however that this (minimal-requirements) crawling model is not a prerequisite for the functionality of DSPM. The algorithm would also offer substantial cost reduction when executed over P2P networks, in clouds charging based on network resources, and generally in any resource-constrained infrastructure.

Unlike the aforementioned extensions of the FIP method, the DSPM approach enables the updates over the Monte-Carlo method by minimizing the dependency on the knowledge of previously conducted random walks, thus decreasing both the transfer volume and the memory requirements. In this Chapter, we will present the bits and pieces of this contribution, by providing both the mathematical analysis establishing it, as well as optimizations improving its efficiency.

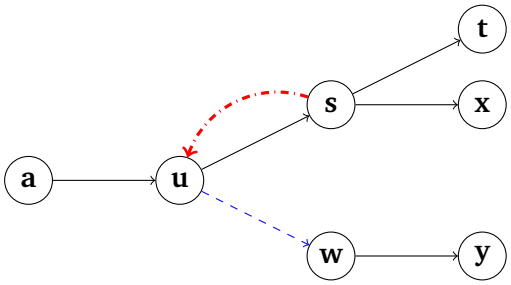
5.2 The DSPM approach

5.2.1 Initialization phase

The initialization phase is useful when there already exists a crawled subset of the web graph, e.g. from a previous crawl. This phase is a direct distributed implementation of the Monte-carlo algorithm for static datasets stopping at dangling nodes [Avr+07], using message passing. Let $\mathcal{P}(n)$ denote all web-pages assigned to node n, and $\mathcal{N}(p)$ the node responsible for page **p**. For the purpose of DSPM, nodes need to maintain only the set of outgoing links and a single counter (the visit count) for each assigned web-page. Then, each node n_x initiates R random walks from each web-page with termination probability ϵ, and increases the visit counts of all visited web-pages in $\mathcal{P}(n_x)$ by one. Whenever a random walk reaches a page **u** that does not belong to $\mathcal{P}(n_x)$, a message is sent to the node responsible for **u**. Notice that all decisions taken by the above algorithm for continuing or terminating the random walk are identical to the ones taken by [Avr+07] (assuming the same random seeding). Therefore, the visit counts (and the PageRank vector) generated by the end of this process are identical to the ones of [Avr+07].

5.2.2 Handling of updates

At first glance, it may seem straightforward to decide about the number of re-directions needed in case of an update edge, even in the absence of the stored random walk segments. Exactly as the FIP method examines (after an update $\mathbf{u} \rightarrow \mathbf{w}$) each transition of $\mathbf{u}$ with a re-direction probability of $\frac{1}{outDeg(\mathbf{u})}$, it may seem feasible to emulate this re-routing process by examining each visit count of $\mathbf{u}$ with the same probability and re-direct the resulting number of walks through the new edge (following the notion of negative random walks we have already introduced).



(a) Two simple graph instances, G_1 lacking $\mathbf{s} \rightarrow \mathbf{u}$ link and G_2 containing it. On both instances, we consider that $\mathbf{u} \rightarrow \mathbf{w}$ arrives as an update edge.

Graph G_1 without $\mathbf{s} \longrightarrow \mathbf{u}$	Graph G_2 with $\mathbf{s} \longrightarrow \mathbf{u}$
$\underbrace{\mathbf{u} \longrightarrow}_{\frac{1}{outDeg(\mathbf{u})}} \mathbf{s} \longrightarrow \mathbf{t}$	$\underbrace{\mathbf{u} \longrightarrow}_{\frac{1}{outDeg(\mathbf{u})}} \mathbf{s} \longrightarrow \underbrace{\mathbf{u} \longrightarrow}_{\frac{\left(1-\frac{1}{outDeg(\mathbf{u})}\right)}{outDeg(\mathbf{u})}} \mathbf{s}$
$\underbrace{\mathbf{u} \longrightarrow}_{\frac{1}{outDeg(\mathbf{u})}} \mathbf{s} \longrightarrow \mathbf{t}$	$\underbrace{\mathbf{u} \longrightarrow}_{\frac{1}{outDeg(\mathbf{u})}} \mathbf{s} \longrightarrow \underbrace{\mathbf{u} \longrightarrow}_{\frac{\left(1-\frac{1}{outDeg(\mathbf{u})}\right)}{outDeg(\mathbf{u})}} \mathbf{s} \longrightarrow \underbrace{\mathbf{u} \longrightarrow}_{\frac{\left(1-\frac{1}{outDeg(\mathbf{u})}\right)^2}{outDeg(\mathbf{u})}} \mathbf{s} \longrightarrow \mathbf{t}$
$\underbrace{\mathbf{u} \longrightarrow}_{\frac{1}{outDeg(\mathbf{u})}} \mathbf{s} \longrightarrow \mathbf{x}$	
$\underbrace{\mathbf{u} \longrightarrow}_{\frac{1}{outDeg(\mathbf{u})}} \mathbf{s}$	
$\underbrace{\mathbf{u} \longrightarrow}_{\frac{1}{outDeg(\mathbf{u})}} \mathbf{s} \longrightarrow \mathbf{t}$	

(b) A possible distribution of visits to walks passing through $\mathbf{u}$ in both G_1, G_2 graph instances. Each step passing through $\mathbf{u}$ is annotated with its respective *re-direction probability*, as concerns the FIP method.

Fig. 5.1 A careful examination of the re-direction probabilities of the state-of-the-art streaming PageRank method.

Figure 5.1 illustrates the reason why this policy would not lead to an unbiased estimator of the number of re-directions required. We consider a simple example of a $\mathbf{u} \rightarrow \mathbf{w}$ update edge and two graph instances, G_1

lacking $\mathbf{s} \rightarrow \mathbf{u}$ link and G_2 containing it. Figure 5.1b depicts a possible distribution of walks and visits of $\mathbf{u}$ in both graph instances, along with the respective re-direction probability of each transition. It is apparent that the cycles existing in a given graph directly affect these re-direction probabilities. As concerns a specific random walk, each subsequent visit to a web-page $\mathbf{u}$ will only be considered for re-direction *iff a previous occurrence of* $\mathbf{u}$ *has not already been selected* for re-routing from its respective point. Thus, we highlight that the task of real-time re-routing of random walks with limited information available about their original paths, is challenging and not straightforward at all.

Driven by this challenge, we propose Algorithm 1 and prove that it outputs an unbiased estimator of the PageRank vector. We formulate a connection between a web-page's return probability (i.e. the probability that a random walk visiting a web-page returns to it) and the expected number of random walk re-directions required in order to update the PageRank scores, in the case of a $\mathbf{u} \rightarrow \mathbf{w}$ update.

Algorithm 1 Streaming algorithm handling an update edge $\mathbf{u} \rightarrow \mathbf{w}$

Input: $\mathbf{u} \rightarrow \mathbf{w}, X_{\mathbf{u}}, outDeg(\mathbf{u}), d_{\mathbf{u}}, X_{FIP_{\mathbf{u}}}$
Output: $X_{\mathbf{v}} \forall \mathbf{v} \in V$

$\tilde{R}_{\mathbf{u}} = 1 - d_{\mathbf{u}}/X_{FIP_{\mathbf{u}}}$
$prob \leftarrow \Big((1-\epsilon)(1-\tilde{R}_{\mathbf{u}})\Big) / \Big(outDeg(\mathbf{u})(1-\tilde{R}_{\mathbf{u}})+1\Big)$
$i \leftarrow 0, ctr \leftarrow 0$
while $i < X_{\mathbf{u}}$ **do**
    Generate a uniformly random number $\alpha \in [0,1]$
    if $\alpha < prob$ **then**
        $ctr \leftarrow ctr + 1$
    $i \leftarrow i + 1$
$\mathcal{RW}_{Neg}(\mathbf{u}, ctr)$
Add new edge: $\mathbf{u} \rightarrow \mathbf{w}$
$\mathcal{RW}(\mathbf{w}, ctr)$

$X_{\mathbf{u}}$ denotes the visit count of web-page $\mathbf{u}$, $\tilde{R}_{\mathbf{u}}$ denotes an estimator of the probability that a random walk reaching $\mathbf{u}$ returns to $\mathbf{u}$ after a finite number of steps and $outDeg(\mathbf{u})$ the number of out-going links of $\mathbf{u}$. An estimate for $\tilde{R}_{\mathbf{u}}$ is maintained by emulating a single out of the R random walks of our approach, using the extension of the FIP algorithm, proposed in Chapter 4. We exploit the knowledge of those FIP-based walk segments, by using their visits as samples of the event of returning back to a web-page, as a result of a graph's cycle. More precisely, if $d_{\mathbf{u}}$ denotes the number of distinct FIP walks that go through a web-page $\mathbf{u}$ and $X_{FIP_{\mathbf{u}}}$ the total visits that those walks have added to $X_{\mathbf{u}}$, then: $\tilde{R}_{\mathbf{u}} = 1 - d_{\mathbf{u}}/X_{FIP_{\mathbf{u}}}$. The fact that this is an unbiased estimator of a walk' s return probability is trivial,

as it is an equivalent formulation to the division of the returning visits to $\mathbf{u}$ ($X_{FIP_{\mathbf{u}}} - d_{\mathbf{u}}$) by the total visits of those FIP-based walks to $\mathbf{u}$ ($X_{FIP_{\mathbf{u}}}$). The $d_{\mathbf{u}}$ term may obviously be greater than one (thus providing our estimator with more samples), as walks initiated from different web-pages may pass through a specific web-page. Notice that apart from the single FIP-based walk initiated from each web-page, the rest $R-1$ walks do not transfer any information about their paths or origin. Since those random walk segments do not need to be linked (this property stems from the stateless property of random walks), the messages corresponding to them contain only the url of the message's receiver (or a unique page id, if this is made available to the algorithm).

The following theorem mathematically establishes our result and constitutes our main theoretical contribution:

Theorem 5.2.1. *If $\tilde{\pi}_{\boldsymbol{v}}$ denotes our PageRank estimator for an arbitrary web-page $\boldsymbol{v}$, then it holds that: $E[\tilde{\pi}_{\boldsymbol{v}}] = \pi_{\boldsymbol{v}}$.*

Proof. At first, we decompose the visits comprising the visit count of each web-page as follows:

$$X_{\mathbf{u}} = X_{\mathbf{u}I} + X_{\mathbf{u}C} \tag{5.1}$$

$X_{\mathbf{u}I}$ corresponds to the sum of initial steps of the walks passing above $\mathbf{u}$ (i.e. the first R steps from the walks starting from $\mathbf{u}$ plus the first occurrences of $\mathbf{u}$ in walk segments propagated due to links of the form $(\mathbf{y}, \mathbf{u})$). $X_{\mathbf{u}C}$ reflects the visits to $\mathbf{u}$ which originate from the cycles in which $\mathbf{u}$ participates. We define the indicator variable $R_{\mathbf{u}}$ as follows:

$$R_{\mathbf{u}} = \begin{cases} 1, & \text{if a walk reaching } u \text{ returns to } u. \\ 0, & \text{otherwise.} \end{cases}$$

We assume a graph without self-loops, as they are counter-intuitive to the notion of PageRank's endorsement strategy [CM08]. We define the return probability to $\mathbf{u}$ as the sum of the probabilities of the tours connecting $\mathbf{u}$ to itself as follows:

$$Prob(R_{\mathbf{u}} = 1) = \sum_{t \in \{\mathbf{u} \rightsquigarrow \mathbf{u}\}} (1-\epsilon)^{L(t)} \prod_{i=0}^{L(t)-1} \frac{1}{outDeg(\mathbf{y_i})}, \tag{5.2}$$

where $L(t)$ denotes the length of each tour t of the form $\mathbf{y_0} \rightarrow \mathbf{y_1} \rightarrow \cdots \rightarrow \mathbf{y_{L(t)}}$ and $outDeg(\mathbf{y_i})$ denotes the out-degree of web-page $\mathbf{y_i}$. We

additionally define $\mathbf{P_{v,u}}$ as the probability of reaching $\mathbf{u}$, given the event of reaching a web-page $\mathbf{v}$ pointed by $\mathbf{u}$:

$$\mathbf{P_{v,u}} = \sum_{t \in \{\mathbf{v} \rightsquigarrow \mathbf{u}\}} (1-\epsilon)^{L(t)-1} \prod_{i=1}^{L(t)-1} \frac{1}{outDeg(\mathbf{y_i})} \tag{5.3}$$

By exploiting the fact that the first step of each tour in Equation 5.2 corresponds to a step through an out-going edge of $\mathbf{u}$, we have:

$$Prob(R_{\mathbf{u}} = 1) = \frac{1-\epsilon}{outDeg(\mathbf{u})} \sum_{\mathbf{v} \in N_{\mathbf{u}}} \mathbf{P_{v,u}},$$

where $N_{\mathbf{u}}$ is the set consisting of the web-pages pointed by $\mathbf{u}$. Thus: $E[R_{\mathbf{u}}] = \frac{1-\epsilon}{outDeg(\mathbf{u})} \sum_{\mathbf{v} \in N_{\mathbf{u}}} \mathbf{P_{v,u}}$. As a result, the expected number of first returns to $\mathbf{u}$ is as follows:

$$E[\sum_{i=1}^{X_{\mathbf{u}I}} R_{\mathbf{u}}] = \sum_{i=1}^{X_{\mathbf{u}I}} E[R_{\mathbf{u}}] = X_{\mathbf{u}I} \frac{1-\epsilon}{outDeg(\mathbf{u})} \sum_{\mathbf{v} \in N_{\mathbf{u}}} \mathbf{P_{v,u}}$$

Therefore, the expected second returns to $\mathbf{u}$ are:

$$X_{\mathbf{u}I} \left(\frac{1-\epsilon}{outDeg(\mathbf{u})} \sum_{\mathbf{v} \in N_{\mathbf{u}}} \mathbf{P_{v,u}} \right)^2$$

Thus, the expected total returns to $\mathbf{u}$ (i.e. the total visits originating from cycles) are:

$$\begin{aligned} X_{\mathbf{u}C} &= X_{\mathbf{u}I} \sum_{i=1}^{\infty} \left(\frac{1-\epsilon}{outDeg(\mathbf{u})} \sum_{\mathbf{v} \in N_{\mathbf{u}}} \mathbf{P_{v,u}} \right)^i \\ &= X_{\mathbf{u}I} \left(\frac{\frac{1-\epsilon}{outDeg(\mathbf{u})} \sum_{\mathbf{v} \in N_{\mathbf{u}}} \mathbf{P_{v,u}}}{1 - \frac{1-\epsilon}{outDeg(\mathbf{u})} \sum_{\mathbf{v} \in N_{\mathbf{u}}} \mathbf{P_{v,u}}} \right), \end{aligned}$$

through the geometric series, by assuming that:

$$\left| \frac{1-\epsilon}{outDeg(\mathbf{u})} \sum_{\mathbf{v} \in N_{\mathbf{u}}} \mathbf{P_{v,u}} \right| < 1$$

Thus, by replacing into Equation 5.1, we have:

$$X_{\mathbf{u}} = \frac{X_{\mathbf{u}I}}{1 - \frac{1-\epsilon}{outDeg(\mathbf{u})} \sum_{\mathbf{v} \in N_{\mathbf{u}}} \mathbf{P_{v,u}}} \tag{5.4}$$

We now define the two settings involved in our proof: we denote by G_1 a graph instance containing no self-loops and no sink web-pages. Let G_2 be the instance G_1 by adding the edge $(\mathbf{u}, \mathbf{w})$ to it. Let $outDeg(\mathbf{u})$ denote the out-degree of $\mathbf{u}$ on graph G_2 and $N_{\mathbf{u}}^{'}$ the set of out-going neighbours of $\mathbf{u}$ by excluding web-page $\mathbf{w}$.

1. The first setting assumes the execution of the Monte-Carlo static algorithm on graph G_2, by considering though the link $(\mathbf{u}, \mathbf{w})$ as inactive, i.e. the flow of visits crossing $(\mathbf{u}, \mathbf{w})$ is considered as pending until this link becomes active. This event is assumed to happen at time $t+1$. Because of the fact that the visits to $\mathbf{u}$ originating from cycles could have returned to $\mathbf{u}$ from a webpage $\mathbf{v}$ other than $\mathbf{w}$ (as the link $(\mathbf{u}, \mathbf{w})$ is still inactive), the visit count of $\mathbf{u}$ at time t is as follows:
$$X_{\mathbf{u}}^{'t} = \frac{X_{\mathbf{u}I}}{1 - \frac{1-\epsilon}{outDeg(\mathbf{u})} \sum\limits_{\mathbf{v} \in N_{\mathbf{u}}^{'}} \mathbf{P}_{\mathbf{v},\mathbf{u}}} \tag{5.5}$$

2. The second setting assumes the execution of the same static algorithm on graph G_1, as part of the DSPM initialization procedure. The on-line phase of our algorithm starts with the edge arrival of $(\mathbf{u}, \mathbf{w})$ at time $t+1$. The visit count of $\mathbf{u}$ at time t is as follows:
$$X_{\mathbf{u}}^{t} = \frac{X_{\mathbf{u}I}}{1 - \frac{1-\epsilon}{outDeg(\mathbf{u})-1} \sum\limits_{\mathbf{v} \in N_{\mathbf{u}}^{'}} \mathbf{P}_{\mathbf{v},\mathbf{u}}} \tag{5.6}$$

The main difference between the previous two amounts lies in the fact that in the static approach' s setting, the probability of choosing each one of the neighbours of $\mathbf{u}$ apart from $\mathbf{w}$ is equal to $\frac{1}{outDeg(\mathbf{u})}$. However, during the propagation of steps in the off-line phase of our approach, the probability of going through each edge of $\mathbf{u}$ is $\frac{1}{outDeg(\mathbf{u})-1}$, as web-page $\mathbf{u}$ has not received the $(\mathbf{u}, \mathbf{w})$ link since time $t+1$. By combining equations 5.5 and 5.6, we have:
$$X_{\mathbf{u}}^{'t} = \frac{1 - \frac{1-\epsilon}{outDeg(\mathbf{u})-1} \sum\limits_{\mathbf{v} \in N_{\mathbf{u}}^{'}} \mathbf{P}_{\mathbf{v},\mathbf{u}}}{1 - \frac{1-\epsilon}{outDeg(\mathbf{u})} \sum\limits_{\mathbf{v} \in N_{\mathbf{u}}^{'}} \mathbf{P}_{\mathbf{v},\mathbf{u}}} X_{\mathbf{u}}^{t} \tag{5.7}$$

The amount of re-directions required on our setting is equal to the expected number of steps that will be propagated through the edge $(\mathbf{u}, \mathbf{w})$ on the first setting at time $t+1$, when this edge becomes active. Since each of the $X_{\mathbf{u}}^{'t}$ will be propagated through $\mathbf{u}$ with a probability of $1-\epsilon$ and will

choose to visit **w** with a probability of $\frac{1}{outDeg(\mathbf{u})}$, the expected amount of re-directions required on our setting is equal to:

$$\frac{1-\epsilon}{outDeg(\mathbf{u})}X'^{t}_{\mathbf{u}} = \frac{1-\epsilon}{outDeg(\mathbf{u})-1}\frac{outDeg(\mathbf{u})-1-(1-\epsilon)\sum\limits_{\mathbf{v}\in N'_{\mathbf{u}}}\mathbf{P}_{\mathbf{v},\mathbf{u}}}{outDeg(\mathbf{u})-(1-\epsilon)\sum\limits_{\mathbf{v}\in N'_{\mathbf{u}}}\mathbf{P}_{\mathbf{v},\mathbf{u}}}X^{t}_{\mathbf{u}}$$

At this point, we exploit the fact that, as concerns our setting, the expected value of the indicator variable $R_{\mathbf{u}}$ is equal to: $E[R_{\mathbf{u}}] = \frac{1-\epsilon}{outDeg(\mathbf{u})}\sum\limits_{\mathbf{v}\in N'_{\mathbf{u}}}\mathbf{P}_{\mathbf{v},\mathbf{u}}$. Thus, we end up to our main result about the expected number of re-directions required in our real-time setting for each update edge $(\mathbf{u},\mathbf{w})$:

$$\frac{(1-\epsilon)(1-\tilde{R}_u)}{(outDeg(\mathbf{u})-1)(1-\tilde{R}_u)+1}X^{t}_{\mathbf{u}}, \tag{5.8}$$

where $\tilde{R}_u$ is our estimator for a walk' s return probability. Since during the execution of our algorithm we choose each of the visits of **u** to initiate a re-direction through the new edge $(\mathbf{u},\mathbf{w})$ with a probability of $\frac{(1-\epsilon)(1-\tilde{R}_u)}{(outDeg(\mathbf{u})-1)(1-\tilde{R}_u)+1}$, then the expression in 5.8 provides the expected amount of re-directions initiated from the DSPM approach. After having provided the required expected number of re-directions needed in case of an update edge, it is straightforward that the expected PageRank on an arbitrary web-page will remain the same on the two settings at time $t+1$, since in expectation the random walks initiated from a web-page are evenly distributed to the web-page's out-going links. □

Note that our algorithm can handle at once multiple edges sharing the same source (a feature incorporated to our implementation) but for simplicity and brevity, we focus our analysis on the single-edge update algorithm.

5.3 Enhancements

5.3.1 Aggregate data exchange

Our solution does not prerequisite the explicit storage of any random walk segment, apart from the ones used for the estimation of the return probability. The significance of avoiding this requirement is that nodes no longer need to exchange data for keeping track of the full random walk segments; instead, nodes can exchange compact, aggregate data. For example, instead of having to transmit k steps from the node holding page **a** to the node holding page **b**, along with their different random walk

identifiers and positions at these random walk segments, it is sufficient to send a single message $< \mathbf{b}, k >$, from the node responsible for page **a** to the node holding **b**. Another direct consequence of our method is the lack of need to store the random walk segments explicitly in the memory of the nodes responsible for the web-pages present in the graph.

5.3.2 Variance reduction

Our policy of minimizing the number of full random walk segments exchanged poses a fundamental challenge concerning the accuracy of negative random walks, because of the lack of explicit knowledge of the paths of their regular counterparts. We devised an enhancement that enables us to accurately and efficiently reject a previously propagated regular random walk without being aware of the exact path it followed. This feature is feasible by solely exploiting pieces of knowledge that are available to the algorithm at no cost (i.e. without the need of any additional data transfer).

To this direction, we track the regular steps passing through each web page and store them in a Map structure. For each web-page **u** of the network, this structure maintains the destination of all regular random walk steps passed through **u**, in an aggregate manner. For example, consider a web-page **u**, pointing to web-pages **x** and **y**. If out of the 10 steps continuing their route through **u**, 6 of them have chosen **x** and 4 of them have chosen **y** as their next step, our structure for web-page **u** would have the form: $< \mathbf{x}, 6 >, < \mathbf{y}, 4 >$. By enriching our knowledge of the route followed by the regular walks, for each negative random walk in the update phase, we choose each step with probability proportional to the steps sent through each neighbour of the current web-page. This means that in the aforementioned example, as concerns a negative step passing through **u**, we would choose **x** with a probability of 0.6 and **y** with a probability of 0.4 as the next negative step (decreasing the visit count). This procedure is equivalent to propagating a walk on a weighted graph.

5.4 Analysis

As long as the number of re-directions initiated on behalf of our method is proven to be the same as from the algorithm in [Bah+10], the worst-case message complexity of DSPM is the same as in the FIP method. Under the random arrival model of update edges [Bah+10], this cost is on the order

of $O(\frac{nRlnm}{\epsilon^2})$. As concerns the space complexity of our method, it is on the order of $O(m + n + \frac{n}{\epsilon})$, whereas the FIP extensions maintain space on the order of $O(m + n + \frac{nR}{\epsilon})$, where m stands for the out-links' maintenance and n for the maintenance of the visit counts. The third term corresponds to the space needed to store the complete random walk paths (which in our case is limited to the storage of the paths used for the estimation of the return probability).

6 Experimental results

6.1 Setup

Our experiments were executed on a server running Ubuntu SMP, sharing 31 Intel Xeon processing units at 2.1 GHz and 28 GB of available main memory.

We conducted experiments on real data, in order to evaluate the communication efficiency, the scalability and the memory footprint against both distributed extensions of the FIP method presented in Chapter 4. The datasets used in our evaluation are publicly available in https://snap.stanford.edu/, and are either real snapshots of portions of the web graph or portions from a social network friendship graph. A random permutation of each dataset is performed before the execution. We considered a 10% portion of each dataset as present during each algorithm's start, in order to emulate their initialization phase. The assumption that the original FIP method is aware of the whole set of web-pages beforehand has also been made. Despite the fact that this assumption is not necessarily realistic under the streaming and distributed model of computation we consider, the original FIP algorithm could not operate correctly otherwise. A description of the employed datasets follows:

- BerkStan: The web-pages belong to the berkeley.edu and stanford.edu domains. The dataset contains 685230 web-pages and 7600595 links.

- Google: The dataset contains general web-pages (not from a particular domain). The data was released in 2002 by Google as part of Google Programming Contest, and contains a total of 875713 web-pages and 5105039 links.

- LiveJournal: The dataset contains a friendship graph from the LiveJournal social network. It contains a total of 4847571 entities and 68993773 links.

6.2 Accuracy & Efficiency

The Spearman rank's correlation coefficient (ρ) was chosen as a measure of each method's accuracy, using as a ground truth the ideal PageRank scores extracted by the accurate, yet centralized, Power Iteration method. The coefficient assesses how well the relationship between two variables can be described using a monotonic function, and takes values in the interval [-1,1], with 1 denoting perfect (linear) relationship. The inter-node transfer volume of the messages involved in each computation is considered as the measure of efficiency, as the intra-node communication is negligible compared to the cost of transferring data in a distributed network.

The plots in Figures 6.1, 6.2, 6.3 depict the Spearman coefficient against the required transfer volume for the three algorithms on all datasets, for different R values $(5, 10, 15, 20, 25)$. These experiments were also executed for varying number of web-pages contained in each node's subgraph, so as to examine the scalability potential of the methods under consideration.

The superiority of the DSPM approach in comparison to the extensions of the FIP method is evident, particularly when high Spearman coefficients (i.e. good PageRank approximations) are desired. In particular, by comparing the transfer volume required for the highest possible same level of accuracy, we notice a $6x - 24x$ improvement of the DSPM method against the original FIP' s extension. Additionally, a $2.2x - 3.6x$ improvement of the DSPM against the optimized FIP approach is observed.

A useful remark is that the efficiency of the original FIP approach is significantly deteriorated when the underlying graph available during the initialization phase contains a non-negligible portion of dangling (sink) web-pages. This behaviour is the consequence of performing random jumps from sink web-pages that will eventually be deleted when those pages acquire at least one out-going edge, during the online phase. This case is particularly obvious in the Google dataset, which totally contains a 15% portion of dangling web-pages.

As concerns the DSPM approach, its improvement against the optimized FIP method is caused by the minimization of the FIP-based walks involved, which require the exact information about each walk segment to be exchanged, as well as by the ability to aggregate the steps of the rest $R - 1$ walks. As concerns the scalability potential, we notice that the cost with all partitioning schemes is approximately the same, thus all methods are

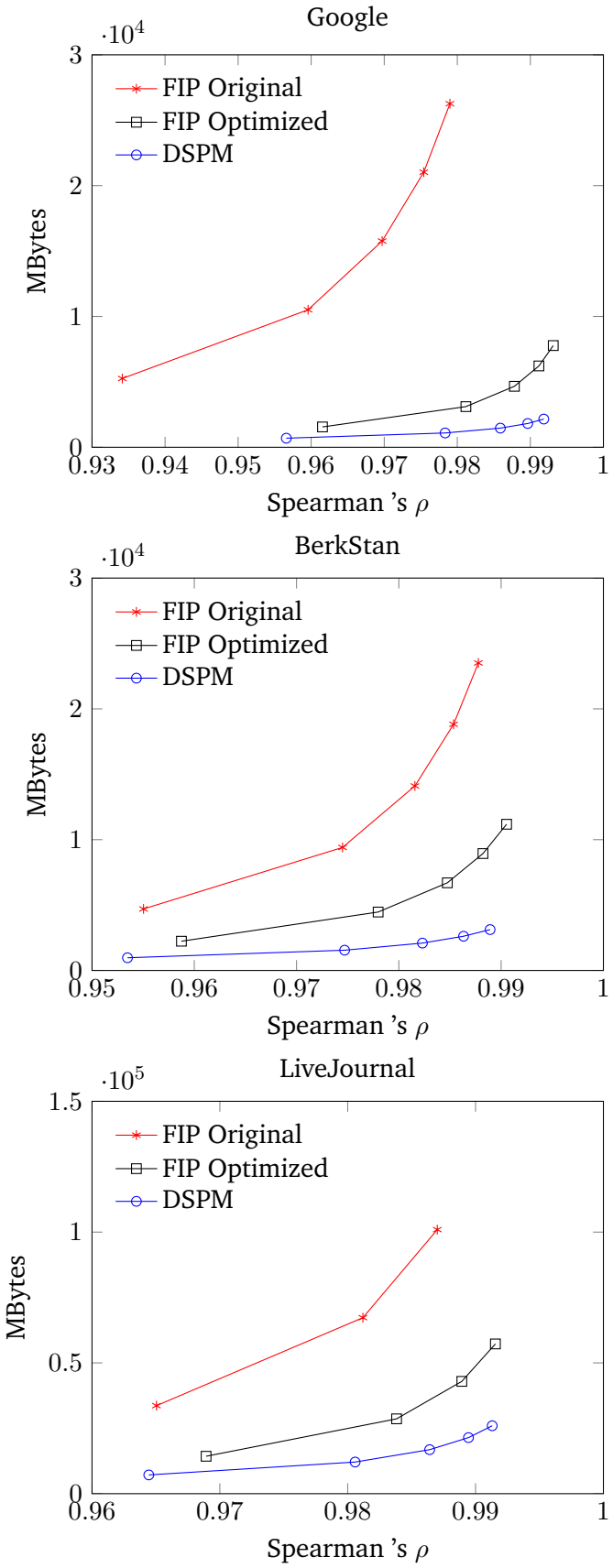


Fig. 6.1 Spearman correlation coefficient versus the required useful transfer volume of FIP Original, FIP Optimized and DSPM, with $R \in \{5, 10, 15, 20, 25\}$. Each node is responsible for maintaining 1K webpages.

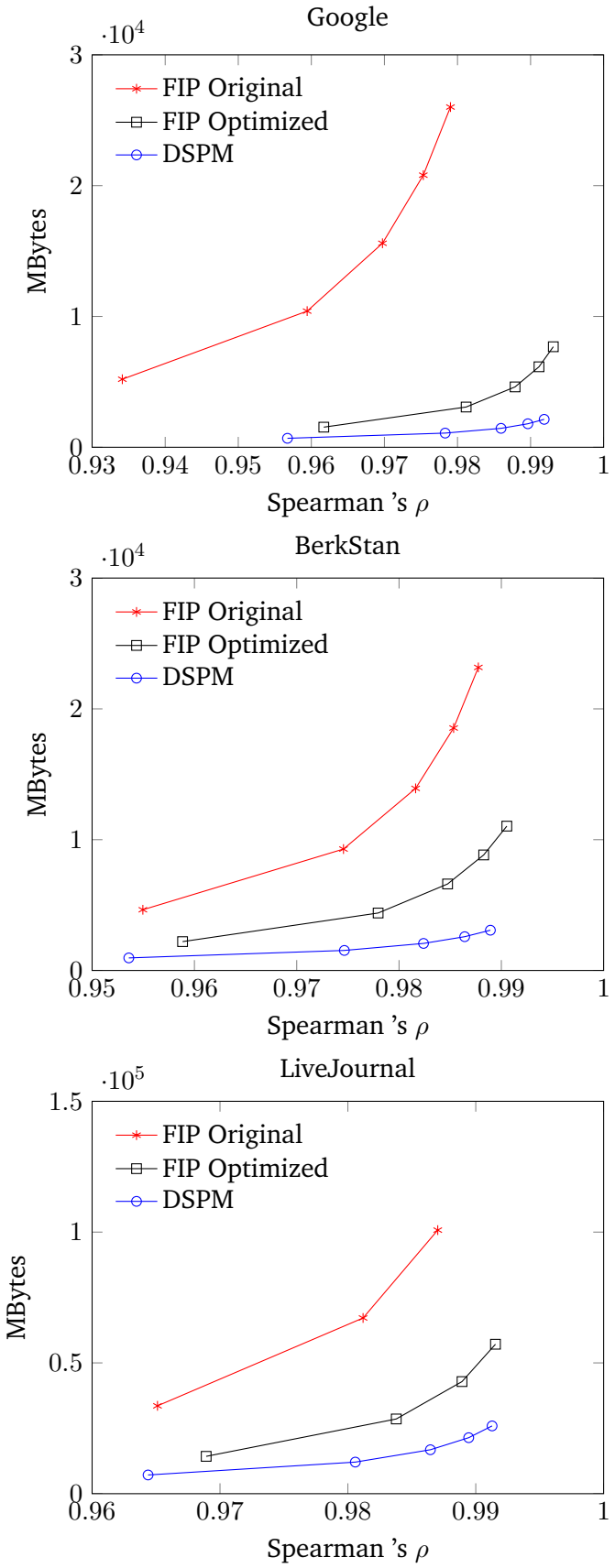


Fig. 6.2 Spearman correlation coefficient versus the required useful transfer volume of FIP Original, FIP Optimized and DSPM, with $R \in \{5, 10, 15, 20, 25\}$. Each node is responsible for maintaining 10K webpages.

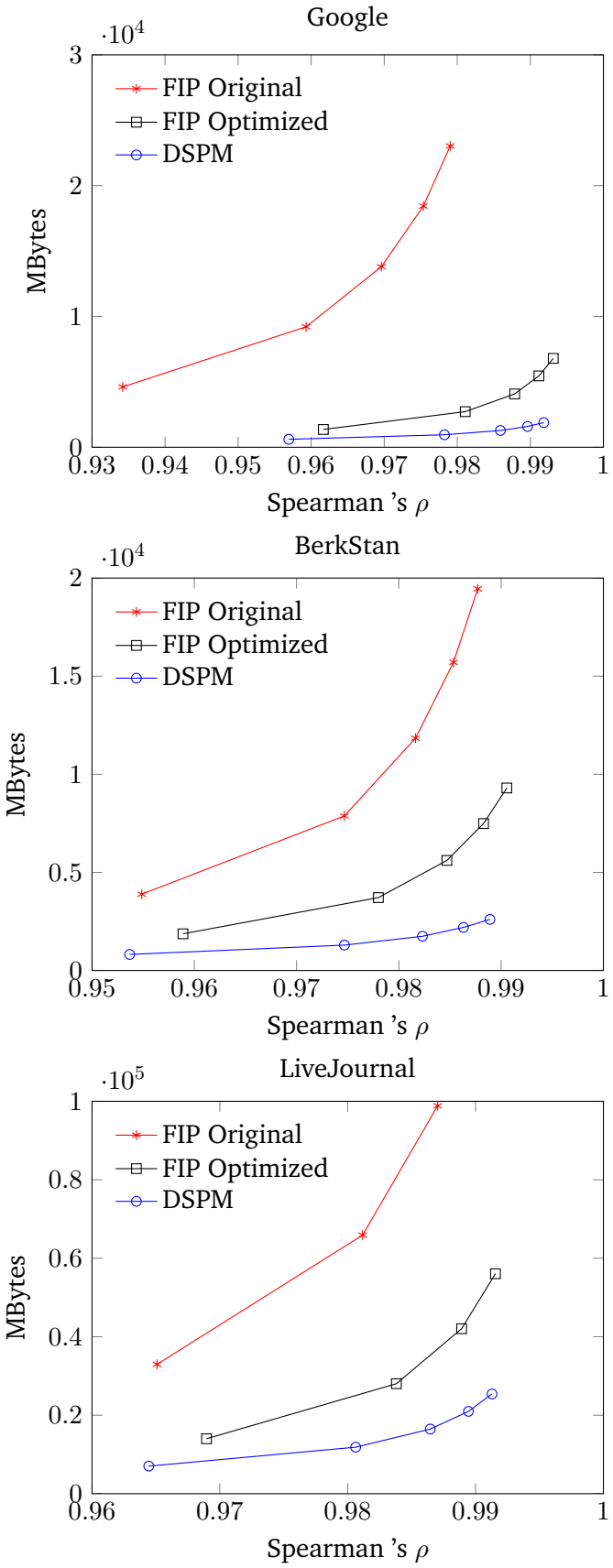


Fig. 6.3 Spearman correlation coefficient versus the required useful transfer volume of FIP Original, FIP Optimized and DSPM, with $R \in \{5, 10, 15, 20, 25\}$. Each node is responsible for maintaining 100K webpages.

considered scalable in terms of the number of web-pages present in each node' s sub-graph. Consequently, the main factor affecting their scalability potential is the number of walks initiated per node. Additionally, we remark that the policy (employed by both the optimized FIP method and the DSPM approach) of requesting each node to inform a coordinator when the local sum of its web-pages deviates substantially from the previously reported value, only incurs a negligible overhead to the total cost.

6.3 Memory requirements

We calculated the amount of memory required by all the methods under comparison, at the end of the real-time execution. The results are depicted in Figure 6.4 and correspond to the cumulative memory requirements of the whole network of web-pages. It is remarkable that the memory needs of DSPM remain almost the same (horizontal line), in spite of increasing the number of initiated walks per node. In particular, the gain of the DSPM approach against both FIP extensions exceeds an order of magnitude when improved PageRank accuracy is desired. Another point worth considering is that both FIP extensions fail to execute for increasing R values in the LiveJournal dataset, as a result of the lack of memory available in the server used for our experiments. These results establish the DSPM method as scalable and applicable to resource-constrained environments.

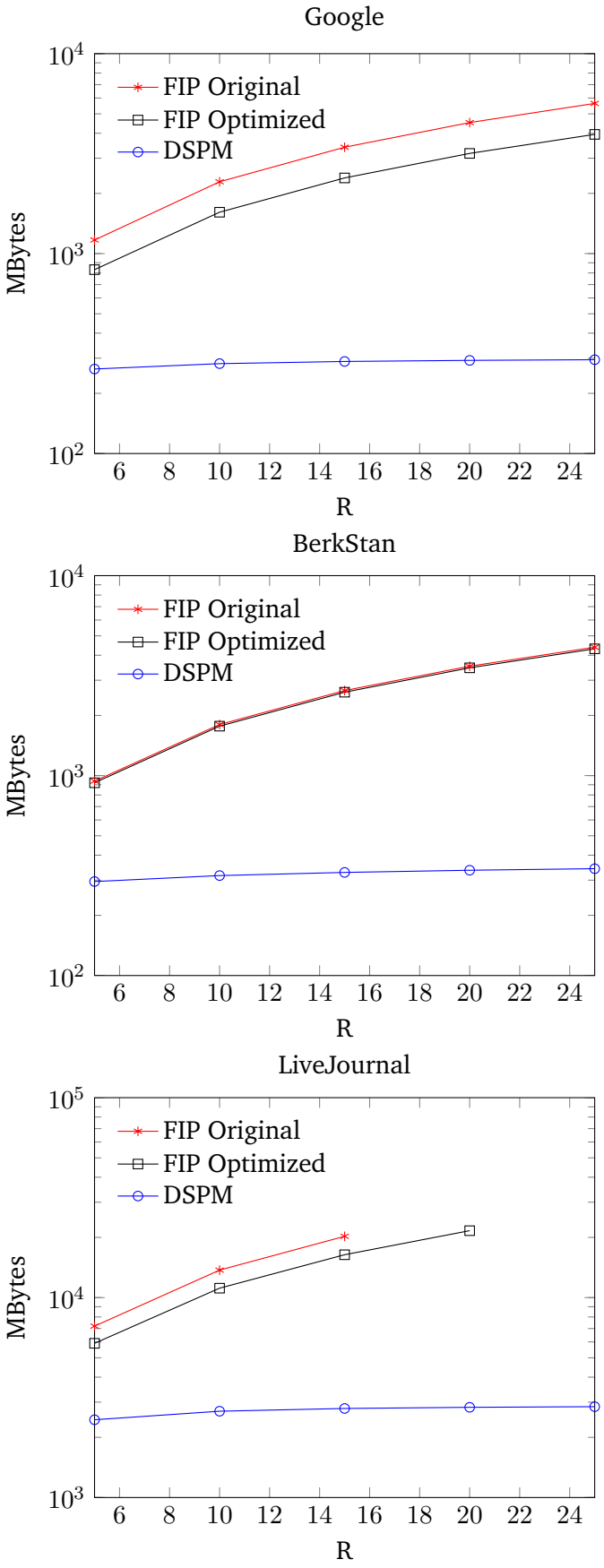


Fig. 6.4 Memory requirements of the FIP Original, FIP Optimized and DSPM approaches with $R \in \{5, 10, 15, 20, 25\}$, for the whole network

7 Conclusions and Future Work

We provided the first, to the best of our knowledge, PageRank algorithm focused on monitoring distributed graph streams, with the aim of minimizing its requirements in both the transfer volume and the memory of the nodes involved in the computation. We mathematically proved and experimentally established that storing all random walks explicitly is not a necessary route, in order to update a random-walk based model through an incremental algorithm in a fully distributed environment. We believe that our work will greatly influence many real-world massive scale social network analysis problems, as the real-time tracking of a shift in the authority of a distributed network could be utilized by social-influence measuring or epidemics prevention applications. Furthermore, we believe that it will pose an impact on the distributed monitoring of other random-walk based measures, such as betweenness centrality [New05]. Another promising research direction is the extension of this work to the continuous monitoring of the top-k PageRank values in a distributed environment.

Bibliography

[Ada+13] Lada A. Adamic, Christos Faloutsos, Theodore J. Iwashyna, B. Aditya Prakash, and Hanghang Tong. „Fractional Immunization in Networks“. In: *SDM*. 2013, pp. 659–667 (cit. on p. 2).

[Agg+11] Charu C Aggarwal, Arijit Khan, and Xifeng Yan. „On Flow Authority Discovery in Social Networks.“ In: *SDM*. 2011, pp. 522–533 (cit. on p. 2).

[And+06] Reid Andersen, Fan Chung, and Kevin Lang. „Local graph partitioning using pagerank vectors“. In: *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*. IEEE. 2006, pp. 475–486 (cit. on p. 2).

[Avr+07] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. „Monte Carlo Methods in PageRank Computation: When One Iteration is Sufficient“. In: *SIAM J. Numer. Anal.* 45.2 (Feb. 2007), pp. 890–904 (cit. on pp. 3, 10, 13, 14, 20, 23, 24).

[Bah+10] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. „Fast Incremental and Personalized PageRank“. In: *PVLDB* 4.3 (2010), pp. 173–184 (cit. on pp. 3, 4, 10, 13, 14, 19, 31).

[Bah+12] Bahman Bahmani, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. „PageRank on an evolving graph“. In: *KDD*. 2012, pp. 24–32 (cit. on p. 16).

[Ber05] Pavel Berkhin. „Survey: A Survey on PageRank Computing“. In: *Internet Mathematics* 2.1 (2005), pp. 73–120 (cit. on p. 13).

[Cal+08] Jérôme Callut, Kevin Françoisse, Marco Saerens, and Pierre Dupont. „Semi-supervised Classification from Discriminative Random Walks“. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Walter Daelemans, Bart Goethals, and Katharina Morik. Vol. 5211. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 162–177 (cit. on p. 8).

[Che+10] Wei Chen, Chi Wang, and Yajun Wang. „Scalable Influence Maximization for Prevalent Viral Marketing in Large-scale Social Networks“. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '10. Washington, DC, USA: ACM, 2010, pp. 1029–1038 (cit. on p. 2).

[CM08] Prasad Chebolu and Páll Melsted. „PageRank and the random surfer model“. In: *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2008, pp. 1010–1018 (cit. on p. 27).

[DS+14] Atish Das Sarma, Anisur Rahaman Molla, Gopal Pandurangan, and Eli Upfal. „Fast distributed PageRank computation“. In: *Theoretical Computer Science* (2014) (cit. on p. 14).

[Eld07] Lars Eldén. *Matrix Methods in Data Mining and Pattern Recognition*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007 (cit. on p. 8).

[FR04] Dániel Fogaras and Balázs Rácz. „Towards Scaling Fully Personalized PageRank“. In: *WAW*. 2004, pp. 105–117 (cit. on p. 13).

[Gar+13] Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. „Sketch-based Geometric Monitoring of Distributed Stream Queries“. In: *Proc. VLDB Endow.* 6.10 (Aug. 2013), pp. 937–948 (cit. on p. 1).

[IT10] H. Ishii and R. Tempo. „Distributed Randomized Algorithms for the PageRank Computation“. In: *Automatic Control, IEEE Transactions on* 55.9 (2010), pp. 1987–2002 (cit. on pp. 2, 14).

[LM03] Amy Nicole Langville and Carl Dean Meyer. „Survey: Deeper Inside PageRank“. In: *Internet Mathematics* 1.3 (2003), pp. 335–380 (cit. on p. 7).

[LM04] Amy Nicole Langville and Carl Dean Meyer. „Updating pagerank with iterative aggregation“. In: *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM. 2004, pp. 392–393 (cit. on pp. 3, 16).

[Lov93] László Lovász. *Random Walks on Graphs: A Survey*. 1993 (cit. on p. 9).

[Low+12] Yucheng Low, Danny Bickson, Joseph Gonzalez, et al. „Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud“. In: *Proc. VLDB Endow.* 5.8 (Apr. 2012), pp. 716–727 (cit. on p. 1).

[LY08] Dionysios Logothetis and Kenneth Yocum. „Ad-hoc data processing in the cloud“. In: *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1472–1475 (cit. on p. 3).

[McS05] Frank McSherry. „A uniform approach to accelerated PageRank computation“. In: *Proceedings of the 14th international conference on World Wide Web*. ACM. 2005, pp. 575–582 (cit. on p. 3).

[New05] Mark EJ Newman. „A measure of betweenness centrality based on random walks“. In: *Social networks* 27.1 (2005), pp. 39–54 (cit. on p. 41).

[Pag+99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Stanford InfoLab, 1999 (cit. on pp. 2, 7, 8, 14).

[Par+06] Josiane Xavier Parreira, Debora Donato, Sebastian Michel, and Gerhard Weikum. „Efficient and decentralized pagerank approximation in a peer-to-peer web search network“. In: *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment. 2006, pp. 415–426 (cit. on pp. 2, 14).

[RU11] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011 (cit. on p. 2).

[Sar+08] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. „Estimating PageRank on graph streams“. In: *PODS*. 2008, pp. 69–78 (cit. on pp. 3, 10, 14).

[SK12] Isabelle Stanton and Gabriel Kliot. „Streaming Graph Partitioning for Large Distributed Graphs“. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '12. Beijing, China: ACM, 2012, pp. 1222–1230 (cit. on pp. 1, 23).

[Spr+06] Neil Spring, Larry Peterson, Andy Bavier, and Vivek Pai. „Using PlanetLab for network research: myths, realities, and best practices“. In: *ACM SIGOPS Operating Systems Review* 40.1 (2006), pp. 17–24 (cit. on p. 3).

[Ton+08] Hanghang Tong, Spiros Papadimitriou, S Yu Philip, and Christos Faloutsos. „Proximity Tracking on Time-Evolving Bipartite Graphs.“ In: *SDM*. Vol. 8. 2008, pp. 704–715 (cit. on pp. 3, 16).

[Vit85] Jeffrey S. Vitter. „Random sampling with a reservoir“. In: *ACM Trans. Math. Softw.* 11.1 (Mar. 1985), pp. 37–57 (cit. on pp. 10, 14).

[Wal08] Edward Walker. „Benchmarking Amazon EC2 for high-performance scientific computing“. In: *Usenix Login* 33.5 (2008), pp. 18–23 (cit. on pp. 1, 3, 24).

[Win+12] Christof Winter, Glen Kristiansen, Stephan Kersting, et al. „Google Goes Cancer: Improving Outcome Prediction for Cancer Patients by Network-Based Ranking of Marker Genes.“ In: *PLoS Computational Biology* 8.5 (2012) (cit. on p. 2).

[Wu+08] Xindong Wu, Vipin Kumar, J Ross Quinlan, et al. „Top 10 algorithms in data mining“. In: *Knowledge and Information Systems* 14.1 (2008), pp. 1–37 (cit. on p. 2).

[Xia+13] Biao Xiang, Qi Liu, Enhong Chen, et al. „Pagerank with Priors: An Influence Propagation Perspective“. In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. IJCAI'13. Beijing, China: AAAI Press, 2013, pp. 2740–2746 (cit. on p. 2).