# VIRTUAL WEB FOR PAGERANK COMPUTING

---

A Dissertation

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

---

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

BO SONG

Dr. Xinhua Zhuang, Dissertation Supervisor

MAY 2018

The undersigned, appointed by the dean of the Graduate School, have examined the dissertation entitled

VIRTUAL WEB FOR PAGERANK COMPUTING

Presented by Bo Song,

A candidate for the degree of doctor of philosophy,

and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Xinhua Zhuang

Dr. Jeffery Uhlmann

Dr. Yunxin Zhao

Dr. Zhihai He

## ACKNOWLEDGEMENTS

I owe my deepest gratitude to my advisor, Prof. Xinhua Zhuang, who enlightened me on the research direction, provided me rigorous scientific training and endow me the ability of innovation. He offered me his greatest help with his talented ideas and rich experiences to make this work possible. He spent countless time in his office to revise my papers and works, and taught me how to do research correctly like a scientist. His kind encouragement, selfless support, and his work ethic will become the driving force of my future life.

I sincerely give my appreciation to my committee members, Dr. Jeffery Uhlmann, Dr. Yunxin Zhao and Dr. Zhihai He, for the knowledge they imparted during my Ph.D. study, and their valuable time and work to serve as my dissertation committee members.

I will give my special thanks to my wife, Jingfei Liu, for all her trust, support and patience during my time as a Ph.D. student.

# TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# ABSTRACT

The enormous size and fast-evolving nature of World-Wide-Web has been demanding an even more efficient PageRank updating algorithm. Web evolution may involve two kinds: (1) link structure modification; (2) page insertion/deletion. When the web evolution is restricted to only link insertion/deletion, we demonstrate the benefit of using the previous PageRank to initialize the current PageRank computation, theoretically and experimentally. When page insertion/deletion occurs, how to effectively use the previous PageRank information to facilitate the current PageRank computation has long been a challenge. To tackle the general case, a so-called "virtual web" is introduced through adding the inserted nodes to the previous web along with some specific "in-home" link structure, where in-links from the previous web and out-links to the previous web are excluded. Through the virtual web, we are able to work out a virtual initialization, which can be efficiently used to calculate the current PageRank. The introduced virtual initialization is "*unbiased*", that assumes least under available knowledge. The virtual web is then integrated with the Power-Iteration and Gauss-Southwell method to solve the node insertion/deletion problem, which are named as Virtual Web Power-Iteration (VWPI) method and Virtual Web Gauss-Southwell (VWGS) method, respectively. Further, we proposed an optimized approach based on VWGS method for updating node insertions. The experiment result shows that the VWGS algorithm significantly outperformed the conventional PageRank computation based on the original model. On the dataset Twitter-2010 with 42M nodes and 1.5B edges, for a perturbation of 400k node and 14 million link insertions plus deletions at one time, our algorithm is about 20 times faster on number of iterations and 3 times faster on running-time in comparison to the Gauss-Southwell method starting from scratch. On the soc-LiveJournal dataset with up to a 20% node insertion, the optimized VWGS method received another 28% gain comparing to the original VWGS method. To compare with the prior work proposed by Ohsaka et al. in [32], our method is 1800x faster per link insertion/deletion on the Twitter-2010 dataset under similar experiment environment.

**KEY WORDS:**

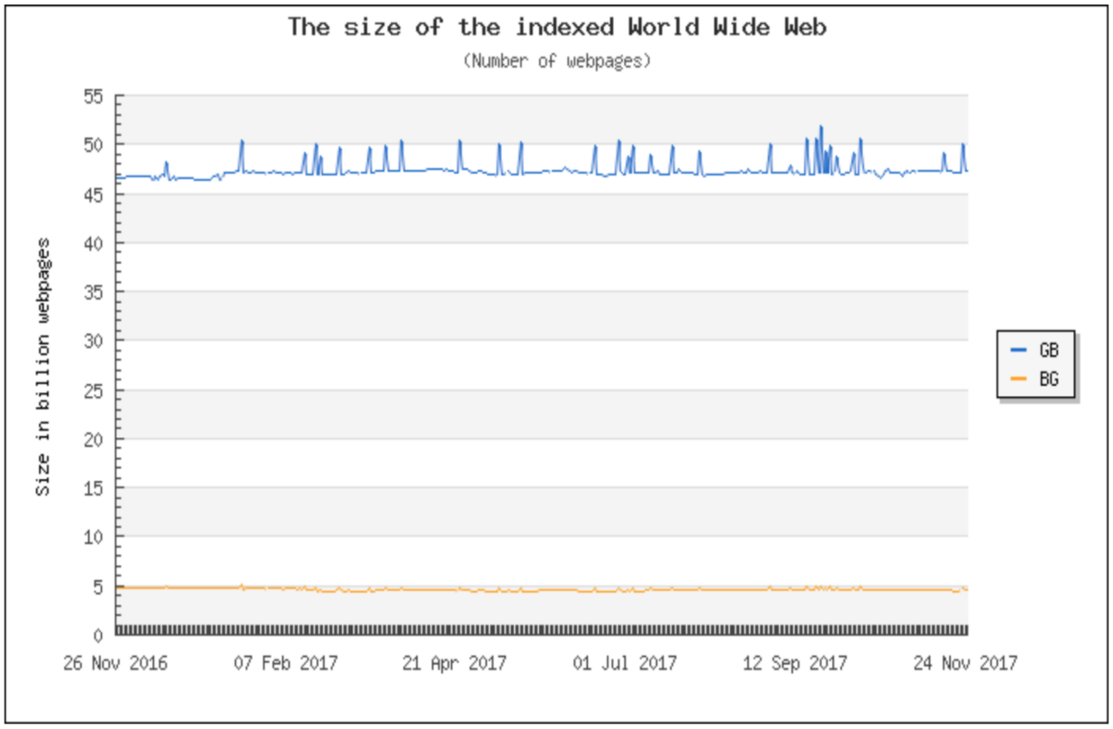
Link Analysis, PageRank Updating, Evolving Web Graph, Web Information Retrieval, Graph Mining

# Chapter 1 Introduction

## 1.1 Motivation

The fast growth of the conventional and mobile networks had led to information explosion. The real-world webs, such as the World-Wide-Web(WWW) or social networks like Twitter or Facebook, are in extremely large scale and also rapidly evolving. As the statistics showed, the estimated size of the indexed WWW is around 45-50 billion web pages [41] (Figure 1.1), and social network like Facebook has 2 billion active users [19], each with astonishing growing speed. Such massive web data foregrounded the importance of information retrieval techniques which help people to accurately sort and collect needed information from the chaos.

Web search engine plays centric role in web browsing and information retrieval nowadays. It serves as the knowledge base by answering queries with the information it retrieved from the World-Wide-Web(WWW), e.g. web pages, social network users, online documents, etc., through continuous probing and sorting the information over Internet. However, even after numerous pre-processing steps to filter out the unrelated information, the search results are usually still too vast for the users to skim through. Thus, an efficient and accurate ranking algorithm which puts the most relevant information at the top of the search results is at the core of the search engines.



GB = Sorted on Google and Bing

BG = Sorted on Bing and Google

**Figure 1.1** The estimated size of the WWW over year 2017

Link analysis based ranking algorithms are widely adopted by search engines such as Google for search result ranking purpose [16]. PageRank (PR)[33], a random-walk based web page ranking algorithm, is one of the most important algorithms to Google's initial success. The PageRank algorithm converts the web graph into a Markov transition matrix, and the PageRank score is the stationary distribution of such transition matrix, which essentially marks the probability of each node being visited in long-term as its importance. Personalized PageRank (PPR) [18, 21], which follows the similar principle as the PageRank but take personalization into consideration, is widely adopted by social networks to rank users on a personalized view [5, 17].

It is easy to see that the PageRank/Personalized PageRank computation is very expensive over such huge networks. More than a decade ago, the naïve Power-Iteration method [33] had elongated the time gap between updates due to the expensiveness of the update computing, which greatly influenced the freshness of the web search results. Hence, the efficient PageRank/Personalized PageRank updating algorithm adaptive to the massive and fast-evolving networks is on demand. On this direction, past works had made significant progresses, especially on updating Personalized PageRank [1, 6, 7, 28, 31, 32, 42]. However, the majority of those works only allow link structure modifications, and the updating for node insertion and deletion remains vague. Needless to say, the node evolvement, such as launching or closing down web sites, opening or terminating social accounts, happens every second in the web. Table

1.1 demonstrates how fast the nodes and links are evolving in the real-world networks. This motivated us to dig deeper and find solutions to the node insertion/deletion problem for efficient PageRank/Personalized PageRank updating.

**Table 1.1: Growth of the Real-World Networks**

| Network | Current Size | Growth Rate |
|---|---|---|
| WWW (Web Sites) | 12.9B | +400/min |
| Facebook users | 2.05B | +600/min |
| Google+ users | 562M | +180/min |

## 1.2 Preliminaries

Given a directed graph $G = (V, E)$ consists of a set of $n = |V|$ nodes and a set of $m = |E|$ edges. The nodes could represent the web pages in WWW or users in social networks, and the edges could be the hyper-links among web pages or relationships connecting the users. Each node $i$ associates with a set of out-neighbors $\mathcal{N}^{out}(i) = \{j: i \to j \in E\}$. Let $d_i = |\mathcal{N}^{out}(i)|$ be the out-degree of $i$. We define a Markovian transition matrix $P$ for $G$, in which the transition probability $p_{ij}$ is defined as the probability that a user at node $i$ to visit node $j$ through link $i \to j$. Precisely, $P$ is defined by:

$$P: \begin{cases} p_{ij} = 1/d_i, & \forall (i \to j) \in E \\ p_{ij} = 0, & otherwise \end{cases} \tag{1.1}$$

A node $i$ with $d_i = 0$ is called dangling node. There are several different ways to handle dangling nodes in $P$, we'll discuss that in detail at Chapter 2.

For the notations later in this thesis, we define $p(i)$ as the $i$-th row of matrix $P$, $\pi[i]$ as the $i$-th element of vector $\pi$, or similarly for other matrices and vectors.

## 1.3 The PageRank Algorithm

PageRank (PR) [33] came from an intuitive yet powerful idea that a web page is more important and ranked higher if it is more frequently visited by web surfers. PageRank utilizes Markov chain to model user's web surfing behavior [12], based on the directed web graph and by allowing a random jump. Today Google delivers a more intelligent web page ranking by including some other features beyond PageRank. However, scientists have verified that PageRank remains a pivoting feature among some possible ranking features [39]. An even more efficient PageRank updating algorithm remains demanded.

The PageRank vector $\pi$ is given by the following equation:

$$\pi = \alpha \pi P + (1 - \alpha)\mu \tag{1.2}$$

where $\mu = [1/n]_{1\times n}$, and the random-jump factor $\alpha = 0.85$ is widely adopted by prior approaches. Initialized by a stochastic vector $\pi^{(0)}$, PageRank is conventionally solved by the Power-Iteration method:

$$\pi^{(t+1)} = \alpha\pi^{(t)}P + (1-\alpha)\mu \tag{1.3}$$

where $\pi^{(0)}$ is selected as $\mu$ naïvely. The iterative process in Eq. (1.3) stops when the $l_1$ distance between $\pi^{(t)}$ and $\pi^{(t+1)}$ becomes smaller than some preset error bound $\delta$, that $\left\|\pi^{(t+1)} - \pi^{(t)}\right\|_1 < \delta$.

## 1.4 The Personalized PageRank

The Personalized PageRank (PPR) is the PageRank of personalized perspective, to provide customized ranking for users with different preferences on web pages, or different positions in social networks. The difference of PPR and PR is the preference vector. PPR is defined as:

$$\pi_u = \alpha\pi_u P + (1-\alpha)\rho \tag{1.4}$$

Unlike the uniform preference vector $\mu$ in PR, the preference vector $\rho$ in PPR is biased to users' interests, which the definition is application-dependent. The general definition of $\rho$ for web search can be found in [18, 21], in which $\rho = \nu$ is a regular stochastic vector; the ego-centric definition of $\rho$ for social ranking is mentioned in works [6, 28, 30, 42], in which $\rho = e_u$, where $e_u$ is an indicator vector with element $e[u] = 1$ and other elements equal to 0.

## 1.5 The Random-Walk Model

The Random-Walk model [35] is the fundamental of the PageRank algorithm. In PageRank, the web surfers are assumed to have two choices when browsing a web page: by probability $\alpha$, the user will follow the link on the current web page and “walk” to another web page by clicking that link; or by probability $1-\alpha$ the surfer types some URL in the browser and randomly “jump” to some other web pages without following any links. The PageRank is the stationary distribution of such stochastic process, which are the probabilities of each web page being visited by such random-jump model in long term.

The choice of $\alpha = 0.85$ is firstly proposed in [33], because Larry Page and Surgey Brin had tested and concluded that other $\alpha$ values significantly increased the number of iterations. Previous studies have tried different $\alpha$ values. Although it seems that the higher ranked nodes are not being affected too much by different $\alpha$, the rankings could be variated greatly for the lower ranked nodes, which may cause the variation to the searching results presented to users [36, 37, 40]. Thus, in this thesis we follow the main-stream selection of $\alpha = 0.85$.

## 1.6 Contributions

In this thesis, we proposed a novel concept named Virtual Web to tackle the general case of PageRank updating, in which both link and node insertion/deletion are allowed during the web evolvement. Our contributions are:

- The novel scheme we proposed is to address the dimensionality change problem caused by node insertions and deletions. It provides a very simple manner to form effective initializations by utilizing information from the previous PageRank.

- To our knowledge, this is the first PR updating algorithm that allows a large amount of nodes being inserted and deleted at the same time, while previous works only allow 1-2 nodes change at a time [42], or they assume fixed dimensionality without any node perturbation during updating [6, 7, 13, 28, 30, 32]. In our experiment, we allow 1%~5% node perturbation. This is already a dramatic change considering the size of the graphs. On the largest dataset we used, the Twitter-2010 dataset, that is equivalent to 0.4M~2M node insertion/deletion plus 14M link insertion/deletion at one time, which is a scale we have never seen in prior works.

- The virtual initializations generated from the proposed scheme can be easily integrated with other mainstream PR/PPR updating approaches, for the latter to overcome the node insertion/deletion problem. We prosed the Virtual Web Gauss-Southwell (VWGS) algorithm as a combination of the virtual initialization and the Gauss-Southwell method. The virtual initialization will not affect the big-O complexity of those approaches.

- The experiment result shows that on the largest dataset we tested, the Twitter-2010 dataset which contains 42 million nodes and 1.47 billion links, with a perturbation of 400,000 nodes insertion/deletion with more than 14 million links insertion/deletion together, the proposed VWGS method runs 3 times faster than the Gauss-Southwell algorithm starting from scratch in terms of run-time, or 20 times faster in terms of iteration number. The updating speed per each link insertion/deletion is 1800x faster than [32] under similar experiment environment.

- Further, we proposed an improved algorithm based on the VWGS method, to accelerate the updating for node insertion, the most common case in web evolvement. The optimized VWGS algorithm is 28% faster than the original VWGS method over the soc-LiveJournal dataset at maximum.

We first show the advantage of using the previous PageRank as initialization for calculating the current PageRank via either PI method or GS method, both theoretically and experimentally. We then introduce the virtual web, which is formed by adding the inserted nodes to the previous web together with some specific “in-home” link structure. Both in-links from the previous web and out-links to the previous web are excluded. The PR of the first virtual web is accurately calculated from the previous PR and the PR of the inserted nodes. We then re-organize the PR of the first virtual web, to form an initialization to compute the PR for the updated web with both link/node insertion/deletion, for faster computing speed. Experimental results show that PageRank updating based on the virtual initials significantly outperform a direct PageRank calculation based on the original model, either by PI method or by GS method.

In experiments, with node insertion/deletion totally up to 5% and link structure modification totally up to 1% of the original web graph, the proposed Virtual Web Gauss-Southwell approach delivered an acceleration of 51% ~ 72% over the Gauss-Southwell approach starting from zero-initial, or an acceleration of 65%~87.7% comparing to the Power-Iteration method initialized by uniform probability vector. The proposed optimized Virtual Web Gauss-Southwell algorithm achieved another 11.7%~28.3% gain over the soc-LiveJournal dataset with up to 20% node insertions. In comparison to the previous work in [32], the updating speed per each link insertion/deletion of VWGS is approximately 1800x faster under similar experiment environment.

This thesis is organized as follows. In Section 2 we discuss some mathematical fundamentals and related approaches. In Section 3, we show the advantage of using the previous PageRank as an initial in calculating the current PageRank via either PI method or GS method, both theoretically and experimentally. The web evolution, however, is restricted to only link insertion/deletion. That is, the node insertion/deletion is excluded. The general web evolution case is handled in Section 4. We introduce a virtual web, which is formed by adding the inserted nodes to the previous web together with some specific “in-home” link structure. Both in-links from the previous web and out-links to the previous web are excluded. With the aid of the virtual web, we are able to build a virtual initial PageRank and a virtual initial residual for updating the PageRank. We proposed a further improvement of the virtual web plus Gauss-Southwell method in Section 5, as an optimization to the node insertion problem. In Section 6, experiments are conducted to test the proposed algorithms. Experimental results show that PageRank updating based on virtual initials significantly outperform a direct PageRank calculation based on the original model, either by PI method or by GS method. Section 7 concludes the work.

# Chapter 2 Related Works

In this chapter, we give brief literature reviews to the previous works in the field of the PageRank(PR)/Personalized PageRank(PPR) computing. We will emphasis on those prior works focusing on updating PR/PPR in dynamic web environments, which enlightened our algorithm introduced in this thesis. We also analyzed the pros and cons of several up-to-date PR/PPR updating algorithms, and compared them to our approach.

## 2.1 Linear Algebra Methods

### 2.1.1 Power-Iteration Method

The Power-Iteration (PI) method is firstly proposed by Larry Page and Sergey Brin in the original PageRank paper[33] published in 1998, which becomes the core of the Google's web search engine later. A more formal notation of the PI method is proposed in later works [24]. Specifically, it is stated in Algorithm 1.

**Algorithm 1.** $PowerIteration(P, \pi, \delta)$

```
Inputs: transition matrix $P$, initialization $\pi^{(0)}$, error-bound $\delta$
1:  $e \leftarrow MAX\_INTEGER$
2:  **while** $e \geq \delta$ **do**
3:      $\pi^{(t+1)} = \alpha\pi^{(t)}P + (1-\alpha)\mu$
4:      $e \leftarrow \left\|\pi^{(t+1)} - \pi^{(t)}\right\|_1$
5:  **end while**
6:  **return** $\pi^{(t+1)}$
```

Here $e$ is the error, or the $l_1$-distance between to consecutive iterations in Algorithm 1. For both computational and storage efficiency, $P$ is further decomposed:

$$P = D^{-1}A \tag{2.1}$$

where $A$ is the adjacency matrix of the web. Because of the sparseness of $A$, it is often stored as an adjacency list and further compressed into the Compressed-Sparse-Column(CSC) format for storage feasibility. $D^{-1}$ is a diagonal matrix with $D_{ii}^{-1} = 1/o_i$ on the diagonal for each node $i$, or $D_{ii}^{-1} = 0$ if $o_i = 0$ for dangling nodes. This decomposition avoided the algorithm from storing $m$ floating-point numbers in $P$, so the storage feasibility is significantly improved. None the less, the decomposition also boosts the computing speed. It takes $m$ expensive floating-point multiplications if we compute $\pi P$ directly in each iteration, but in contrast, the product of $\pi D^{-1}$ can be computed through $\Omega(n)$ floating-point multiplications, and the computing of $\pi D^{-1}$ with $A$ (which consists of only 0s and 1s) is in a relatively lighter weight manner, which is faster.

The PI method is the fundamental method for PageRank updating. More than a decade ago, the naïve PageRank updating is through periodically applying PI method starting from $\pi^{(0)} = \mu$ to the most updated web graph. However, the $\Omega(m)$ complexity is its main drawback because $m$ can be enormous in real webs, thus an updating requires even only 1 iteration is still expensive. The initialization $\pi^{(0)}$ is the key for PI acceleration. The prior works that aiming to boost the computing speed of the PI method are through forming effective initializations to reduce the number of those expensive iterations. We'll show a theoretical proof of the effect of the initializations, and give a more accurate complexity analysis of the PI method in Chapter 3.

### 2.1.2 Block-based Decomposition Schemes

One direction of research on PR/PPR updating is divide-and-conquer. The idea is straight forward that if the PR/PPR computing can be broke down into smaller segments, then it can be accelerated through parallel/distributed computing. BlockRank [22] proposed by Kamvar et al. uses a host-induced block structure to decompose the web graph. Each host maintains and updates its local PR, so that the intra-host update computing can be distributed. The inter-host PR computing is over a much more contracted host-level graph, which is light-weight. The intra/inter-host PRs are then combined to form an initialization for the PI method applied to the entire web. However, due to the inaccuracy of the inter-host PageRank estimation, the gain of the proposed initialization is insignificant in the experiment we conducted. In [38], we proposed an algorithm based on Bayes rule to decompose the network into multiple mutually-disconnected components, so that the component-wise local PR can be computed in parallel. The local PRs can then be simply weighted by their sizes and integrated into the global PR, without the following-up Power-iteration computation like BlockRank. A mathematical proof is given in [3] by Avrachenkov et al.. However, because the real webs are often highly connected, we had hard times to identify the well-divided isolated components in the real web datasets, which is the main drawback of this method.

Although the performance of the block-based decomposition schemes seems unsatisfied, the advantages of the block structure, such as its parallelizable nature and easy implementation, have been inspired our work proposed in this thesis, which we will discuss in later chapters.

### 2.1.3 Aggregation/Disaggregation Method

Another way to accelerate PageRank updating is through reducing the size of the web graph for less works and faster computing speed. The idea of the Aggregation/Disaggregation methods comes from the intuition that the web structure modifications will only affect a small portion of local nodes that are surrounding the modification, and the nodes far away from the modifications should maintain invariant. Hence, the Aggregation/Disaggregation methods aggregate those invariant nodes into a single “super node”, and only updates PR values for the nodes that could be affected, so that the size of the entire web graph is significantly reduced.

From such principle, in [14], Chien et al. use the Aggregation/Disaggregation method to collect the end nodes of all inserted/deleted links into a set, while aggregating all other nodes into a single super node to form a contracted network, for which its PageRank is calculated. The PageRank values of the single nodes in the contracted network together with the original PageRank values for the nodes represented by the super node are simply combined and normalized as an approximation to PageRank update. In [34], Parreira et al. proposed a distributed PR updating scheme over the P2P network, in which each peer associates a sub-graph. It estimates its own version of the global PR based on its subgraph and a super node, which is an aggregation of all other nodes outside the peer. Each peer doesn’t know the node connectivity within the super node. As a result, the storage for each peer becomes feasible, good for distributed computing. A “meeting” procedure is introduced that allows peers communicating pair-wisely to expand their knowledges of the global web for improved estimation accuracy but at a high cost. The scheme assumes a static global web, even though each peer may add in more nodes hence more links during the meeting with another peer to improve the accuracy of its own version of the global PR.

The accuracy is the major concern of the Aggregation/Disaggregation method. As commented and tested by Ohsaka et al. in [32], and in works [25], the choice of the subset of nodes not being aggregated would greatly affect the accuracy, and the works as guidance for selecting such subsets provide no theoretical guarantee.

## 2.2 Monte-Carlo Method

Monte-Carlo method, a random-walk simulation method, fundamentally differs from Linear Algebra method. An iteration requires running random-walk at each of $n$ nodes. Let $k$ be the number of iterations. The PageRank score at a node is approximated by the frequency of the node being visited during the $k$ iterations of random-walk simulation. The method is easy to implement and fully parallelizable. However, the convergence could be quite slow. As mentioned by Avrachenkov et al. in [4], if the accuracy is unsatisfied after the first iteration, it is hard to improve through more iterations since the error can be reduced only by an extremely small factor $1/\sqrt{kn}$ on average. In [6], Bahmani et al. use Monte-Carlo method to track PPR on an evolving network. The approach maintains multiple random-walk segments, from which the PPR scores can be quickly approximated. As the link structure modifies, the segments are reconstructed. The method requires massive pre-processing and storage for random-walk segments. Page insertion/deletion issue is not addressed. In [7], a probing strategy as how to choose some appropriate starting points for random-walk simulation is studied, where the graph is assumed slowly evolving with only link structure modification. In [28], Lofgren et al. use a pair-wise approach to estimate PPR. A pair-wise PPR $\pi_s(t)$ is defined as the probability from source-node $s$ to target-node $t$ via random-walk. Node $\omega$ is considered "close" to node $t$ if $\pi_\omega(t)$ is greater than some threshold. $\pi_s(t)$ can be estimated bi-directionally. First, trace backward from the target node through links to find a "frontier set", which consists of nodes each "close" to the target node. Then, $\pi_s(t)$ is approximated by $\pi_s(\omega) \cdot \pi_\omega(t)$ summed over nodes $\omega$ but only in the frontier set, where $\pi_\omega(t)$ is estimated by Gauss-Southwell method and $\pi_s(\omega)$ by multiple Monte-Carlo random-walk simulations initiated from the source node. This method represents a novel approach for fast PPR computation in social networks, even though the demand for pre-processing and storage remains high. In another of their work based on assumed undirected (hence symmetric) graph [30], Gauss-Southwell method is used in the forward work ($\pi_s(\omega)$) while Monte-Carlo method in the backward work ($\pi_\omega(t)$). As a result, the frontier set is omitted and the pre-computation and storage are saved. The accuracy remains an issue and the node insertion/deletion issue is not addressed.

## 2.3 Local-Push Method

Local-push Methods [2] (also named as Gauss-Southwell method or Bookmark Coloring Algorithm [8], we follow the name of Gauss-Southwell method [32] in our work) is often used for PageRank updating acceleration, particularly in Personalized PageRank (PPR) updating[30, 32, 42]. In [32], Ohsaka et al. use the previous snapshot of the web to efficiently form a residual vector to initialize Gauss-Southwell algorithm for real-time PPR tracking application. The complexity of the algorithm is $O(\bar{d}/\epsilon)$ for updating a single link insertion or deletion, or $O(k + \bar{d} \log k/\epsilon)$ for $k$ updates of sequential edge insertions or deletions, where $\bar{d}$ is the average out-degree of the web graph. Zhang et al. proposed the *LazyForwardUpdate* algorithm in [42] to address the similar problem for directed graphs, that they tried to modify the estimation vector and the residual vector according to the edge insertion/deletion, in order to reduce the iteration numbers of the local forward-push algorithm. The complexity for $k$ updates of sequential edge insertions or deletions is improved to $O(k + 1/\epsilon)$. The experiment result shows that it is faster than Ohsaka et al.'s work when both methods achieve similar accuracy. This method allows node insertion and deletion, but and the links of the nodes must be inserted or removed one-by-one, which could be slow in practice.

### 2.3.1 The Gauss-Southwell Method

The Gauss-Southwell (GS) method mentioned in [32] received more attentions for PPR computing in recent years. The GS method was originally adopted for PR computing in [8], by replacing the personalization vector $\nu$ to the uniform probability vector $\mu$. In this section, we'll mainly discuss GS method under the context of PR computing. It is an alternative to the PI method, which transformed the convergence from checking the $l_1$-distance of the PR vectors to checking the maximum-norm of a residual vector $r$. Experiment results in previous works showed that it has a comparable accuracy but faster computing speed in comparison to the PI method. In Chapter 3, we analyzed and compared it with the PI method in details.

The residual vector $r$ is defined as:

$$r^{(t)} = \pi(I - \alpha P) - x^{(t)}(I - \alpha P) \tag{2.2}$$

where $\pi$ is the true PR, and $x^{(t)}$ is the estimated PR vector given by GS method at the $t$-th iteration. Notice that the residual vector $r$ is essentially the difference between the estimation $x$ and the true PR vector $\pi$. When $r^{(t)} \to 0$, we have

$$\pi(I - \alpha P) = x^{(t)}(I - \alpha P) \tag{2.3}$$

Eq. (2.3) is satisfied when $x^{(t)} = \pi$. Hence, the goal of the GS iterations is to soothe the residual vector $r$ and make it tend to zero. Since Eq. (1.2) can be written as:

$$\pi(I - \alpha P) = (1 - \alpha)\mu \tag{2.4}$$

so the residual vector turns to:

$$r^{(t)} = (1 - \alpha)\mu - x^{(t)}(I - \alpha P) \tag{2.5}$$

Therefore, we can calculate a corresponding $r^{(t)}$ for any given estimation $x^{(t)}$. In the basic GS method, $x$ is initialized by the zero-vector, i.e. $x^{(0)} = \mathbf{0}$, so that $r^{(0)} = (1 - \alpha)\mu$ which avoided calculating the product of $x^{(t)}(I - \alpha P)$. We call this basic GS method as GS method from scratch later in this thesis.

During each iteration $t$, the algorithm search for the residual elements in $r^{(t)}$ with norm larger than a given error bound $\epsilon$. If such residual is found, say the $r^{(t)}[i] > \epsilon$, a "push" operation will be applied to it. Firstly, the $\alpha$ portion of the residual will be re-distributed to node $i$'s out-neighbors, which forms $r^{(t+1)}$; the residual will also be added to the $i$-th element in the PR estimation vector, which forms $x^{(t+1)}$.

As stated in line 4 of Algorithm 2, the norm of $r$ is decreased by $\left|r^{(t-1)}[i]\right| > \epsilon$ after each iteration, until when the norms of all residuals in $r$

become smaller than $\epsilon$ so that $|r| < n\epsilon$, then $x$ is considered as converged to the true PageRank $\pi$. Generally, it's the norm of the initial residual $r^{(0)}$ to decide the convergence speed of the GS method, which we will show in Section 3.2.

The detailed algorithm of the GS method is as below:

**Algorithm 2**. *GaussSouthwell* $(P, x^{(0)}, r^{(0)}, \epsilon)$

```
Inputs: transition matrix $P$, error-bound $\epsilon$
Initialization: $x^{(0)} = 0$, $r^{(0)} = (1-\alpha)\mu$
1: $t \leftarrow 0$
2: while $\exists i$ so that $\left|r^{(t)}[i]\right| > \epsilon$
3:         $x^{(t+1)} = x^{(t)} + r^{(t)}[i]e_i$
4:         $r^{(t+1)} = r^{(t)} - r^{(t)}[i]e_i + r^{(t)}[i]e_i\alpha P$
5:         t++;
6: end while
7: return $\langle \pi^{(t)}, r^{(t)} \rangle$
```

In the implementation, we use a queue for line 2 in Algorithm 2 as stated in [32]. Any node $i$ becomes that $\left|r^{(t-1)}[i]\right| > \epsilon$ when computing line 4 will be enqueued right away for later iterations, thus the complexity of searching such satisfying node $i$ becomes $O(1)$. Algorithm 2 stops when the queue is emptied.

To adapt to the dynamic web environment, Ohsaka et al. proposed a simple solution in [32] for updating PPR which accelerates the computation for web evolvement with only link perturbations, based on previous estimation and the difference between the previous and the updated networks. We converted their approach to adapt PR computing. Assuming a web $W$ is evolved to $W^*$ with only link modifications. From Eq. (2.5) we have:

$$r = (1-\alpha)\mu - x(I - \alpha P)$$

$$r^* = (1-\alpha)\mu - x^*(I - \alpha P^*)$$

where $r$, $x$ and $r^*$, $x^*$ are the converged residual and PR vector of the previous and the updated web, respectively. Combining the two equations above, we get

$$\begin{aligned} r^* &= r + x(I - \alpha P) - x^*(I - \alpha P^*) \\ &= r - (x^* - x) + \alpha(x^* P^* - xP) \end{aligned} \tag{2.6}$$

If we use the previous PR vector as initialization for calculating the updated PR, that $x^{*(0)} = x$, we have

$$r^{*(0)} = r + \alpha x(P^* - P) \tag{2.7}$$

Eq. (2.7) is the key idea of [32] for PR updating under link structure modification. Assuming that only a very small portion of nodes $\mathcal{N} \in V$ are related with the link perturbation, that their lost some existing out-links or they acquired some new out-links, then the majority of the entries in matrix $P^* - P$

will be zero, except only a few non-zero rows corresponding to $\mathcal{N}$. This character boosts two advantages: first, the computing for the product $\alpha x(P^* - P)$ will be fast due to the many zeros in $(P^* - P)$; second, since there are many zeros elements in the vector given by $\alpha x(P^* - P)$, the norm of the new initial residual vector $r^{*(0)}$ will not be dramatically increased from $r$ (which has been converged to almost zero), so the iteration number of the updating computation is expected to be much lower than starting from scratch (i.e., from $x^{*(0)} = 0$ and $r^{*(0)} = (1 - \alpha)\mu$). Therefore, it can be summarized that the Gauss-Southwell algorithm is useful on updating PR for link perturbations in small scale.

## 2.4 The Dangling Node Effect

### 2.4.1 The Dangling Nodes

A dangling node is defined as a node without any outlinks. In the original PageRank paper [33], Larry Page and Sergey Brin had seen the dangling nodes as "rank sinks" for not being able to distribute the importance scores to other non-dangling nodes, so they suggested to remove those dangling nodes before PR computation. The dangling nodes are handled in several different ways, including inserting self-links to the dangling nodes [23]; or replacing the rows corresponding to dangling nodes with the personal preference vector [10]. We adopt the solution in [24] because it is more suitable for general PR computing in Eq. (1.2). More details of these dangling nodes handling can be found in the survey paper [15].

### 2.4.2 Dangling Node Handling

A typical solution to handle the dangling nodes is to replace the rows corresponding to the dangling nodes with $\mu$ [24], which can be intuitively interpreted as a random-jump from the dangling node since the user has no link to follow. In that case, $P$ turns to

$$P_u = D^{-1}A + d^T\mu \tag{2.8}$$

where $d$ is the vector that $d_i = 1$ if $i$ is dangling node, and $d_i = 0$ if not. Step 3 in Algorithm 1 is then written as:

$$\pi^{(t+1)} = \alpha\pi^{(t)}D^{-1}A + \left(1 - \alpha + \alpha\pi^{(t)}d^T\right)\mu$$

Langville et al. provided a way to handle the dangling nodes under such case. We put it here for completeness. Assume there are $k < n$ dangling nodes among $n$ nodes, which, without loss of generality, are arranged at the bottom of the transition matrix. Thus, $P$ could take two different forms:

$$P_u = \begin{bmatrix} P_{(n-k)\times n} \\ \frac{1}{n}\, 1_{k\times n} \end{bmatrix}, \qquad P_z = \begin{bmatrix} P_{(n-k)\times n} \\ 0_{k\times n} \end{bmatrix}$$

Let $\pi_u$ and $\pi_z$ be their respective PageRank. By Eq. (2.4) it's easy to verify

$$\pi_u - \pi_z = \alpha(\pi_u - \pi_z)P_u + \alpha\pi_z(P_u - P_z)$$

$$(\pi_u - \pi_z)(I - \alpha P_u) = \alpha \pi_z \begin{bmatrix} 0_{(n-k)\times n} \\ \frac{1}{n}\ 1_{k\times n} \end{bmatrix} = \alpha \|\pi_z{}''\|_1 \frac{1}{n}\ 1_{1\times n}$$

where $\pi_z = (\pi_z', \pi_z'')$, $\pi_z''$ is a row vector of $k$ dimensional, $\|\pi_z''\|_1$ is its $l_1$ norm. From the last equation and $\mu = \frac{1}{n}\ 1_{1\times n}$, we obtain

$$\pi_u - \pi_z = \alpha \|\pi_z''\|_1 \mu (I - \alpha P_u)^{-1}$$

Since $\pi_u = (1-\alpha)\mu(I - \alpha P_u)^{-1}$, we arrive at

$$\pi_u - \pi_z = \frac{\alpha}{(1-\alpha)} \|\pi_z''\|_1 \pi_u$$

$$\pi_z = (1 - \frac{\alpha}{(1-\alpha)} \|\pi_z''\|_1)\pi_u$$

Let $P_z$ be written as $P_z = \begin{bmatrix} P' & P'' \\ 0 & 0 \end{bmatrix}$, where $P' \in \mathbb{R}_{(n-k)\times(n-k)}$ is corresponding to the non-dangling nodes and $P'' \in \mathbb{R}_{(n-k)\times k}$ is corresponding to the dangling nodes. Then,

$$\pi_z' = \alpha \pi_z' P' + (1-\alpha)\mu' \qquad (2.9)$$

$$\pi_z'' = \alpha \pi_z' P'' + (1-\alpha)\mu'' \qquad (2.10)$$

By Eq. (2.9) & (2.10), it's clear that in order to calculate $\pi_z$ we need only to calculate $\pi_z'$ since $\pi_z''$ can be obtained via Eq. (2.10) with no need for iteration. From Eq. (2.4) we have

$$\pi = (1-\alpha)\mu(I-\alpha P)^{-1} = \lim_{t\to\infty}(1-\alpha)\mu \sum_{0\le k<t}(\alpha P)^k$$

Similarly,

$$\pi_z' = \lim_{t}(1-\alpha)\mu' \sum_{0\le k<t}(\alpha P')^k \tag{2.11}$$

We have $\pi_u > 0$, $\pi_z > 0$ because of Eq. (2.11), $\pi_z > (1-\alpha)\mu' > 0$. From the last equation and $\|\pi_u\|_1 = 1$, we obtain

$$\pi_u = \frac{\pi_z}{\|\pi_z\|_1}$$

It is clear that in order to calculate PageRank $\pi_u$ we need only to calculate PageRank $\pi_z$ on the scarcer transition matrix $P'$ without any dangling nodes. Since the dangling nodes can be handled in this way, we do not assume any dangling node fixes in our work. Other works including [20, 26] which estimate the PageRank for dangling nodes base on the Aggregation/Disaggregation concept which aggregate all dangling into one node and compute PageRank over the contracted network.

# Chapter 3 Theoretical Analysis

In this chapter, we analyzed the methods related to our work. We showed the effectiveness of using previous PageRank as initializations to accelerate the computation of the updated PageRank, theoretically and experimentally. We also compared the two state-of-art PageRank computing methods, the Power-iteration method and the Gauss-Southwell method.

## 3.1 The Effect of Initialization for Power-Iteration

Initialization is one of the key factors to accelerate PageRank computing. A vast majority of the previous works on PR updating, e.g. [6, 7, 28-30, 32], are focusing on forming effective initializations to reduce the iteration numbers of PageRank computing. We give a theoretical analysis in this section to show the effect of the initialization to PageRank.

For $t > 0$, the iterative process in Algorithm 1 can be decomposed that

$$\pi^{(t)} = \alpha\pi^{(t-1)}P + (1-\alpha)\mu$$

$$= \pi^{(t-2)}(\alpha P)^2 + (1-\alpha)\mu(\alpha P) + (1-\alpha)\mu$$

$$= \cdots = \pi^{(0)}(\alpha P)^t + (1-\alpha)\mu \sum_{0\le k<t} (\alpha P)^k$$

$$= \pi^{(0)}(\alpha P)^t + (1-\alpha)\mu \sum_{0\le k<t} (\alpha P)^k \tag{3.1}$$

The first term in Eq. (3.1) represents a transient process, while the second one a steady-state process. As known, the $l_1$ norm of a vector is defined by the sum of the absolute values of its components, while the $l_1$ norm of a matrix by the maximum of the $l_1$ norms of its row vectors. Moreover, the $l_1$ norm of a vector multiplied by a matrix from right is no larger than the product of the $l_1$ norm of the vector and the $l_1$ norm of the matrix. It can be easily verified that $\left\|\pi^{(0)}(\alpha P)^t\right\|_1 = \alpha^t$, which approaches to zero at limit. Thus, PageRank $\pi$ is given by

$$\pi = lim(1-\alpha)\mu \sum_{0\le k<t} (\alpha P)^k$$

$$= (1-\alpha)\mu \sum_{0\le k} (\alpha P)^k \tag{3.2}$$

$$= (1-\alpha)\mu(I - \alpha P)^{-1} \tag{3.3}$$

which implies Eq. (2.4). Even though Eq. (3.2) can also be used to iteratively calculate $\pi$, it may not be an efficient way.

**Lemma 1.** $\left\|\pi - \pi^{(t)}\right\|_1 \leq \left\|\pi - \pi^{(0)}\right\|_1 \alpha^t$

***Proof.*** By Eq. (3.1) and (3.2), it follows

$$\begin{aligned}\pi - \pi^{(t)} &= -\pi^{(0)}(\alpha P)^t + (1-\alpha)\mu \sum_{k \geq t} (\alpha P)^k \\ &= -\pi^{(0)}(\alpha P)^t + \left\{(1-\alpha)\mu \sum_{k \geq 0} (\alpha P)^k\right\}(\alpha P)^t \\ &= (\pi - \pi^{(0)})(\alpha P)^t \qquad (3.4)\end{aligned}$$

From the above, Lemma 1 follows.

Lemma 1 shows that given a transition matrix $P$, it is the closeness of the initial $\pi^{(0)}$ to $\pi$ to decide convergence rate of $\pi^{(t)}$ to $\pi$. In the original PageRank algorithm, $\pi^{(0)} = \mu$ is chosen by default.

Let $P$ and $P^*$denote the transition matrix for the previous web graph $G$ or the current web $G^*$, respectively. From the previous web to the current one, only link structure modification is assumed. It is obvious that $P$ and $P^*$ share the same dimensionality. Let $\pi$ and $\pi^*$ be their respective PageRank. If the difference between $P$ and $P^*$ is small, intuitively $\pi$ should be close to $\pi^*$ and could be effectively used to initialize the computation of $\pi^*$. The following holds:

If the previous PageRank $\hat{\pi}$ is used as $\pi^{(0)}$ in calculating $\pi$, we have

$$\pi^* - \pi = \alpha(\pi^* P^* - \pi P)$$

$$= \alpha(\pi^* - \pi)P + \alpha\pi(P^* - P)$$

that is

$$(\pi^* - \pi)(I - \alpha P^*) = \alpha\pi(P^* - P)$$

$$\pi^* - \pi = \alpha\pi(P^* - P)(I - \alpha P^*)^{-1}$$

$$\|\pi^* - \pi\|_1 \leq \alpha\|\pi\|_1 \|P^* - P\|_1 \|\textstyle\sum_{k\geq 0}(\alpha P^*)^k\|_1$$

$$\leq \frac{\alpha}{1-\alpha}\|P^* - P\|_1 \tag{3.5}$$

which implies

$$\left\|\pi^* - \pi^{*(t)}\right\|_1 = \|(\pi^* - \pi)(\alpha P^*)^t\|_1 \leq \frac{\alpha^{t+1}}{1-\alpha}\|P^* - P\|_1 \tag{3.6}$$

**Lemma 2.** The closeness between the previous PageRank $\pi$ and the current PageRank $\pi^*$ is determined by $\|P^* - P\|_1$. When the previous PageRank $\pi$ is used to initialize the computation of the current PageRank $\pi^*$, the convergence rate of $\pi^{*(t)}$ to $\pi^*$ is also determined by $\|P^* - P\|_1$.

From Lemma 2 and Eq. (3.5, 3.6), one can make a straight forward conclusion that if the difference between $P^*$ and $P$ is small, say $P^*$ is formed from $P$ with a small portion of links added or removed, then using the previous PageRank to initialize the PageRank computation for the updated graph, i.e. let $\pi^{*(0)} = \pi$, may reduce the $l_1$ distance between the initialization and the final PageRank, in order to reduce the number of iterations. To further verify this conclusion, we conducted preliminary experiments in section 3.1.2 & 3.1.3.

### 3.1.1 Preliminary Experiment Setup

We conduct preliminary experiments to verify the effect of the initialization on PageRank computation. The datasets are selected from Table 6.1. For each given dataset, we create a uniformly random edge-stream by re-sorting the links to random order, and remove the last 5% of the links ($0.05m$) to create the original graph $G_0 = (V, E_0)$, which is represented by the transition matrix $P_0$. Next, the removed links are inserted back by every 1%, which forms 5 updated graphs $G_1 = (V, E_1)$ to $G_5 = (V, E_5)$ with progressive link insertion proportion, each associated with a transition matrix from $P_1$ to $P_5$. We use the uniform probability vector $\mu$ as initialization to compute the original PageRank $\pi_0$ over

$G_0$. Then, for each dataset $G_k$, we calculate its PageRank $\pi_k$ with $\pi_0$ as initialization, and compare it with the naïve method which use $\mu$ as initialization. The error bound for Power-iteration method is set to $\delta = 10^{-3}$, and the error bound for Gauss-Southwell method is set to $\epsilon = 10^{-9}$.

### 3.1.2 Effect of the Initialization for Power-iteration Method

We first test the effect of using previous PageRank $\pi_0$ to initialize the computation of the updated PageRank $\pi_1 \sim \pi_5$ through Power-Iteration method. The results are list in Table 3.1, which clearly shows that using previous PageRank as initialization on updated dataset with only link perturbations achieved significant gain. On the datasets we tested, the maximum acceleration in terms of iteration numbers achieved are from 73.5% to 78.9% (for 1% link insertion), and the minimum gain are from 57.7% to 68.7% (for 5% link insertion).

We noticed that the percentage of gain achieved by using $\pi_0$ as initialization is dataset independent. The gain achieved on smaller datasets with only 5 million edges or very large dataset with more than 1.4 billion edges are similar. We did not put the run-time result here because for each dataset the run-time per iteration is almost the same in PI method, thus the acceleration rate in terms of run-time or iteration number are similar.

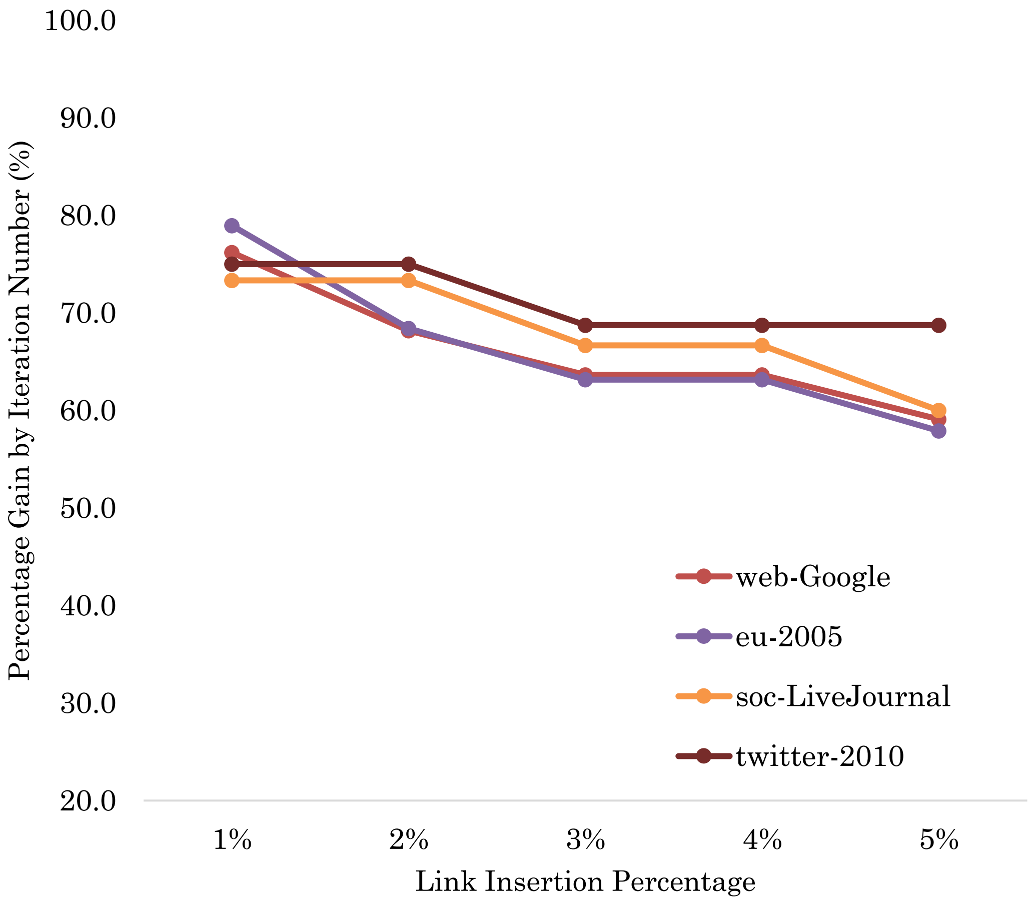


**Figure 3.1** The acceleration rate (measured by iteration number) by using pervious PR as initialization for PI method, when 1% ~ 5% links are inserted.

**Table 3.1 PI with $\pi_0$ or $\boldsymbol{\mu}$ as initialization**

| Dataset | Method | Initial | Link Insertion Percentage (Run-time: ms) | | | | |
|---|---|---|---|---|---|---|---|
| | | | 1% | 2% | 3% | 4% | 5% |
| web-Google | Iteration Number | $\mu$ | 21 | 22 | 22 | 22 | 22 |
| $\|V\|$ = 0.91M | | $\pi_0$ | 5 | 7 | 8 | 8 | 9 |
| $\|E\|$ = 5M | Runtime | $\mu$ | 2027 | 2300 | 2141 | 2305 | 2254 |
| | | $\pi_0$ | 505 | 715 | 749 | 820 | 922 |
| eu-2005 | Iteration Number | $\mu$ | 19 | 19 | 19 | 19 | 19 |
| $\|V\|$ = 0.86M | | $\pi_0$ | 4 | 6 | 7 | 7 | 8 |
| $\|E\|$ =19.2M | Runtime | $\mu$ | 5167 | 5205 | 5209 | 5288 | 5299 |
| | | $\pi_0$ | 1106 | 1641 | 1931 | 1964 | 2242 |
| soc-LiveJournal | Iteration Number | $\mu$ | 15 | 15 | 15 | 15 | 15 |
| $\|V\|$ = 4.85M | | $\pi_0$ | 4 | 4 | 5 | 5 | 6 |
| $\|E\|$ = 69M | Runtime | $\mu$ | 25839 | 26286 | 26429 | 26569 | 26685 |
| | | $\pi_0$ | 6840 | 6893 | 8692 | 8779 | 10586 |
| Twitter-2010 | Iteration Number | $\mu$ | 16 | 16 | 16 | 16 | 16 |
| $\|V\|$ = 41.7M | | $\pi_0$ | 4 | 4 | 5 | 5 | 5 |
| $\|E\|$ = 1.47B | Runtime | $\mu$ | 762839 | 769386 | 772553 | 783303 | 790829 |
| | | $\pi_0$ | 190768 | 192870 | 242700 | 245240 | 247573 |

## 3.2 Effect of the Initialization on Gauss-Southwell Method

Here we analyze the effect of using previous results as initialization for the GS method on updated web graph. Prior work of utilizing pervious PR as initial as been proposed as Eq. (2.7), but no theoretical analysis was provided to prove that the effectiveness of the initialization for GS method, so we show that here. Firstly, we have the following lemma:

**Lemma 3.** $r^{(t)} = \left(\pi - x^{(t)}\right)(I - \alpha P)$ holds for GS method (before its convergence).

**Proof.** By GS method, we have

$$\begin{aligned} r^{(0)} &= (1-\alpha)\mu \ - \ x^{(0)}(I - \alpha P) \\ &= \pi(I - \alpha P) - x^{(0)}(I - \alpha P) \\ &= (\pi - x^{(0)})(I - \alpha P) \end{aligned}$$

$$\begin{aligned} r^{(t+1)} &= r^{(t)} - r^{(t)}[i] e_i (I - \alpha P) \\ &= r^{(t)} - \left(x^{(t+1)} - x^{(t)}\right)(I - \alpha P) \end{aligned}$$

Suppose at $t$,

$$r^{(t)} = \left(\pi - x^{(t)}\right)(I - \alpha P)$$

for $t > 0$, then

$$r^{(t)} = \left(\pi - x^{(t)}\right)(I - \alpha P) - \left(x^{(t+1)} - x^{(t)}\right)(I - \alpha P)$$

$$= \left(\pi - x^{(t)}\right)(I - \alpha P)$$

By introduction, Lemma 3 is proved.

**Lemma 4.** $\left\|\pi - x^{(t)}\right\|_1 \leq \left\|r^{(t)}\right\|_1 \frac{1}{(1-\alpha)}$ holds for GS method (before its convergence).

**Proof.** By Lemma 3, $\pi - x^{(t)} = r^{(t)}(I - \alpha P)^{-1}$.

Thus,

$$\left\|\pi - x^{(t)}\right\|_1 \leq \left\|r^{(t)}\right\|_1 \|(I - \alpha P)^{-1}\|_1 \leq \left\|r^{(t)}\right\|_1 \frac{1}{(1-\alpha)}$$

**Corollary 1**. $\left\|\pi - \pi^{(t)}\right\|_1 \leq \delta$ if $\left\|r^{(t)}\right\|_1 \leq (1-\alpha)\delta$.

As shown in [32], suppose GS method converges at the $\tau$*-th* iteration, i.e., $\forall i$, $|r^{(\tau)}[i]| \leq \epsilon$ for the first time. Then, $\|r^{(\tau)}\|_1$ is reduced from $\|r^{(0)}\|_1$ by at least $(1-\alpha)\epsilon\tau$. That is,

$$\|r^{(\tau)}\|_1 \leq \|r^{(0)}\|_1 - (1-\alpha)\sum_{t=0}^{\tau-1} r^{(t)}[i_t] \leq \|r^{(0)}\|_1 - (1-\alpha)\epsilon\tau$$

we obtain

**Lemma 5.** $\tau \leq \frac{\|r^{(0)}\|_1}{(1-\alpha)\epsilon}$ and $\tau = O\left(\frac{\|r^{(0)}\|_1}{\epsilon}\right)$

As can be imagined, a good initial $x^{(0)}$, which produces a smaller residual $\|r^{(0)}\|_1$, would effectively reduce the iteration number.

We conduct experiments to test the effect of the initialization following the experiment setup in Section 3.1.1, unlike [32], in initializing $r_k^{(0)}$ we do not use $x_k^{(0)} = 0$ for each updated graph $G_k$. Let $x_k^{(0)} = \pi_0$, to which the initial residual for the updated graph $r_k'^{(0)} = (1-\alpha)\mu - \pi_0(I - \alpha P_k)$ is associated. Like Eq. (2.7) in Section 2, we have

$$r'^{(0)} = r^{(\tau)} + \alpha\pi_0(P_k - P_0), k = 1, \ldots, 5$$

The detailed results are listed in Figure 3.2, 3.3 and Table 3.2.

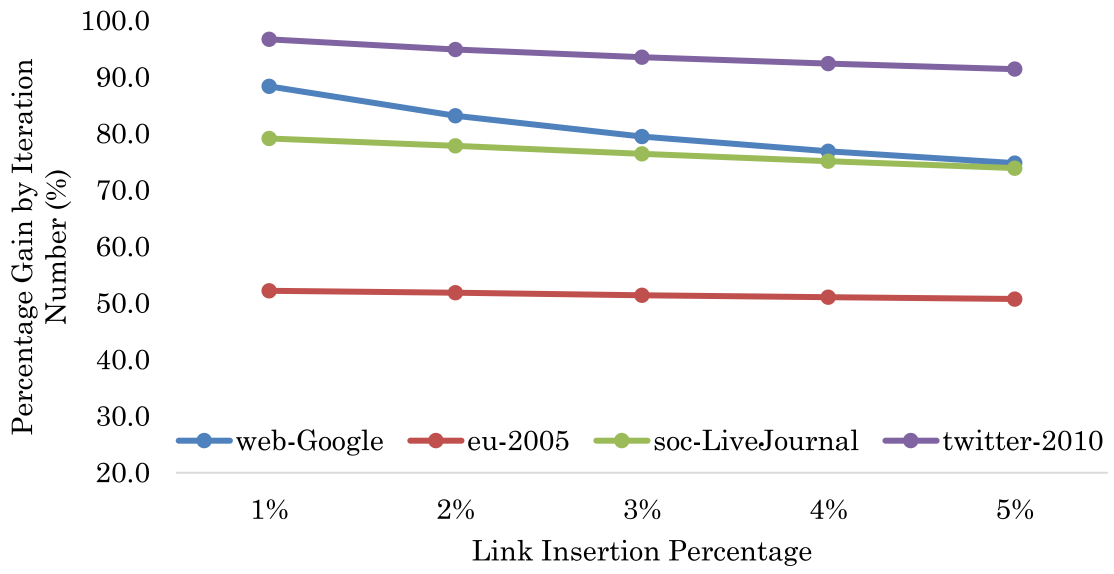


**Figure 3.2** The acceleration rate by iteration number of using pervious PR as initialization for GS method, when 1% ~ 5% links are inserted.

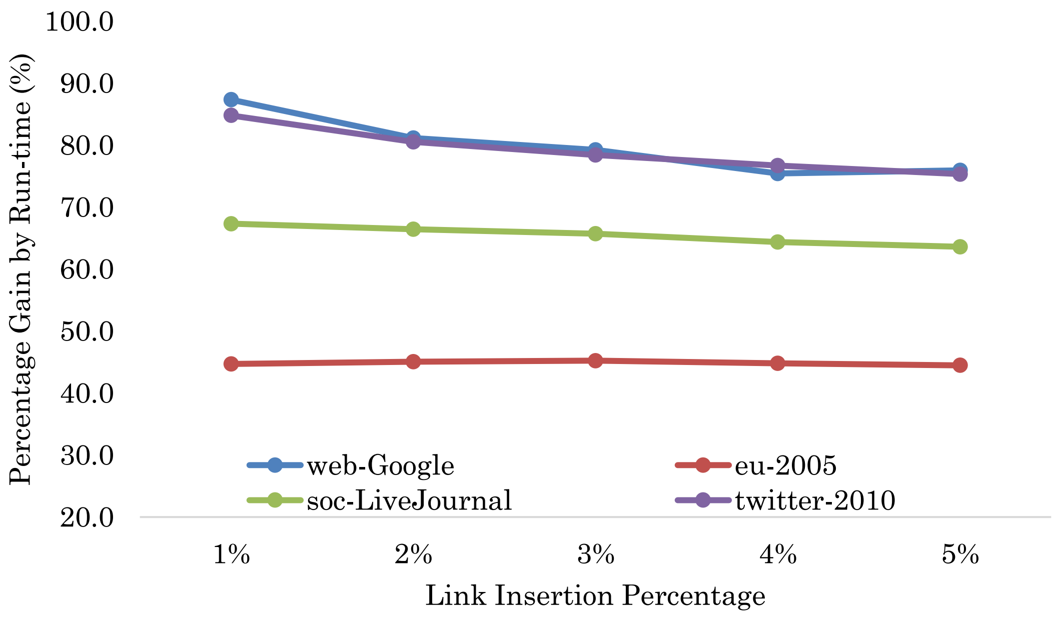


**Figure 4.3** The acceleration rate by run-time of using pervious PR as initialization for GS method, when 1% ~ 5% links are inserted.

**Table 3.2 GS with $\langle \pi_0, r_0 \rangle$ or $\langle 0, (1-\alpha)\mu \rangle$ as initialization**

| Dataset | Method | Initial | Link Insertion Percentage (Run-time: ms) | | | | |
|---|---|---|---|---|---|---|---|
| | | | 1% | 2% | 3% | 4% | 5% |
| web-Google | Iteration Number | $\langle 0, (1-\alpha)\mu \rangle$ | 7.89M | 7.92M | 7.96M | 8.00M | 8.03M |
| $\lvert V \rvert$ = 0.91M | | $\langle \pi_0, r_0 \rangle$ | 0.92M | 1.33M | 1.63M | 1.85M | 2.02M |
| $\lvert E \rvert$ = 5M | Runtime | $\langle 0, (1-\alpha)\mu \rangle$ | 1827 | 1919 | 1888 | 1951 | 1910 |
| | | $\langle \pi_0, r_0 \rangle$ | 231 | 362 | 392 | 479 | 459 |
| eu-2005 | Iteration Number | $\langle 0, (1-\alpha)\mu \rangle$ | 5.20M | 5.22M | 5.24M | 5.25M | 5.26M |
| $\lvert V \rvert$ = 0.86M | | $\langle \pi_0, r_0 \rangle$ | 2.49M | 2.51M | 2.54M | 2.57M | 2.59M |
| $\lvert E \rvert$ =19.2M | Runtime | $\langle 0, (1-\alpha)\mu \rangle$ | 1918 | 1944 | 1948 | 1974 | 1971 |
| | | $\langle \pi_0, r_0 \rangle$ | 1060 | 1068 | 1067 | 1090 | 1095 |
| soc-LiveJournal | Iteration Number | $\langle 0, (1-\alpha)\mu \rangle$ | 39.53M | 39.60M | 39.67M | 39.74M | 39.81M |
| $\lvert V \rvert$ = 4.85M | | $\langle \pi_0, r_0 \rangle$ | 8.24M | 8.77M | 9.35M | 9.88M | 10.38M |
| $\lvert E \rvert$ = 69M | Runtime | $\langle 0, (1-\alpha)\mu \rangle$ | 26289 | 26635 | 26799 | 26886 | 27222 |
| | | $\langle \pi_0, r_0 \rangle$ | 8588 | 8942 | 9191 | 9586 | 9907 |
| Twitter-2010 | Iteration Number | $\langle 0, (1-\alpha)\mu \rangle$ | 131.16M | 131.22M | 131.28M | 131.34M | 131.39M |
| $\lvert V \rvert$ = 41.7M | | $\langle \pi_0, r_0 \rangle$ | 4.31M | 6.67M | 8.48M | 9.97M | 10.13M |
| $\lvert E \rvert$ = 1.47B | Runtime | $\langle 0, (1-\alpha)\mu \rangle$ | 481020 | 485240 | 496679 | 502453 | 513854 |
| | | $\langle \pi_0, r_0 \rangle$ | 73159 | 94446 | 107135 | 117000 | 126736 |

The experiment result shows that the maximum gains in terms of iteration number are listed from 52.2% to 96.7%, or 45.2% to 87.3% in terms of runtime, which are consistent with the theoretical analysis. Notice that the runtime gain is a little lower than the iteration number gain, because unlike in PI method where the cost of each iteration is $\Omega(m)$, the cost per each iteration varies in GS method.

## 3.3 Theoretical Analysis of Power-Iteration Method

To estimate the iteration number for the PI method, it is convenient to introduce residual vectors $r^{(t)}$ like in GS method, as follows:

$$
\begin{aligned}
r^{(t)} &= (1-\alpha)\mu - \pi^{(t)}\,(I-\alpha P) \\
&= (\pi - \pi^{(t)})(I-\alpha P) && (3.7) \\
&= \left(\pi - \pi^{(0)}\right)(\alpha P)^t\,(I-\alpha P) \\
&= \left(\pi - \pi^{(0)}\right)(I-\alpha P)(\alpha P)^t \\
&= r^{(0)}(\alpha P)^t && (3.8)
\end{aligned}
$$

As seen from Eq. (3.8), $\pi = \pi^{(t)}$ if and only if $r^{(t)} = 0$. By Eq. (3.7) and (3.8),

$$\pi - \pi^{(t)} = r^{(0)}(\alpha P)^t(I-\alpha P)^{-1} \qquad (3.9)$$

Thus

$$\left\|\pi - \pi^{(t)}\right\|_1 \leq \left\|r^{(0)}\right\|_1 \frac{\alpha^t}{(1-\alpha)} \tag{3.10}$$

Let $\delta$ be a preset error bound. Then, $\left\|\pi - \pi^{(t)}\right\| \leq \delta$ if

$$\left\|r^{(0)}\right\|_1 \frac{\alpha^t}{(1-\alpha)} \leq \delta \tag{3.11}$$

When $\left\|r^{(0)}\right\|_1 > (1-\alpha)\delta$, the iteration number $\sigma$ of PI method is estimated as

$$\sigma = \left\lceil \log \frac{\left\|r^{(0)}\right\|_1}{(1-\alpha)\delta} / \log \alpha^{-1} \right\rceil \tag{3.12}$$

$$\sigma = O\left( \log \frac{\left\|r^{(0)}\right\|_1}{\delta} c \right) \tag{3.13}$$

because from Eq. (3.12), we have $\sigma \geq \frac{\log \frac{\left\|r^{(0)}\right\|_1}{(1-\alpha)\delta}}{\log \alpha^{-1}}$ or $\left\|r^{(0)}\right\|_1 \frac{\alpha^\sigma}{(1-\alpha)} \leq \delta$ or $\left\|\pi - \pi^{(\sigma)}\right\|_1 \leq \delta$. Here c is the constant $\frac{1}{\log \alpha^{-1}}$ which equals to around 14.168.

## 3.4 Complexity Analysis: GS Method vs. PI Method

### 3.4.1 Iteration Number Comparison

We noticed that the convergence criterion used by GS method, i.e., $\|r^{(\tau)}\|_{max} \leq \epsilon$, is different from the $\|\pi - \pi^{(\sigma)}\|_1 \leq \delta$ used by PI method. To compare the iteration numbers between two methods, their convergence criterion should be the same. Certainly, $\epsilon$ should be made much smaller than δ.

Let GS method converges at time $\tau$. That is, $|r^{(\tau)}[i]| \leq \epsilon, i = 1, 2, 3, \dots, n$, resulting in $\|r^{(\tau)}\| \leq n\epsilon$. By Corollary 1, it is clear that the convergence condition $\|\pi - x^{(\tau)}\|_1 \leq \delta$ (as required by PI method) is met if $\|r^{(\tau)}\| \leq (1-\alpha)\delta$ or if $n\epsilon \leq (1-\alpha)\delta$. Thus, as $\epsilon \leq \frac{(1-\alpha)\delta}{n}$, the convergence of GS method assures $\|\pi - x^{(\tau)}\|_1 \leq \delta$, required by PI method. Thus, we set

$$\epsilon = \frac{(1-\alpha)\delta}{n} \tag{3.14}$$

We must emphasize that Eq. (3.14) is sufficient but not necessary in order to assure the convergence condition of PI algorithm being met at time $\tau$. As analyzed in [32], the GS methods iteration number is bounded by:

$$\tau \leq \frac{\|r^{(0)}\|_1}{(1-\alpha)\epsilon} \tag{3.15}$$

Based on Eq. (3.14), the iteration number of GS method can be re-estimated as follows:

$$\tau \leq \frac{\left\|r^{(0)}\right\|_1}{(1-\alpha)\epsilon} \leq \frac{n\left\|r^{(0)}\right\|_1}{(1-\alpha)^2\delta} \tag{3.16}$$

$$\tau = O\left(\frac{n\left\|r^{(0)}\right\|_1}{\delta}\right) \tag{3.17}$$

### 3.4.2 Total Running-time Comparison

In each iteration, PI method needs $m$ multiplications and $m$ (or less) additions, which the complexity is $\Omega(m)$; while GS method averagely needs $\bar{d} = m/n$ multiplications. That is, the running-time of GS method in each iteration is just $1/n$ of the running-time of PI method. Thus, based on Eq. (3.17), the total running-time $\mathbb{P}$ by PI method over the total running-time $\mathbb{G}$ by GS method is derived as follows:

$$\mathbb{P}/\mathbb{G} = O\left(\log\frac{\left\|r^{(0)}\right\|_1}{\delta}c\right)/O\left(\frac{\left\|r^{(0)}\right\|_1}{\delta}\right) \tag{3.18}$$

We'll compare the two algorithms through the experiments conducted in Chapter 6.

# Chapter 4 The Virtual Web for PageRank Computing

On the direction of updating PageRank in dynamic webs, most of the previous works are focusing on PR updating optimization for dynamic link structures, as we have reviewed in Chapter 2. However, as we have showed in Table 1.1, the nodes are also rapidly inserted and removed as time elapsed. An efficient PR updating algorithm to handle the generic case of the web evolvement, in which many link/node insertions and deletions could happen simultaneously, in on demand. In this chapter, we proposed such an algorithm which enables PR updating over the dynamic web graphs with dimensionality changes.

## 4.1 Hierarchical PageRank Computing

### 4.1.1 The Isolated Component

An Isolated Component (IC) in a web graph $G = (V, E)$ is defined as a cluster of nodes that only links to or be linked by other nodes within the same cluster. There's no link connections between the ICs. By this definition, suppose $G$ can be uniquely decomposed into $N$ clusters $G_1, G_2, \dots, G_N$ up to an order

$$G = G_1 \cup G_2 \cup \dots \cup G_l$$

where

$$G_i \cap G_j = \emptyset, \forall i, j \leq N \,\&\, i \neq j$$

Each cluster $G_k$ denotes an IC. A simple example of the isolated components is as Figure 4.1 demonstrated.

### 4.1.2 Local PageRank of the Isolated Component

Given a web graph $G = (V, E)$ that contains $N$ isolated components $G_1, \dots, G_N$, each with size from $n_1, \dots, n_N$. We first reorder the rows and columns of $G$'s transition matrix $P$ in accordance to those isolated components identified, which puts the sub-transition matrices corresponding to each isolated component on the diagonal. It can be written as:

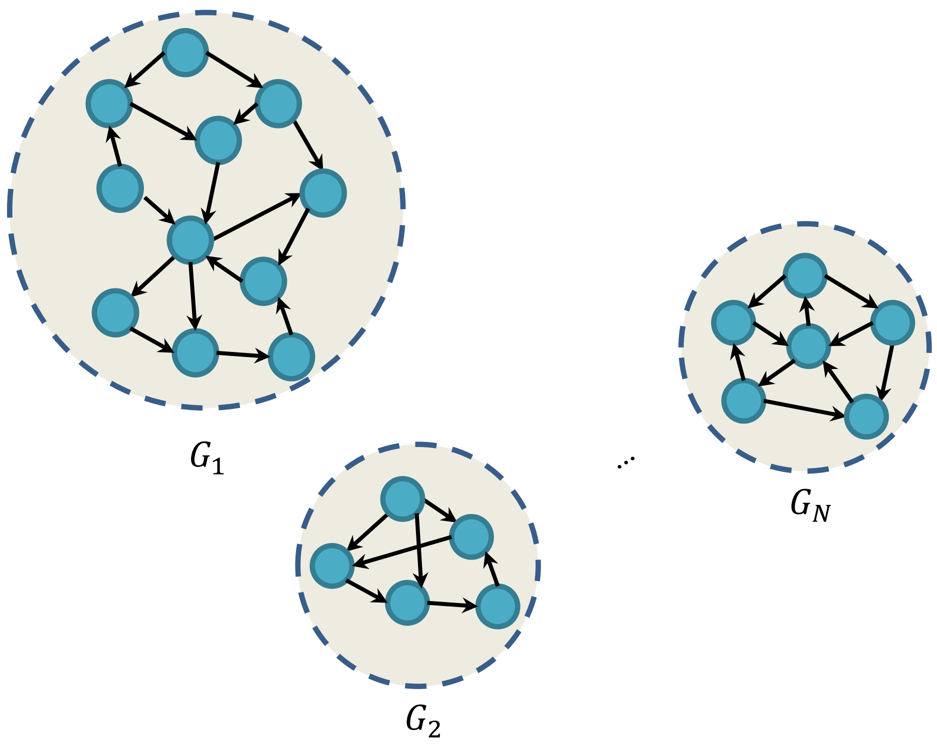


**Figure 4.1** An illustration of the Isolated Components in the web graph

$$P = \begin{bmatrix} P_1 & & & \\ & P_2 & & \\ & & \ddots & \\ & & & P_N \end{bmatrix} \tag{4.1}$$

Each sub-transition matrix $P(k)$ is exactly the local transition matrix for each sub-graph $G_k$ that satisfies Eq. (1.1). Thus, the local PageRank for the sub-graph $G_k$ is defined as:

$$\pi_k = \alpha \pi_k P_k + (1-\alpha)\mu_k \tag{4.2}$$

where $\mu_k = [1/n_k]_{1\times n_k}$.

According to the definition and properties of the PageRank mentioned in Section 1, $\pi_k$ is the local stationary distribution of the sub-graph $G_k$, which denotes the probability of each node being visited by the surfer within $G_k$. We will introduce how the local PageRank $\pi_k$ can be integrated to the global PageRank. We first state the Lemma 7 here:

**Lemma 7.** Given a web graph $G = (V, E)$, which can be decomposed into mutually-disconnected sub-graphs $G_1$, $G_2$,..., $G_N$, each with size from $n_1, \dots, n_N$, then the PageRank vector $\pi$ for $G$ is then given by

$$\pi = \left\{\frac{n_1}{n}\pi_1, \dots, \frac{n_N}{n}\pi_N\right\} \tag{4.3}$$

where $\pi_k$ is the local PageRank of the sub-graph $G_k$ given by Eq. (4.2).

### 4.1.3 Bayes' Rule based Inter-Component PageRank Computing

As defined in PageRank, the probability of a page $i$ that $i \in G_k$ being visited within $G_k$ is represented by $\pi_k[i]$ in the converged ranking vector. However, that probability is conditional, because user will only visit $i$ following probability $\pi_k[i]$ given user in visiting $G_k$. The process can be represented by the conditional probability

$$\pi_k[i] = prob(visit\ i|visit\ G_k)$$

Following the Bayes' rule, the probability of a random page $i \in G$ being visited is:

$$\pi[i] = prob(visit\ i)$$

$$= prob(visit\ G_k) \cdot prob(visit\ i | visit\ G_k), i \in G_k \quad (4.4)$$

Therefore, after the local PageRank computation, the only unknown here is the probability of the isolated component $G_k$ being visited. According to the random suffer model and Eq. (1.2), the web surfers will have a very small probability to jump to a random page without following any links on the current node, which is called random jump. In the structure of Eq. (4.1), it is impossible for the web surfers to skip between isolated components following the links. Therefore, web surfers can only go from one isolated component to another through random jump, which follows the distribution given by $(1-\alpha)\mu$. This implies that the probability of each node being visited by random jump is equal. Hence, the probability of the isolated component $G_k$ being visited by random jump is computed by:

$$prob(visit\ G_k) = \sum_{i \in V_k} (1-\alpha)\mu_i = \frac{n_k}{n} \quad (4.5)$$

Now all the factors corresponding to compute $\pi[i]$ in such decomposed web graph $G$ is clarified. For an arbitrary node $i \in G_k$, $\pi[i]$ is computed by

$$\pi[i] = \frac{n_k}{n} \cdot \pi_k[i]$$

So, for the PageRank $\pi$ of the entire graph $G$, the score is computed by

$$\pi = \left\{\frac{n_1}{n}\pi_1, \dots, \frac{n_N}{n}\pi_N\right\}$$

which is Lemma 6.

Eq. (4.3) can also be proved though linear algebra method as in [3]. We show the proof here for completeness.

***Proof.*** We intend to prove that Lemma 6 stands. First, let us define

$$E' = \begin{bmatrix} (1/n_1)E_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & (1/n_N)E_N \end{bmatrix}$$

where $E_i \in \mathbb{R}_{n_i \times n_i}$ is matrix with all entries equal to one. Let $P$ be reordered according to Eq. (4.1), and assuming $\pi$ is given by Eq. (4.3), we then have

$$\alpha\pi P + (1-\alpha)\mu$$

$$= \pi[\alpha P + (1-\alpha)E' - (1-\alpha)E'] + (1-\alpha)\mu$$

$$= \pi[\alpha P + (1-\alpha)E'] + (1-\alpha)\mu - (1-\alpha)\pi E'$$

$$= \alpha \left[ \left(\frac{n_1}{n}\right) \pi_1 P_1, \dots, \left(\frac{n_N}{n}\right) \pi_1 P_N \right] + (1-\alpha) \left[ \frac{n_1}{n} \mu_1, \dots, \frac{n_N}{n} \mu_N \right]$$

$$= [(n_1/n)\pi_1, \dots, (n_N/n)\pi_N] = \pi$$

which implies that the $\pi$ given by Eq. (4.3) is the PageRank of $G$. Hence Lemma 6 is proved.

The hierarchical PageRank computing with isolated components seems promising: the local PRs can be managed and updated through a distribution system, in which the updating speed can be boosted through parallel computing techniques. It is better than the BlockRank because the size-based weighting scheme is quite simply to implement and at almost zero cost. However, since the contemporary webs are usually highly connected, hence it is idealistic that such well divided isolated components can be always found in real web datasets as we have tested in our experiments. Although not practical in many datasets, its beautiful property as in Eq. (4.3) inspired us and is one of the fundamental theory of our work.

## 4.2 The Virtual Web

How to effectively use the previous PageRank to initialize the calculation of the current PageRank has long been a challenge when node insertion/deletion occurs. One obvious difficulty is the possible web dimensionality change. A *Virtual Web*, which is to be introduced soon, represents an extension to the original web, with some artificial (virtual) pages being inserted, carrying certain specific link structure. The virtual web becomes "equivalent" to the original web in a sense that their PageRank can be straightforwardly calculated one from another with no need for iteration.

### 4.2.1 Virtual Web Design

Let the updated web $W^*$ be evolved from the previous web $W$ through insertion of $a$ new pages and deletion of $d$ old pages plus some link structure modification. We assume no special process for dangling nodes. Let $P$ be the transition matrix of $n \times n$ for $W$ and $\pi$ be its PageRank. Let $P^*$ denote the transition matrix of $n^* \times n^*$ for $W^*$, where $n^* = n + a - d$. Let $\pi^*$ be its PageRank. Without loss of generality, we may organize two transition matrices $P$ and $P^*$ as follows:

$$P = \begin{bmatrix} P_r & P_{(n-d)\times d} \\ P_{d\times(n-d)} & P_d \end{bmatrix} \tag{4.6}$$

$$P^* = \begin{bmatrix} P_r^* & {P^*}_{(n-d)\times a} \\ {P^*}_{a\times(n-d)} & P_a \end{bmatrix} \tag{4.7}$$

where $P_d \in \mathbb{R}_{d\times d}$ is corresponding to the deleted nodes; $P_a \in \mathbb{R}_{a\times a}$ is corresponding to the inserted nodes; $P_r, P_r^* \in \mathbb{R}_{(n-d)\times(n-d)}$ are corresponding to the rest of nodes not being affected by node perturbation, but there could be some link structure modification from $P_r$ to $P_r^*$.

For PageRank updating, we propose a concept called "*Virtual Web*". A virtual web $W_v$ is defined for $W$ by simply adding the virtual transition matrix $P_a$ to $P$ as follows:

$$P_v = \begin{bmatrix} P & 0_{n\times \mathrm{a}} \\ 0_{a\times n} & P_a \end{bmatrix} \tag{4.8}$$

More detailed, $P_v$ is

$$P_v = \begin{bmatrix} P_r & P_{(n-d)\times d} & 0_{(n-d)\times a} \\ P_{d\times(n-d)} & P_d & 0_{d\times \mathrm{a}} \\ 0_{a\times(n-d)} & 0_{a\times d} & P_a \end{bmatrix} \tag{4.9}$$

Here $P_v \in \mathbb{R}_{n_v \times n_v}$, where $n_v = n + a$ is the size of the virtual transition matrix. $\pi_a$ can be calculated in a light-weight fashion due to a relatively small size of $P_a$. By Lemma 6, the virtual web $W_v$ defined by $P_v$ is "equivalent" to $W$ in a sense that their PageRank can be computed from one to another (by Lemma 6):

$$\pi_v = \left(\frac{n}{n+a}\pi, \frac{a}{n+a}\pi_a\right) \tag{4.10}$$

Next, we form the second virtual web $W_v^*$ by integrating the updated web $W^*$ with the deleted nodes $W_d$, that its transition matrix $P_v^*$ defined as

$$P_v^* = \begin{bmatrix} P^* & 0_{n\times d} \\ 0_{d\times n} & P_d^* \end{bmatrix} \tag{4.11}$$

Similarly, in a more detailed notation, $P_v^*$ is

$$P_v^* = \begin{bmatrix} P_r^* & P_{(n-d)\times a}^* & 0_{(n-d)\times d} \\ P_{a\times(n-d)}^* & P_a^* & 0_{a\times d} \\ 0_{d\times(n-d)} & 0_{d\times a} & P_d^* \end{bmatrix} \tag{4.12}$$

where $P_v^* \in \mathbb{R}_{n_v\times n_v}$ is at the same dimension with $P_v$. Like Eq. (4.10) we have

$$\pi_v^* = \left(\frac{n+a-d}{n+a}\pi^*, \frac{d}{n+a}\pi_d\right) \tag{4.13}$$

The updated web $W^*$'s PageRank, $\pi^*$, can be extracted from $\pi_v^*$ by:

$$\pi^* = \frac{n+a}{n+a-d}\hat{\pi}_v^* \tag{4.14}$$

where $\hat{\pi}_v^*$ is the first $n+a-d$ elements in $\pi_v^*$. Figure 4.2 provides an intuitive illustration of the web evolvement and the virtual web. If we reorder $P_v^*$ so its node order is in accordance to $P_v$, that

$$P_v^* = \begin{bmatrix} P_r^* & 0_{(n-d)\times d} & P_{(n-d)\times a}^* \\ 0_{d\times(n-d)} & P_d^* & 0_{d\times a} \\ P_{a\times(n-d)}^* & 0_{a\times d} & P_a^* \end{bmatrix} \tag{4.15}$$

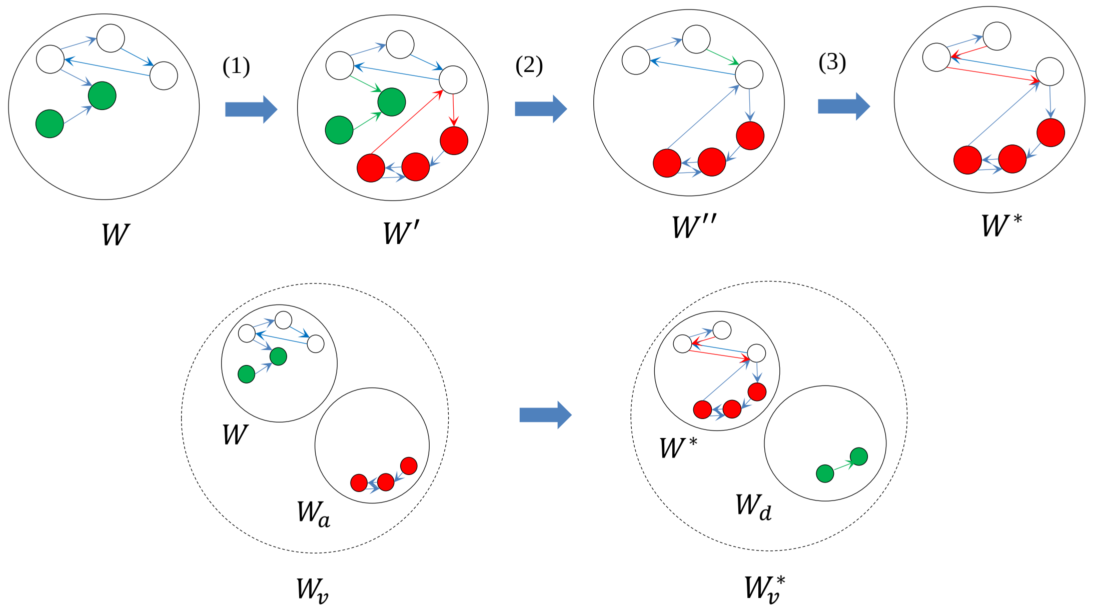


**Figure 4.2** The web evolvement. Up: (1) Node Insertion (red colored): some nodes together with some links are inserted into the original web $W$, resulting in $W'$; (2) Node Deletion (green colored): some nodes together with relevant links removed, resulting in $W''$; (3) Link structure modification (red links inserted and green link removed) occurs among the remaining nodes (white colored), resulting in $W^*$ finally. Down: the inserted nodes $W_a$ and the old web $W$ form the first virtual web $W_v$; Then, $d$ deleted nodes are removed from $W_v$, which forms the virtual web $W_v^*$. $W_v$ and $W_v^*$ share the same set of vertices, the only difference in between lies in their link structures.

It is clearly that the only difference between $P_v$ and $P_v^*$ is their link structures. If $a$ and $d$ are very small comparing to $n$, and the difference between $P_r$ and $P_r^*$ is not dramatic, then by Lemma 2, the PR of $P_v$ can serve as a good initialization to accelerate the computing of the PR of $P_v^*$. And by Lemma 6, the PR of the updated web can be extracted from the virtual web's PR without loss of accuracy. This procedure helps us to utilize the previous PR information for computing the updated PR with node insertion and deletion involved. Hence, how to effectively utilize $\pi_v$ as initialization for computing PR over $P_v^*$ is the problem to be addressed.

### 4.2.2 The Virtual Initialization

To use $\pi_v$ as initialization to compute the PR over $P_v^*$, first we need to re-order its elements in accordance to the order of $P_v^*$. According to the arrangement in Eq. (5.11), the vertex order in $\pi_v$ can be re-arranged with no name change as follows:

$$\pi_v = \left(\frac{n}{n+a}\pi_r, \frac{a}{n+a}\pi_a, \frac{n}{n+a}\pi_d\right) \tag{4.16}$$

where $\pi_r$ is the part of $\pi_v$ corresponding to the nodes remaining after deletion; $\pi_d$ is the other part of $\pi$ which relevant to the deleted nodes; $\pi_a$ is the PageRank of the inserted nodes given by $P_a$.

One can use the re-arranged $\pi_v$ as initialization directly compute updated PR over $P_v^*$. However, the procedure can be further optimized through. Let $\pi_v^* =$

$\{\pi_v^{*\prime}, \pi_v^{*\prime\prime}\}$ where $\pi_v^{*\prime}$ and $\pi_v^{*\prime\prime}$ are corresponding to $P^*$ and $P_d$, respectively, and we have $\mu_v = [1/n_v]_{1\times n_v}$ divided as $\mu_v = \{\mu_v', \mu_v''\}$ in the same way. From Eq. (1.2) and Eq. (5.11),

$$\pi_v^* = \alpha \pi_v^* P_v^* + (1-\alpha)\mu_v \tag{4.17}$$

$$= \alpha\{\pi_v^{*\prime}, \pi_v^{*\prime\prime}\} \begin{bmatrix} P^* & 0_{n\times d} \\ 0_{d\times n} & P_d^* \end{bmatrix} + (1-\alpha)\{\mu_v', \mu_v''\}$$

So that

$$\begin{cases} \pi_v^{*\prime} = \alpha \pi_v^{*\prime} P^* + (1-\alpha)\mu_v' \\ \pi_v^{*\prime\prime} = \alpha \pi_v^{*\prime} P_d^* + (1-\alpha)\mu_v'' \end{cases} \tag{4.18}$$

Eq. (4.18) shows that the calculation of $\pi_v^{*\prime}$ is independent to the transition matrix of the deleted nodes, $P_d^*$. We then simplified the procedure of the virtual web, so the information related to the deleted nodes are disregarded, which avoided the extra calculation and storage.

Now we define a virtual initial $\chi_v$ of $1 \times n^*$, $n^* = n + a - d$ as follows:

$$\chi_v = \left(\frac{n}{n+a}\pi_r, \frac{a}{n+a}\pi_a\right) \tag{4.19}$$

$$\frac{\chi_v}{\|\chi_v\|_1} = \left(\frac{n}{n\|\pi_r\|_1 + a}\pi_r, \frac{a}{n\|\pi_r\|_1 + a}\pi_a\right) \tag{4.20}$$

where $\chi_v/\|\chi_v\|_1$ in Eq. (4.20) will be used to initialize the computation of target PageRank $\pi^*$ over $P^*$ directly.

In summary, the target of the virtual web is to generate an effective initialization for faster PR updating on the new web, which involves node and link insertion/deletion simultaneously from the old web. Since many existing PR updating algorithms are somehow relied on initializations to boost the computing speed, the concept of the virtual web is more like a framework which can be integrated with other existing PR/PPR computing methods to overcome the node insertion/deletion problem in PageRank updating for dynamic networks. We will show two examples of such integration in the following sections.

## 4.3 The Virtual Web Power-Iteration Method

We first show a straightforward initialization-based integration of the virtual web and the Power-iteration method, which we named as *VirtualWebPowerIteration(VWPI)* method and is stated in Algorithm 3.

Experiments are conducted to test the proposed algorithm. We ran Algorithm 3 on 4 different datasets. The details of the experiment can be found in in Chapter 6.

```
Algorithm 3. VirtualWebPowerIteration $(P, P^*, P_a, \pi, \delta)$
Inputs: transition matrix for pervious web $P$, transition matrix for updated web $P^*$, transition matrix for inserted nodes $P_a$, previous PR $\pi$, error bound $\delta$.
1: $\pi_a$ = PowerIteration ($P_a$, $\mu_a$, $\delta$)
2: Create $\pi_r$ by removing the elements relevant to deleted nodes in $\pi$
3: $\chi_v = \left(\frac{n}{n+a}\pi_r, \frac{a}{n+a}\pi_a\right)$
4: $\frac{\chi_v}{\|\chi_v\|_1} = \left(\frac{n}{n\|\pi_r\|_1+a}\pi_r, \frac{a}{n\|\pi_r\|_1+a}\pi_a\right)$
5: $\pi^*$ = PowerIteration ($P^*$, $\frac{\chi_v}{\|\chi_v\|_1}$, $\delta$)
6: return $\pi^*$
```

## 4.4 Virtual Web via Gauss-Southwell Method

We present an algorithm in this section to integrate the virtual web and the Gauss-Southwell method, to helps the latter to handle the dimensionality change problem in PR/PPR computing. However, since in GS method we maintain the residual vector $r$ besides the PR estimation vector $x$, we need to prove that $r$ also satisfies Lemma 6 at first.

### 4.4.1 The Virtual Residual Vector

The integration of the virtual web and Gauss-Southwell method is not as straightforward as the Power-Iteration method because of the residual vector. An intuitive solution is to apply the virtual PageRank $x_v$ estimated by GS method to Eq. (2.5), and calculate the residual vector $r_v$ of the virtual web, for the updating GS computation. However, it is very expensive to compute the product of $x_v(1-\alpha P_v)$ in Eq. (2.5). Here we proved that the residual vector $r_v$ in Gauss-Southwell method also satisfies Lemma 7, so that we can re-assemble $r_v$ from the residual vector of the previous web and the inserted web, to avoid re-computation of the entire residual vector for the virtual web.

**Lemma 8**. Given the PageRank of the first virtual web $P_v$ which is constructed from the previous and inserted PageRank estimated by GS method, $x$ and $x_a$. We have

$$x_v = \left(\frac{n}{n+a}x, \frac{a}{n+a}x_a\right) \tag{4.21}$$

Then the virtual residual vector $r_v$ can be similarly constructed, i.e.,

$$r_v = \left(\frac{n}{n+a}r, \frac{a}{n+a}r_a\right) \tag{4.22}$$

**Proof.** By Eq. (2.5) we have

$$r_v = (1-\alpha)\mu_v - x_v(I-\alpha P_v)$$

where $\mu_v = [1/n_v]_{1\times n_v}$. Thus,

$$r_v = (1-\alpha)\mu_v - \left(\frac{n}{n+a}x, \frac{a}{n+a}x_a\right)\begin{bmatrix} I-\alpha P & 0 \\ 0 & I-\alpha P_a \end{bmatrix}$$

$$= (1-\alpha)\mu_v - \left(\frac{n}{n+a} x(I-\alpha P), \frac{a}{n+a} x_a(I-\alpha P_a)\right)$$

$$= (1-\alpha)\mu_v - \left(\frac{n}{n+a}\big((1-\alpha)\mu - r\big), \frac{a}{n+a}\big((1-\alpha)\mu_a - r_a\big)\right)$$

$$= (1-\alpha)\mu_v - (1-\alpha)\left(\frac{n}{n+a}\mu, \frac{a}{n+a}\mu_a\right) + \left(\frac{n}{n+a} r, \frac{a}{n+a} r_a\right)$$

$$= \left(\frac{n}{n+a} r, \frac{a}{n+a} r_a\right)$$

Thus Eq. (4.22) is proved.

After re-ordering of $P_v$ and $x_v$, the re-ordered $r_v$ with no name change takes the following form:

$$r_v = \left(\frac{n}{n+\mathrm{a}} r_r, \frac{a}{n+a} r_a, \frac{n}{n+a} r_d\right) \tag{4.23}$$

Without loss of generality, we may assume

$$r = (r_r, r_d) \tag{4.24}$$

### 4.4.2 Virtual Initialization for Gauss-Southwell Method

According to [32], Gauss-Southwell initialization $x_v^{*(0)}$ and its corresponding residual vector $r_v^{*(0)}$ for the second virtual web $P_v^*$ is given by

$$x_v^{*(0)} = x_v \tag{4.25}$$

$$r_v^{*(0)} = r_v + \alpha x_v(P_v^* - P_v) \tag{4.26}$$

In Eq. (4.26), the term $x_v(P_v^* - P_v)$ can be further optimized. Firstly, only the rows corresponding to the nodes which have affiliation with the inserted or deleted nodes (i.e., the inserted/deleted nodes themselves, or those nodes pointing to them) are different from $P_v$ to $P_v^*$, while all other rows remain the same, which means the subtract operations for those unchanged rows can be omitted. Since the size of the node insertion/deletion are usually very small in comparison to the size of the entire web, the number of different rows between $P_v$ and $P_v^*$ should be small, therefore the complexity of the subtraction is predicted to be much smaller than $m$. Second, since $P_v^* - P_v$ is expected to be a highly sparse matrix, thus the complexity of the multiplication $x_v(P_v^* - P_v)$ should also be significantly smaller than $m$. Combining these two factors, the usage of Eq. (4.26) could be a better choice than Eq. (2.5) for computing the residual vector.

We want to further optimize our algorithm by omitting the computation for $P_d^*$ like in Section 4.2.2. We confirmed the feasibility of ignoring $P_d^*$ in in GS method by Theorem 1.

**Theorem 1**. Given a directed graph $G = (V, E)$. A path $\mathbb{P}(i{\sim}j)$ from node $i$ to $j$ exists if $j$ can be reached from $i$ through some paths following the links, or $i = j$. We divide $V$ into two subsets $\mathcal{S}$ and $\mathcal{U}$, $\mathcal{S} \cup \mathcal{U} = V$ and $\mathcal{S} \cap \mathcal{U} = \emptyset$, so that there exist no such path $\mathbb{P}(s{\sim}u)$ or $\mathbb{P}(u{\sim}s)$ for $\forall s \in \mathcal{S}$, $u \in \mathcal{U}$. The transition matrix can be written as:

$$P = \begin{bmatrix} P_{\mathcal{S}} & 0 \\ 0 & P_{\mathcal{U}} \end{bmatrix}$$

where $P_{\mathcal{S}}$ and $P_{\mathcal{U}}$ are the sub-matrices corresponding to $\mathcal{S}$ and $\mathcal{U}$, respectively. Then, given initialization $x^{(0)} = \left(x_{\mathcal{S}}^{(0)}, x_{\mathcal{U}}^{(0)}\right)$ and $r^{(0)} = \left(r_{\mathcal{S}}^{(0)}, r_{\mathcal{U}}^{(0)}\right)$, the GS iterations over $\mathcal{S}$ is solely based on $(x_{\mathcal{S}}, r_{\mathcal{S}})$, without being interfered by $(x_{\mathcal{U}}, r_{\mathcal{U}})$.

**Proof.** According to the definitions in Theorem 1, let $P = \begin{bmatrix} P_{\mathcal{S}} & 0 \\ 0 & P_{\mathcal{U}} \end{bmatrix}$. At any iteration $t$ in Algorithm 2, given $i \in V$ that $r^{(t)}[i] > \epsilon$ we have

$$\begin{cases} r^{(t+1)} = r^{(t)} - r^{(t)}[i]e_i(I - \alpha P) \\ x^{(t+1)} = x^{(t)} + r^{(t)}[i]e_i \end{cases}$$

which can be derived as

$$r^{(t+1)} - r^{(t)} = -\left(x^{(t+1)} - x^{(t)}\right)\left(I - \alpha \begin{bmatrix} P_{\mathcal{S}} & 0 \\ 0 & P_{\mathcal{U}} \end{bmatrix}\right) \tag{4.27}$$

Denote $P$'s PR and residual vector given by GS as $x = (x_{\mathcal{S}}, x_{\mathcal{U}})$ and $r = (r_{\mathcal{S}}, r_{\mathcal{U}})$, where $x_{\mathcal{S}}$ and $x_{\mathcal{U}}$ (or $r_{\mathcal{S}}$ and $r_{\mathcal{U}}$) are corresponding to $P_{\mathcal{S}}$ and $P_{\mathcal{U}}$, respectively. So,

$$r_{\mathcal{S}}{}^{(t+1)} - r_{\mathcal{S}}{}^{(t)} = -\left(x_{\mathcal{S}}{}^{(t+1)} - x_{\mathcal{S}}{}^{(t)}\right)(I - \alpha P_{\mathcal{S}}) \tag{4.28}$$

$\forall i \in \mathcal{S}$ that $r_{\mathcal{S}}^{(t)}[i] > \epsilon$, Eq. (4.28) is equivalent to the following iterative process over $P_{\mathcal{S}}$:

$$\begin{cases} r_{\mathcal{S}}^{(t+1)} = r_{\mathcal{S}}^{(t)} - r_{\mathcal{S}}^{(t)}[i] e_i^{\mathcal{S}} (I - \alpha P_{\mathcal{S}}) \\ x_{\mathcal{S}}^{(t+1)} = x_{\mathcal{S}}^{(t)} + r_{\mathcal{S}}^{(t)}[i] e_i^{\mathcal{S}} \end{cases}, i \in \mathcal{S} \tag{4.29}$$

Eq. (4.28) and Eq. (4.29) show that the GS's iterative process is solely based on $x_{\mathcal{S}}^{(t)}$ and $r_{\mathcal{S}}^{(t)}$ over $\mathcal{S}$, without being interfered by $\mathcal{U}$. Thus Theorem 1 is proved, and $\mathcal{U}$ can be discarded when computing the PageRank of $\mathcal{S}$.

In summary, we have the virtual initialization for GS method on the updated web as:

$$x_v = \left(\frac{n}{n + \mathrm{a}} x_r, \frac{a}{n+a} x_a, \frac{n}{n+a} x_d\right) \tag{4.30}$$

$$\hat{r}_v = \left(\frac{n}{n+a} r_r, \frac{a}{n+a} r_a\right) \tag{4.31}$$

$$\hat{x}_v^{*(0)} = \left(\frac{n}{n+a}x_r, \frac{a}{n+a}x_a\right) \tag{4.32}$$

$$\hat{r}_v^{*(0)} = \hat{r}_v + \alpha x_v\left(\hat{P}_v^* - \hat{P}_v\right) \tag{4.33}$$

$P_v$ is re-arranged first in accordance to the order of $P_v^*$. $\hat{P}_v^*$ and $\hat{P}_v$ are matrices of the first $n^*$ columns of $P_v^*$ and $P_v$, respectively, which are

$$\hat{P}_v = \begin{bmatrix} P_r & 0_{(n-d)\times a} \\ 0_{a\times(n-d)} & P_a \\ P_{d\times(n-d)} & 0_{d\times \mathrm{a}} \end{bmatrix} \tag{4.34}$$

$$\hat{P}_v^* = \begin{bmatrix} P_r^* & P_{(n-d)\times a}^* \\ P_{a\times(n-d)}^* & P_a^* \\ 0_{d\times(n-d)} & 0_{d\times a} \end{bmatrix} \tag{4.35}$$

Finally, the updated PR is given by

$$x^* = \frac{n+a}{n+a-d}\hat{x}_v^* \tag{4.36}$$

The computation of $\hat{P}_v^* - \hat{P}_v$ is actually not expensive when the number of inserted nodes and deleted nodes are relatively small, so that the subtraction of the rows corresponding to the unchanged nodes could be omitted. The proposed *VirtualWebGaussSouthwell* (VWGS) approach is summarized in Algorithm 4.

```
Algorithm 4. VirtualWebGaussSouthwell $(P, P^*, P_a, x, r, \epsilon)$
Inputs: transition matrix of pervious web $P$, transition matrix of updated web $P^*$, transition matrix of inserted nodes $P_a$, previous PR $x$, previous residual $r$, error bound $\epsilon$.
1: Initialization $x_a^{(0)} = 0$, $r_a^{(0)} = (1-\alpha)[1/a]_{1\times a}$
2: $\langle x_a, r_a \rangle = GaussSouthwell\ (P_a, x_a^{(0)}, r_a^{(0)}, \epsilon)$
3: Weight $x$, $x_a$, $r$, $r_a$ according to Eq. (4.21) & Eq. (4.22)
4: Compute updated initialization $\hat{x}_v^{*(0)}$, $\hat{r}_v^{*(0)}$ based on Eq. (4.30) to Eq. (4.33)
5: $\langle \hat{x}^*, \hat{r}^* \rangle = GaussSouthwell\ (P^*, \hat{x}_v^{*(0)}, \hat{r}_v^{*(0)}, \epsilon)$
6: Compute $x^* = \frac{n+a}{n+a-d}\hat{x}_v^*$
7: return $x^*$
```

## 4.5 Complexity

Theoretically, the virtual web adds a $O(1/\epsilon + n)$ factor to the complexity of original GS method; or a $O(m_a + n)$ factor to the complexity of the original PI method, where $m_a$ is the number of links among the inserted nodes. However, due to the relatively small size, the PR computations for the inserted nodes are usually very fast in practical. And since usually the factor $1/\epsilon \gg n$ or $m_a$, and $m \gg n$, so virtual web does not change the big-O complexity of the original methods.

# Chapter 5 An Optimization of the Virtual Web Gauss-Southwell Algorithm

We proposed a further optimization of the Algorithm 4. As the we have concluded in Lemma 4 at section 3.2, the norm of the initial residual vector $\left\|r^{(0)}\right\|_1$ decided the convergence speed of the updating computation. From Eq. (2.7), since the previous estimation $x$ and residual vector $r$ are given, thus the term $P^* - P$ is only factor that could affect the norm of the initial residual vector. Since in our setup, there exist only link modification from the first virtual web $P_v$ to the second virtual web $P_v^*$, which decides the initial residual vector $r_v^{*(0)}$. We are curious about how the link modification could affect the residual vector and PR estimation $x$ in an evolving graph.

### 5.1.1 The Effect of the Link Perturbation to GS Estimation

To study the effect of the link structure modification to the PR estimation of the GS method, we conducted two preliminary experiments to track the evolvement of the PR values of those nodes, which are directly affected by the link perturbation, i.e. a link insertion or deletion.

For a given dataset $G = (V, E)$, we firstly remove a random link $(u, v)$ from $E$ and treat this web graph as the original web graph, then run GS method from scratch to get the original PR estimation $x$; then, we add this link back which forms the updated web graph, and we run GS method from scratch again to calculated the updated PR estimation $x^*$. The experiment procedure for link deletion is just reversed.

The experiments are conducted on web-Google dataset and eu-2005 dataset. We select 100 random links in each dataset for insertion (or deletion), and compute the original and updated PR estimation accordingly. During this process, by Eq. (2.6), the nodes with PR and residual affected directly by this modification are node $u$, node $v$ and $u$'s out-neighbors $q \in \mathcal{N}^{out}(u)$. We monitor the PR estimation value change in term of $l_1$-distance of these nodes from $x$ to $x^*$, i.e. $\|x^*[q] - x[q]\|_1$ . The results are listed in Table 5.1.

The results show clearly that the single link $(u, v)$'s insertion or deletion caused more PR value change averagely on $v$ than $u$ and $u$'s out-neighbors. We plot the specific data of the 100 link insertions and deletions for each dataset in Figure 5.3 for a more intuitive view. The figure shows clearly that in the vast majority of the link insertion or deletion cases in both datasets, *v*'s PR value changed significantly, while *u* and *u*'s out-neighbors' PR values remain almost invariant. In majority cases, *v*'s PR value changed around 2~7.5 times more than *u*'s out-neighbors' PR values. We summarized our observation, that in a directed dynamic web graph, the PR values of the end nodes of the inserted/deleted links are more likely to be changed significantly, while the PR values of the start nodes (and their out neighbors) of those links keep invariant.

**Table 5.1: Average PR Change by GS Method over 100 Single Link Insertions/Deletions**

| | Link Insertion | | Link Deletion | |
|---|---|---|---|---|
| Dataset | eu-2005 | web-Google | eu-2005 | web-Google |
| *u*'s out-neighbors | 5.98E-09 | 3.05E-09 | 1.18E-09 | 5.93E-09 |
| *u* | 2.48E-09 | 5.50E-10 | 7.43E-10 | 4.61E-09 |
| *v* | 9.58E-08 | 3.98E-08 | 1.69E-08 | 7.36E-08 |

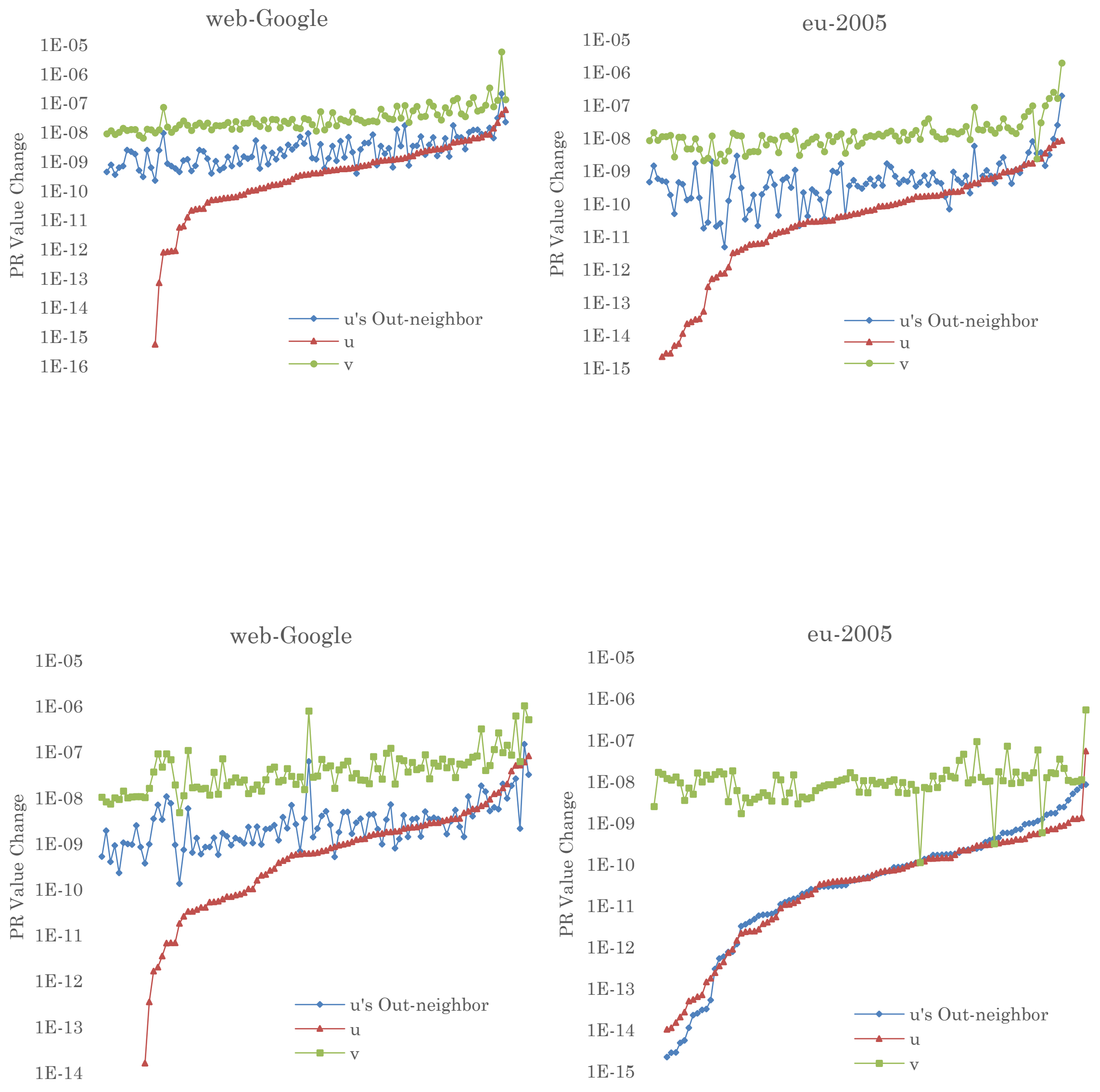


**Figure 5.1** The PR value change after a single link insertion or deletion. The PR values are estimated by GS method. Each data point is the $l_1$-distance between the PR values before and after a random link $(u, v)$ is inserted (or removed). The PR change for $u$'s out-neighbors is measured by the average PR change of the nodes linked by $u$, excluding $u$ and $v$. The data is sorted according to $u$'s PR change. Up: 100 random link insertions; Down: 100 random link deletions.

Based on the experimental observations and line 3 in Algorithm 2 (the GS method), since the PR value in $x$ is one-way accumulated from $r$, an increased distance from $x[i]$ to $x^*[i]$ implies an increased difference between $r[i]$ and $r^*[i]$ at first; similarly, an invariant PR value in $x$ means it is likely that the corresponding residual in $r^*$ also remains invariant from $r$. Therefore, when forming the initial residual vector for the updated web, our goal is to find an ${r^*}^{(0)}$ that better fits the experimental observation here, so ${r^*}^{(0)}$ could become closer to the converged residual vector of the updated PR.

On this direction, we noticed a drawback of the previous GS updating algorithm with the updated initialization given in Eq. (2.7). Assuming that a link $(u, v)$ is inserted from $P$ to $P^*$, therefore, each non-zero element (links pointing to $u$'s out-neighbors) in the $u$-th row of $P^*$ is alternated from $1/o_u$ to $1/(o_u + 1)$. Thus, those elements in the $u$-th row of the matrix $P^* - P$ becomes non-zero. Consequently, the product $\alpha x(P^* - P)$ will add those non-zeros to the updated initial residual vector $r^{*(0)}$ in Eq. (2.7), that may increase the norm of the residuals of $u$'s out-neighbors, which breaks the invariant and induced to more iterations. Hence, we intend to optimize this procedure by avoiding such increment of the residuals of $u$'s out-neighbors other than $v$.

### 5.1.2 An Optimized Initialization for Virtual Web via GS Method

We proposed an optimized initialization scheme here for VWGS method to better handle the node insertion cases. This is meaningful, because the WWW and social networks and fast-growing webs. Like illustrated in Table 1.1, node insertions are more commonly happened in those networks. The optimized initialization is computed based on Eq. (2.6) and Eq. (4.33). We use another modified virtual PR estimation, $\tilde{x}_v$, to cancel out the increment of the residuals for the out-neighbors of the nodes with link perturbations. Briefly, the optimized virtual initialization is given by

$$\hat{\hat{r}}_v^{*(0)} = \hat{r}_v - (\hat{\tilde{x}}_v - \hat{x}_v) + \alpha\big(\tilde{x}_v\hat{P}_v^* - x_v\hat{P}_v\big) \tag{5.1}$$

in which

$$\tilde{x}_v = x_v\mathcal{W} \tag{5.2}$$

$\mathcal{W}$is the diagonal matrix that

$$\mathcal{W}_{ii} = \begin{cases} \dfrac{o_{v_i}^*}{o_{v_i}}, \forall i < n-d \\ 1, \quad otherwise \end{cases} \tag{5.3}$$

where $o_{v_i}$ and $o_{v_i}^*$ are the out-degrees of node $i$ in the first and second virtual webs, respectively.

We hypothesis the proposed optimization better suits the PR updating for node insertion. Firstly, for the node $u$ with link perturbation from $P_v$ to $P_v^*$, $\mathcal{W}$ makes the elements in the product $\alpha x_v(\mathcal{W}\hat{P}_v^* - \hat{P}_v)$ corresponding to $u$'s out-neighbors equal to 0, so the invariance in $\hat{\tilde{r}}_v^{*(0)}$ is maintained for $u$'s out-neighbors. Secondly, the term $\hat{r}_v - (\hat{\tilde{x}}_v - \hat{x}_v)$ is likely to decrease $u's$ residual in $\hat{\tilde{r}}_v^{*(0)}$, because $\hat{r}_v[u] \geq 0$ and $\hat{\tilde{x}}_v[u] - \hat{x}_v[u] > 0$ for link insertion (which is induced by node insertion). Therefore, the invariance of $u$'s value in $\hat{\tilde{r}}_v^{*(0)}$ could also be maintained, which matches with the experimental observation. Thus, $\tilde{x}_v$ could be a closer estimation to the PR of the second virtual web than $x_v$ for node insertion.

Eq. (5.1) can be efficiently computed because the rows corresponding to the nodes without any changes from $P_v$ to $P_v^*$ can be set to 0 in $\hat{P}_v^*$ and $\hat{P}_v$. Notice that such kind of nodes are abundant because we assumed all perturbations are in relatively small scale, thus the cost of computing $\tilde{x}_v\hat{P}_v^* - x_v\hat{P}_v$ is greatly reduced. Also, the elements in $(\hat{\tilde{x}}_v - \hat{x}_v)$ corresponding to the same set of nodes can be set to 0 without any calculation. The details of the optimized VWGS method can be found in Algorithm 5.

**Algorithm 5**. *VirtualWebGaussSouthwellOptimized*$(P, P^*, P_a, x, r, \epsilon)$

```
Inputs: transition matrix of pervious web $P$, transition matrix of updated web $P^*$, transition matrix of inserted nodes $P_a$, previous PR $x$, previous residual $r$, error bound $\epsilon$.
1: Initialization $x_a^{(0)} = 0$, $r_a^{(0)} = (1-\alpha)[1/a]_{1\times a}$
2: $\langle x_a, r_a \rangle = GaussSouthwell\ (P, x_a^{(0)}, r_a^{(0)}, \epsilon)$
3: Weight $x$, $x_a$, $r$, $r_a$ according to Eq. (5.21) & Eq. (5.22)
4: Compute the weight vector $\omega$ according to Eq. (5.39), and then $\hat{\hat{x}}_v^{*(0)} = \hat{\hat{x}}_v$
   based on Eq. (5.38).
5: Compute updated initialization $\hat{\hat{r}}_v^{*(0)}$ from Eq. (5.37)
6: $\langle \hat{x}_v^*, \hat{r}_v^* \rangle = GaussSouthwell\ (P^*, \hat{\hat{x}}_v^{*(0)}, \hat{\hat{r}}_v^{*(0)}, \epsilon)$
7: Compute $x^* = \frac{n+a}{n+a-d} \hat{x}_v^*$
8: return $x^*$
```

We conduct experiments to test the gain and accuracy of the optimized virtual initialization $\hat{\hat{r}}_v^{*(0)}$ in Section 6.5, for updating node insertions.

# Chapter 6 Experiments

We conducted experiments in this chapter to test the performance of the virtual web and its optimizations. All experiments are based on real-world datasets for credibility and generality.

## 6.1 Experiment Setup

### 6.1.1 Datasets

We list the details of the datasets we used in the experiments here. We use four real-world datasets as shown in Table 6.1. The eu-2005 and twitter-2010 datasets are acquired from Laboratory for Web Algorithmics [8, 9, 11] and the rest datasets are obtained from the Stanford Large Network Dataset Collection [27].The selection of the datasets covers the web graphs and social graphs. The statistics of the datasets are shown in Table 6.1.

For storage feasibility, the datasets are stored in Compressed Sparse Row (CSR) format on disk. Each adjacency matrix is loaded into memory as the Sparse Matrix format provided by the Eigen library, a C++ based open-source linear algebra library.

**Table 6.1: Statistics of the Datasets**

| Dataset | $\lvert\bar{V}\rvert$ | $\lvert\bar{E}\rvert$ |
|---|---|---|
| web-Google | 0.91M | 5.1M |
| eu-2005 | 0.86M | 19.2M |
| soc-LiveJournal1 | 4.8M | 69M |
| Twitter-2010 | 4.17M | 1.47B |

### 6.1.2 Dataset Preparation

Since the history log of the evolvement of the datasets are not provided, we have to make random modifications on those datasets to simulate the web evolving process, i.e., node/link insertion and deletion.

***Node Insertion/Deletion***: We firstly generate a random seed and start a Breadth-First-Search from it to fetch a subset of nodes of certain size $a$, to simulate the process of new node discovery by web crawlers like in the real web search engines. This subset of nodes is treated as the nodes to be inserted. We then remove that subset of nodes, and re-organize the rest of nodes in a random order, which forms the original web before any node insertion or deletion. Next, we remove the last *d* nodes in the original web to simulate the node deletion process, and add the nodes to be inserted back which creates the updated web with both node insertion and deletion from the original web. For node insertion in multiple proportions, we first remove all nodes to be inserted to generate the original web, and add partial of those nodes back gradually. The process is illustrated in Figure 6.1.

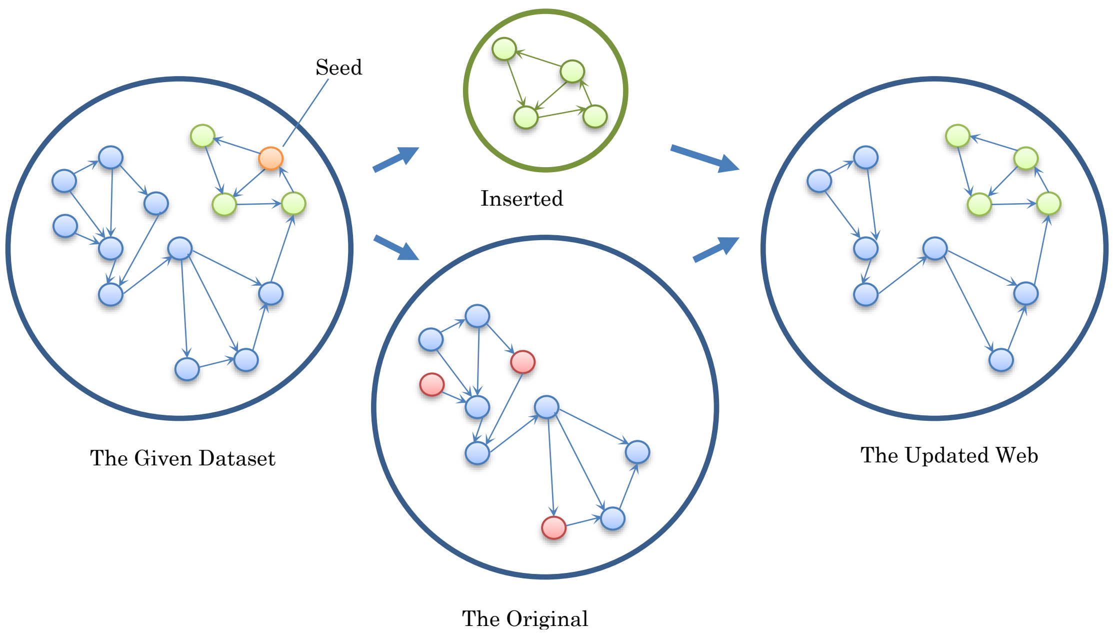


**Figure 6.1** Dataset generation. Left: the given dataset; Middle: the inserted nodes are selected from a seed (orange colored at left) through BFS, and the rest of nodes consists the original web; Right: some randomly selected nodes in the original graph (red colored at middle) are deleted, while the inserted nodes are included together with the links to form the updated web.

***Link Insertion/Deletion***: the link insertion/deletion follows the similar procedure as the node insertion/deletion, that we randomly choose some links and remove them in the original web, then add those links back and remove another set of links in the updated web. Notice that the link insertion/deletion is only done among the remaining nodes which are neither inserted nor removed, e.g., the blue colored nodes in Figure 6.1. We do not assume link perturbations among the inserted nodes because the link structure for the inserted nodes should be what it is when discovered by the crawler for the current web.

### 6.1.3 Experiment Environment

All our experiments are conducted on a Linux workstation with an Intel Xeon E5-2680 2.7GHz CPU, with 64 GB memory. The algorithms are implemented in C++ 11 using the Eigen library[1], and complied with g++ 5.4 with –O3 and –fopenmp option.

## 6.2 Performance of Virtual Web Power-Iteration Method

We conduct experiments to test the performance of the Virtual Web Power-Iteration method for general web evolvement with both node and link insertion/deletion. For each dataset, we generate 25 datasets of the combination of node insertions from 0.5% to 2.5%, for every 0.5%, and node deletions of the same proportion, which makes a combined node perturbation from 1% to 5%.

[1] The Eigen library is retrieved from: http://eigen.tuxfamily.org

A link perturbation combined 0.5% link insertion and 0.5% link deletion is also involved, which together makes a 1% link perturbation. We run VWPI algorithm proposed as Algorithm 3, and compared it with the regular PI method start from scratch. The run-time of computing the PR of the inserted node is counted in the total run-time. The results are listed in Table 6.2, in which the "average" is the average performance over the 25 datasets; "min" is the performance on the dataset with least node perturbation, with 0.5% node insertion and 0.5% node deletion; "max" is the performance on the dataset with most node perturbation, with 2.5% node insertion and 2.5% node deletion. The error bound $\delta = 10^{-3}$.

The result in Table 6.2 showed that among the four datasets, the proposed VWPI method is averagely 47% ~65% faster on run-time (or 49%~65% on number of iterations) than the PI method from scratch (PI initialized from $\mu$). For the smallest perturbation case on each dataset, which contains node and link perturbation each of 1%, the gains VWPI method achieved are listed from 60% ~69% on run-time (or 59% ~69% on iteration number).

## 6.3 Performance of Virtual Web Gauss-Southwell Method

We also conduct experiments to test the performance of the Virtual Web Gauss-Southwell method for general web evolvement case with node and link insertion/deletion. We used the same setup as in Section 6.2, except to set the error bound as $\epsilon = 10^{-9}$. The results are listed in Table 6.3.

**Table 6.2 VWPI with $\chi_v/\|\chi_v\|_1$ and $\mu^*$ Initializations**

| Dataset | Initial | Metric | Performance | | |
|---|---|---|---|---|---|
| | | | Average | Min | Max |
| web-Google | $\chi_v/\|\chi_v\|_1$ | Iteration Num | 11 | 9 | 13 |
| | | Run-time (ms) | 1092 | 818 | 1481 |
| | $\mu^*$ | Iteration Num | 22 | 22 | 22 |
| | | Run-time (ms) | 2050 | 2024 | 2035 |
| eu-2005 | $\chi_v/\|\chi_v\|_1$ | Iteration Num | 9.6 | 8 | 11 |
| | | Run-time (ms) | 2497 | 2085 | 2891 |
| | $\mu^*$ | Iteration Num | 19 | 19 | 19 |
| | | Run-time (ms) | 4918 | 4906 | 4884 |
| soc-LiveJournal | $\chi_v/\|\chi_v\|_1$ | Iteration Num | 5.8 | 5 | 7 |
| | | Run-time (ms) | 10161 | 8168 | 12876 |
| | $\mu^*$ | Iteration Num | 15 | 15 | 15 |
| | | Run-time (ms) | 24003 | 24982 | 21279 |
| Twitter-2010 | $\chi_v/\|\chi_v\|_1$ | Iteration Num | 5.6 | 5 | 6 |
| | | Run-time (ms) | 270998 | 251292 | 284803 |
| | $\mu^*$ | Iteration Num | 16 | 16 | 16 |
| | | Run-time (ms) | 772538 | 802484 | 782502 |

**Table 6.3. VWGS with $\hat{x}_v^{*(0)}$, $\hat{r}_v^{*(0)}$ and 0 Initializations**

| Dataset | Initial | Metric | Performance | | |
|---|---|---|---|---|---|
| | | | Average | Min | Max |
| web-Google | $\hat{x}_v^{*(0)}$, $\hat{r}_v^{*(0)}$ | Iteration Num | 3.40M | 1.64M | 5.43M |
| | | Run-time (ms) | 717 | 370 | 1081 |
| | 0 | Iteration Num | 7.69M | 7.69M | 7.69M |
| | | Run-time (ms) | 1488 | 1475 | 1498 |
| eu-2005 | $\hat{x}_v^{*(0)}$, $\hat{r}_v^{*(0)}$ | Iteration Num | 1.39M | 0.86M | 1.76M |
| | | Run-time (ms) | 607 | 381 | 746 |
| | 0 | Iteration Num | 5.05M | 5.08M | 5.04M |
| | | Run-time (ms) | 1581 | 1753 | 1532 |
| soc-LiveJournal | $\hat{x}_v^{*(0)}$, $\hat{r}_v^{*(0)}$ | Iteration Num | 7.09M | 3.98M | 9.76M |
| | | Run-time (ms) | 6681 | 3868 | 9010 |
| | 0 | Iteration Num | 38.8M | 38.8M | 38.8M |
| | | Run-time (ms) | 24507 | 24783 | 24201 |
| Twitter-2010 | $\hat{x}_v^{*(0)}$, $\hat{r}_v^{*(0)}$ | Iteration Num | 8.35M | 4.14M | 11.64M |
| | | Run-time (ms) | 174859 | 146579 | 201840 |
| | 0 | Iteration Num | 100.57M | 100.57M | 100.57M |
| | | Run-time (ms) | 424084 | 428658 | 422442 |

The result in Table 6.3 shows that among the four datasets, the proposed VWGS method is averagely 51%~72% faster on run-time (or 55%~91% on number of iterations) than the GS method from scratch (GS initialized from 0). For the smallest perturbation case on each dataset, which contains node and link perturbation each of 1%, the gains VWGS method achieved are listed from 66% ~84% on run-time (or 79% ~96% on iteration number).

## 6.4 PI Method vs. GS Method

We compare the PI method and GS method here as stated in Section 3.4. To fairly compare GS method to PI method, we first run GS method with error bound $\epsilon$ till convergence at iteration $\tau$. Then by Corollary 1, we set the error bound of PI method as $\delta = \left\|r^{(\tau)}\right\|_1/(1-\alpha)$. We compared the run-time of the PI and GS method initialized by $\mu$, for link insertions from 0% (the original web) to 5%.

**Table 6.4 Comparison between GS and PI Initialized by $\mu$**

| Dataset | Method | Link Insertion(Run-time, ms) | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0% | 1% | 2% | 3% | 4% | 5% |
| web-Google | PI-Runtime | 3424 | 2820 | 2704 | 2803 | 2742 | 2883 |
| | GS-Runtime | 1565 | 1377 | 1259 | 1284 | 1276 | 1396 |
| Twitter-2010 | PI-Runtime | 446300 | 469330 | 473025 | 478445 | 482518 | 486136 |
| | GS-Runtime | 239312 | 227510 | 237730 | 238572 | 248360 | 240609 |

Result listed in Table 6.4 shows that practically GS method run faster than PI method in call cases, from 46% to 54%. And from the results in Table 6.2 & 6.3, the VWGS also run faster than VWPI by 34% ~ 75%.

## 6.5 VWGS Method vs. VWGS-Optimized Method

We compared the VWGS method and VWGS-Optimized method for node insertions on the web-Google dataset and soc-LiveJournal dataset. We create 10 datasets for each given dataset, each with node insertion from 2% to 20%, for every 2%. We run the VWGS method, VWGS-Optimized method, and GS method from scratch (initialized from 0 vector) and compared them in terms of run-time, iteration number and average accuracy. The average accuracy is measured by

$$Average\ Accuracy = \left\| x_{VWGS\_Optimized} - x_{GS} \right\|_1 / n \tag{6.1}$$

where the term $x_{Proposed}$ is the PR estimation given by VWGS or VWGS-Optimized method, and $x_{GS}$ is the PR estimation given by the GS method from scratch.

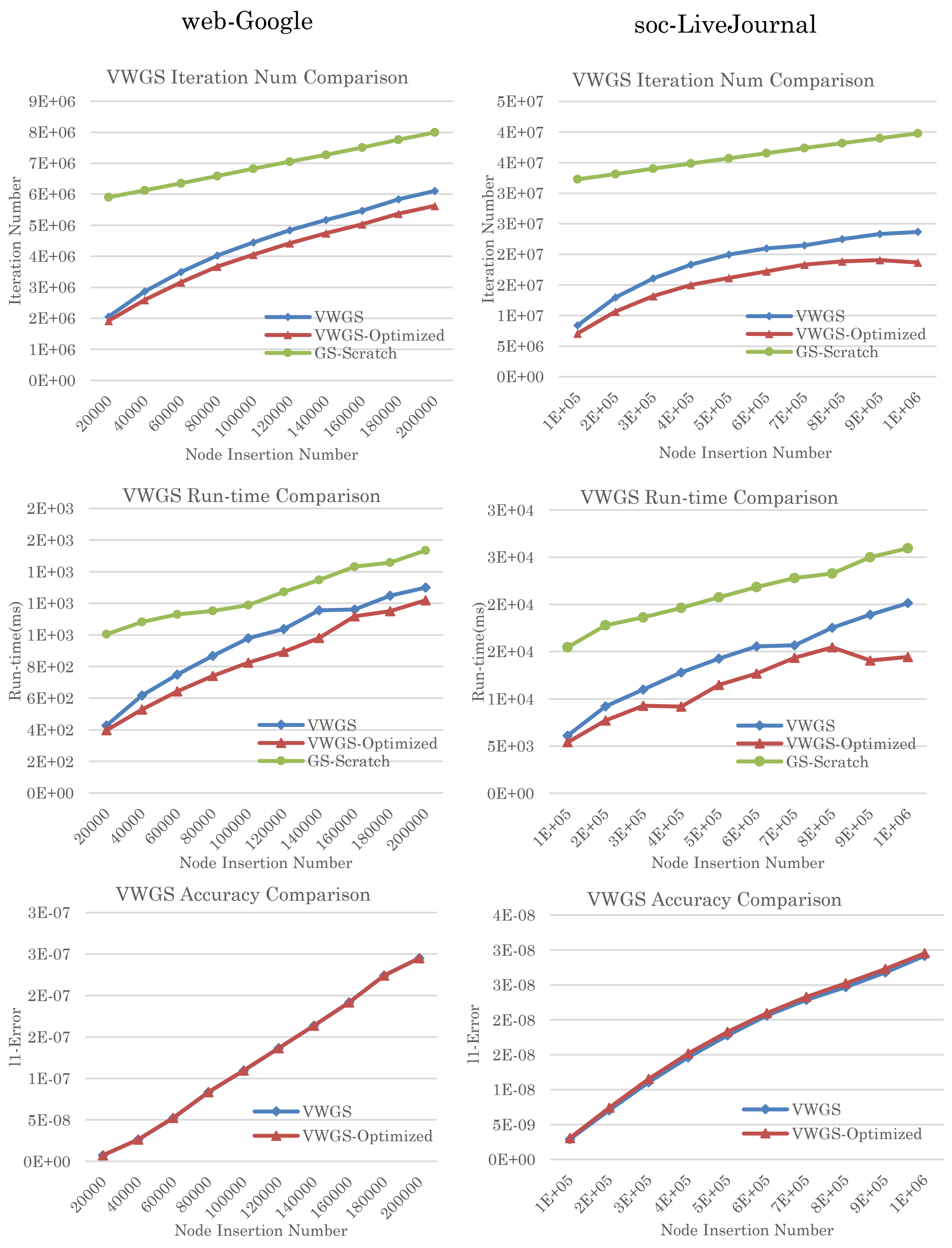


**Figure 6.2** Comparison of the VWGS and VWGS-Optimized algorithms. Left column: web-Google dataset; Right column: soc-LiveJournal dataset.

The result in Figure 6.2 demonstrates clearly that VWGS-Optimized algorithm out-performed VWGS method in all datasets. Even when 20% nodes are inserted, the VWGS-Optimized method shows significant gain in comparison to VWGS method in terms of both iteration number and run-time. At the same time, VWGS-Optimized maintained almost exactly the same accuracy with VWGS method. The average gains over the soc-LiveJournal dataset are 15.4%~21.1% on iteration number and 11.7%~28.3% on run-time.

## 6.6 Comparison to the TrackingPPR Method

It is difficult to directly compare our virtual web algorithms to the previous works, most of which are focusing on only link structure modifications. Here we compare the proposed algorithm, VWGS, to the prior work TrackingPPR method proposed in [32]. We compared the results of our experiment conducted on 3 datasets, each with 1% link perturbation plus 1% node perturbation, to the results reported in [32]. It is a fair comparison because: (1) our experiment environment is almost the same with [32], with the same datasets; (2) There's no difference between PR & PPR for GS method during updating, the difference only occurs at the initialization from scratch; (3) for fairness, we do not take the link perturbation caused by the node insertion/deletion into account for our experiment, because the TrackingPPR is only able to handle the link modifications. The comparison of the average updating time per each link insertion or deletion is listed in Table 6.5. The run-time of TrackingPPR is measured by the average of the time cost per single link insertion and deletion, because both link insertions and deletions are involved in our experiment.

From the comparison, the VWGS has the same average time cost as TrackingPPR on the web-Google dataset; about 3 times faster on the soc-LiveJournal dataset; and more than 1800 times faster on the Twitter-2010 dataset. More importantly, the VWGS method can handle node insertions/deletions while the TrackingPPR cannot. From the comparison, we suggest using small-batch based updating instead of single-link based updating for tracking PR/PPR in long term. To make VWGS adaptive to PPR is remained as the future work.

**Table 6.5 Comparison of VWGS and TrackingPPR on Run-time ($\mu s$)**

| Dataset | VWGS | TrackingPPR |
|---|---|---|
| web-Google | 7.17 | 7.35 |
| soc-LiveJournal | 5.53 | 18.4 |
| Twitter-2010 | 10.47 | 19468 |

# Chapter 7 Conclusions

## 7.1 Conclusion of Our Work

In this thesis, a new concept named virtual web is introduced to solve the general case of PageRank updating, which allows both node insertion/deletion and link insertion/deletion. The virtual initializations generated from the proposed scheme can be easily integrated with other mainstream PR/PPR updating approaches, for the latter to overcome the node insertion/deletion problem. The proposed Virtual Web Power-Iteration(VWPI) algorithm and Virtual Web Gauss-Southwell (VWGS) algorithm significantly out-performed the original Power-Iteration and Gauss-Southwell method, respectively. To our knowledge, this is the first PR updating algorithm that allows a large number of nodes being inserted and deleted simultaneously: on the largest dataset we used, the

Twitter-2010 dataset, the scale of the perturbation is from 0.4M to 2M node perturbation with 14M link perturbation at one time.

The experiment results conducted on real-world datasets show that on the twittwer-2010 dataset which contains 42 million nodes and 1.47 billion links, with a perturbation of 400,000 node insertions plus deletions with more than 14 million link insertions plus deletions, the proposed VWGS method runs 3 times faster than the Gauss-Southwell algorithm starting from scratch, or 20 times faster in terms of iteration number. The updating speed per single link insertion/deletion is 1800x faster than the prior work.

## 7.2 Future Work

### 7.2.1 Virtual Web for PPR

We are aiming to deliver a faster Personalized PageRank (PPR) computation scheme in the future, since the virtual web framework can be easily adopted to address the node insertion/deletion problem in PPR updating. Under such context, Lemma 8 may need to be re-proved to make GS method available for the virtual web in PPR. We need to make a detailed comparison of our algorithms to Zhang et al.'s work [42], which allows only 2 node insertions or deletions at a time for PPR updating.

### 7.2.2 Optimized VWGS for Node Deletions

From the preliminary experiment conducted in Chapter 5, the effect of the link deletion is similar to link insertion. However, as per Eq. (5.1), for a removed link $(u, v)$, the weight $\omega$ proposed in Eq. (5.2) will increase $u$'s residual in the initial virtual residual vector $\hat{\hat{r}}_v^{*(0)}$, which conflicts with experimental observation. Thus, how to further optimize the VWGS method for node deletion is another future task.

# BIBLIOGRAPHY

[1] Abiteboul, S., Preda, M., Cobena, G. Adaptive On-Line Page Importance Computation. WWW2003.

[2] Andersen, R., Chung, F., Lang, K. Local graph partitioning using pagerank vectors. FOCS. 2006:475-86.

[3] Avrachenkov, K., Litvak, N. Decomposition of the Google PageRank and Optimal Linking Strategy. INRIA; 2004.

[4] Avrachenkov, N., Litvak, N., Nemirovsky, D., Osipova, N. Monte-Carlo methods in pagerank computation: When one iteration is sufficient. SIAM Journal on Numerical Analysis. 2007;45(2):890-904.

[5] Backstrom, L., Leskovec, J., editors. Supervised random walks: predicting and recommending links in social networks. The fourth ACM international conference on Web search and data mining; 2011.

[6] Bahmani, B., Chwdhury, A., Goel, A. Fast Incremental and Personalized PageRank. VLDB. 2010;4(3):173-84.

[7] Bahmani, B., Kumar, R., Mahdian, M., Upfal, E. PageRank on evolving graph. KDD. 2012.

[8] Berkhin, P. Bookmark-coloring approach to personalized PageRank computing. Internet Math. 2006;3(1):41-62.

[9] Boldi, P., Vigna, S. The WebGraph Framework: Compression Techniques: ACM Press; 2004.

[10] Boldi, P., Posenato, R., Santini, M., Vigna, S. Traps and Pitfalls of Topic-Biased PageRank. WAW2006.

[11] Boldi, P., Rosa, M., Santini, M., Vigna, S. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks: ACM Press; 2011.

[12] Brin, S., Page, L. The anatomy of a large-scale hypertextual web search engine. Computer Networks ISDN System. 1998;30(1):107-17.

[13] Chen, Y.-Y., Gan, Q., Suel, T. Local Methods for Estimating PageRank Values. CIKM2004.

[14] Chien, S., Dwork, C., Kumar, R., Sivakumar, D. Link evolution: Analysis and algorithms. Internet Math. 2004;1(3):277-304.

[15] Gleich, D. F. PageRank Beyond the Web. SIAM Review. 2015;57(3):321-63.

[16] Google Inside Search - Algorithms: https://www.google.com/intl/en_us/insidesearch/howsearchworks/algorithms.html

[17] Gupta, P., Goel, A., Lin, J., Sharma, A., Wang, D., Zadeh, R., editors. Wtf: The who to follow service at twitter. Proceedings of the 22nd international conference on World Wide Web; 2013: International World Wide Web Conferences Steering Committee.

[18] Haveliwala, T. H. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. IEEE Transactions on Knowledge and Data Engineering. 2003;15(4):784-96.

[19] Internetlivestats.Com - http://www.internetlivestats.com/.

[20] Ipsen, I. C. F., Selee, T. M. PageRank Computation, with Special Attention to Danglingnodes. SIAM Journal on Matrix Analysis and Applications. 2007;29(4):1281-96.

[21] Jeh, G. W., Jennifer, editor Scaling personalized web search. Proceedings of the 12th international conference on World Wide Web; 2003: ACM.

[22] Kamvar, S. D., Haveliwala, T. H., Manning, C. D., Golub, G. H. Exploting the Block Structure of the Web for Computing PageRank. Technical Report, Stanford University. 2003.

[23] Kolda, T. G., Procopio, M. J. Generalized BadRank with Graduated Trust. Sandia National Laboratories; 2009.

[24] Langville, A. N., Meyer, C. D. Deeper Inside PageRank. Internet Mathematics. 2004;1(3):335-80.

[25] Langville, A. N., Meyer, C. D. Updating markov chains with an eye on google's pagerank. SIAM Journal on Matrix Analysis and Applications. 2006;27(4):968-87.

[26] Lee, C. P.-C., Golub, G. H., Zenios, S. A. A Fast Two-Stage Algorithm for Computing PageRank and Its Extensions. Technical Report. Stanford University; 2003.

[27] SNAP Datasets: Stanford Large Network Dataset Collection [Internet]. 2014. Available from: http://snap.stanford.edu/data.

[28] Lofgren, P., Banerjee, S., Goel, A., Seshadhri, C. FAST-PPR: Scaling Personalized PageRank Estimation for Large Graphs. KDD. 2014.

[29] Lofgren, P. Efficient Algorithms for Personalized PageRank: Stanford University; 2015.

[30] Lofgren, P., Banerjee, S., Goel, A. Bidirectional PageRank Estimation-From Average-Case to Worst-Case. WAW 2016. 2015.

[31] Mcsherry, F. A Uniform Approach to Accelerated PageRank Computation. WWW. 2005.

[32] Ohsaka, N., Maehara, T., Kawarabayashi, K.-I. Efficient PageRank Tracking in Evolving Networks. KDD. 2015:875-84.

[33] Page, L., Brin, S. The PageRank Citition Ranking: Bringing Order to the Web. Technical Report, Stanford University. 1998.

[34] Parreira, J. X., Castillo, C., Donato, D., Michel, S., Weikum, G. The Juxtaposed approximate PageRank method for robust PageRank approximation in a peer-to-peer web search network. The VLDB Journal. 2008;17:291-313.

[35] Pearson, K. The problem of the Random Walk. Nature. 1905;72(294).

[36] Pretto, L., editor A theoretical analysis of PageRank. Proceedings of the Ninth International Sympo- sium on String Processing and Information Retrieval; 2002; Lisbon, Portugal.

[37] Pretto, L. Link Analysis for ranking webpages: University of Padua; 2002.

[38] Song, B. Webpage Ranking Using Bayes' Rule and Connected Components: University of Missouri-Columbia; 2012.

[39] Su, A.-J., Hu, Y. C., Kuzmanovic, A., Koh, C.-K. How to Improve Your Search Engine Ranking: Myths and Reality. ACM Transactions on the Web. 2014;8(2).

[40] Thorson, K. Modeling the Web and the computation of PageRank: Hollins University; 2004.

[41] Worldwidewebsize.Com - http://www.worldwidewebsize.com/.

[42] Zhang, H., Lofgren, P., Goel, A. Approximate Persionalized PageRank on Dynamic Graphs. ACM SIGKDD international conference on Knowledge discovery and data mining; San Francisco, CA, USA2016.

# VITA

Bo Song was born on April 7, 1987 in Daqing, China. He received a B.E. in Computer Science & Technology from China University of Petroleum-Beijing, Beijing, China (2008), and M.S. and Ph.D. in Computer Science from University of Missouri-Columbia (2012, 2018).

From 2010 to 2018, he was a graduate research assistant in Department of Computer Science at University of Missouri-Columbia. His research interest is focusing on efficient algorithms for PageRank and Personalized PageRank computing on big data. He also joined the research project to develop an efficient image compression codec in 2015. His other research interests include Machining Learning, Deep Learning, Image Processing, Computer Vision and Computer Graphics.