A Dissertation for the Degree of Ph.D. in Engineering

Fast Index-based Random Walks on Dynamic Graphs for Personalized Analysis

February 2025

Graduate School of Science and Technology,
Keio University

Tsuyoshi Yamashita

Abstract

With the exponential growth of a wide variety of digital data in recent years, random walks on graphs have attracted attention as a personalized analysis that flexibly reflects user interests and preferences.

Chapter 1 describes a personalized analysis method using random walks. Random walk is a graph computation whose input is a source node that represents the user's interest and a termination probability that controls the exploration range. Both parameters need to be set effectively. In addition, it is important to utilize an index that maintains a set of pre-performed random walk paths for fast computation.

Chapter 2 organizes the preliminaries of this dissertation. In particular, it introduces Personalized PageRank (PPR), a personalized analysis method based on random walks, and presents the FORA framework, a state-of-the-art PPR computation method. Then, it explains FORA+, an index-based acceleration method.

Chapter 3 describes the three challenges addressed by this dissertation. Chapter 4, Chapter 5, and Chapter 6 present the dissertation's contribution to each of these challenges.

Chapter 4 establishes parameter-setting guidelines for PPR. Although many methods for setting the source node have been discussed, the termination probability has been blindly set to a fixed value. This chapter investigates how to monotonically balance the influence of global importance and source proximity on PPR results with termination probability, which controls the average path length of the random walk. A case study using a movie rating dataset shows that setting termination probability to shorten random walk monotonically increases the rating of nodes that are directly related to the source node. Statistical evaluation also revealed that varying the average path length of the random walk from 1.05 to 100 resulted in a monotonic change in the cosine similarity between PPR and the global importance vector from 0.003 to 0.76 at the maximum.

Chapter 5 proposes a fast random walk path generation method for arbitrary parameters. For fast computation, it is important to reference an index, but the accepted termination probability is limited to the value at the time of index generation. As a result, to generate paths for arbitrary termination probabilities, it was necessary to perform random walks without the index. This dissertation proposes an algorithm, αFlexWalk, to quickly generate random walk paths for arbitrary termination probabilities by manipulating the paths in the index. In particular, αFlexWalk focuses on the fact that the random walk path length varies stochastically as the termination probability changes. The algorithm generates mathematically guaranteed paths by connecting and cutting the indexed paths. Evaluations showed that αFlexWalk was up to 11.2 times faster than existing index-free methods.

Chapter 6 proposes a fast and lightweight index management method for high-accuracy computation on dynamic graphs. When a graph is updated after index generation, it is necessary to re-generate some indexes to guarantee accuracy, but the time and space computation cost is significant. This dissertation proposes an index management method that eliminates index re-generation for graph updates, focusing on the fact that index references in PPR computation concentrate on nodes whose index does not change much after re-generation. The evaluation revealed that even if the number of edges is doubled or halved from the time of index generation, the loss of accuracy is less than 0.3%.

Finally, Chapter 7 concludes this dissertation.

Acknowledgments

First of all, I would like to express my sincere gratitude to Assoc. Prof. Kunitake Kaneko. He has carefully guided me as my advisor for six years and is the chair of this dissertation committee. I would like to express my gratitude to the vice chairs of this dissertation committee: Assoc. Prof. Yoshiaki Oda, Dr. Yuuki Takai, and Prof. Hiroki Matsutani. I would also like to thank Dr. Naoki Matsumoto for collaborating with me on many projects. I would also like to thank Prof. Fumio Teraoka for his helpful advice on my research. Although it is difficult to mention them all by name, I would also like to thank the members of my laboratory. Finally, I would like to express my sincere gratitude to my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

As diverse digital data has grown exponentially, personalized analysis that flexibly reflects users' interests and preferences has become increasingly important. According to a white paper published by the International Data Corporation, the total amount of digital data created, captured, or replicated in the world in 2019 is estimated to be 45ZB (1ZB=10^{21} bytes), and that in 2025 is predicted to be 175ZB [1]. In addition, it is important to comprehensively handle the data for effective utilization of digital data since its format is diverse [1]. Furthermore, personalized analysis for each user is also critical [2–8]. In particular, it is necessary to comprehensively analyze diverse data based on the interests and feelings of each user, which are estimated from their history or input, to determine important digital data personalized for each user. Figure 1.1 shows an example of personalized analysis in

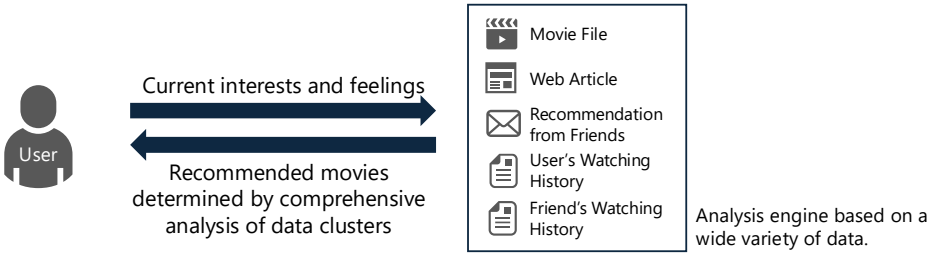


Figure 1.1: An example of personalized movie recommendations based on a wide variety of data.

movie recommendations. In Figure 1.1, the user inputs current interests and feelings into the analysis engine. The analysis engine then comprehensively analyzes the movie content, web articles, recommendations from friends, and the user's or friend's watching history to determine the recommended movie.

In this situation, adopting a general-purpose representation of dynamic and diverse digital data and scalable personalized analysis is important [2–12]. To handle diverse digital data, it is necessary to represent digital data of arbitrary formats and the relationships between them in a general-purpose way [2–6]. Designing an analysis method for each data to be processed is challenging, and applying a single analysis method to arbitrary data types is desirable. In addition, assuming a dynamic situation where data will be added or deleted is required [11–20]. Moreover, for scalable personalized analysis, local exploration starting from the user's interest data is important [9, 10, 21]. The reason is that it is not practical to analyze all the enormous data, and the computational cost is reduced by limiting the scope

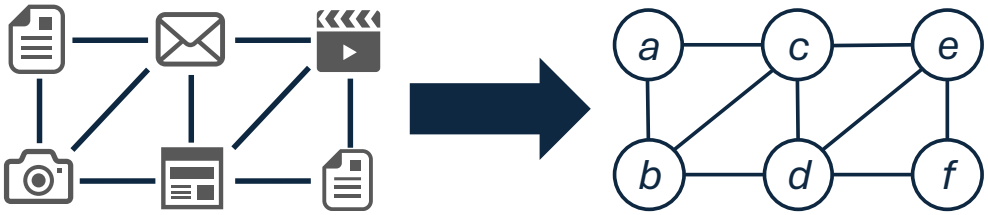


Figure 1.2: A graph created by the diverse digital data.

of analysis while considering the focused data and query.

This dissertation then focuses on graph-based personalized analysis. Because graphs can abstractly represent any digital data as nodes and the relationships between digital data as edges, they help represent data in general-purpose ways. Figure 1.2 shows an example of creating a graph from various digital data. Each node represents digital data, and users or digital data owners create edges by defining the relationship between data. In such a graph, personalized analysis is realized by determining the source node based on the user's interests and current feelings and then analyzing the topology of the source node's neighborhood.

1.2 Random-Walk-Based Personalized Analysis on Dynamic Graphs

As a general-purpose personalized analysis method for graphs, this dissertation focuses on random walks. In this dissertation, a random walk is a

graph computation that takes a source node s and a termination probability α as inputs and outputs a path. Starting from s, it repeatedly transitions to random adjacent nodes and terminates at each visited node with probability α. In personalized analysis, s corresponds to the starting point of the exploration, and α corresponds to the scope of the exploration [2–6]. Therefore, it is important to set these two parameters effectively. Since random walks explore the graph by traversing adjacent nodes, they are suitable for local exploration around the user's interest nodes without considering the whole graph. In addition, since they do not use information other than the graph topology, they are also suitable for general-purpose computations. An example of graph analysis using random walks includes the metric of node importance called Personalized PageRank (PPR), which quantifies the importance of each node by focusing on the distribution of the nodes visited in many random walks performed from the source node [9].

Furthermore, an "index" is effective for fast random walks [10–12, 21]. The index stores the pre-performed random walk paths starting from each node. When performing graph computations using random walks, it is possible to speed up them by referencing the index and omitting some random walks.

However, there are three main problems with personalized analysis using random walks on dynamic graphs [2–6]. Firstly, there is no established guideline for setting the termination probability α in personalized graph analysis queries. In personalized analysis using random walks, although the effec-

tive setting of α is required, it has been set to a fixed value blindly. Therefore, it is necessary to establish guidelines for setting α according to user requirements by clarifying how the results of personalized analysis change depending on α.

Secondly, in personalized analysis, although it is necessary to perform a random walk for arbitrary termination probability α, the problem with the index-based speeding-up method is that it only accepts fixed α [10, 21]. Therefore, using the index is impossible when performing a random walk for arbitrary α. Consequently, achieving flexible and fast analysis is essential for personalized analysis, which this dissertation assumes.

Thirdly, re-indexing, which re-generates some indexed random walk paths, is necessary to guarantee accuracy when the graph is updated after the index generation. However, there are problems with scalability because of the significant time and space costs involved in identifying the parts that require re-indexing and re-generating themselves [11–20]. Therefore, assuming a dynamic graph, re-indexing for each graph update becomes a bottleneck. In addition, since the analysis queries also have to wait until the re-indexing is completed, the scalability of personalized analysis is also limited. Consequently, a fast, lightweight, and accurate index management method is required.

1.3 Contributions of this Dissertation

This dissertation solves the problems described in Section 1.2. The main contributions of this dissertation are summarized as follows.

- Chapter 4 establishes the guidelines for setting termination probability α by experimentally clarifying that α monotonically balances the influences of global importance and source proximity on PPR results.
- Evaluations confirmed that the cosine similarity between PPR and PageRank vectors, which represents global importance, changes monotonically from 0.003 to 0.76 at the maximum by changing α from 0.95 to 0.01.
- Chapter 5 proposes an index-based method, αFlexWalk, to generate random walk paths for arbitrary termination probabilities by stochastically connecting or cutting the indexed random walk paths.
- Evaluations showed that αFlexWalk improves the processing time by at most 11.2 times compared with the index-free method.
- Chapter 6 proposes a method that significantly reduces heavy re-indexing while achieving comparable accuracy to the guaranteed methods. It depends on the observation that the index references are concentrated on the nodes whose index is stable.
- Evaluations also show that the proposed method's accuracy reduction

is less than 0.3% compared with the guaranteed methods until 20% of the edges are updated.

1.4 Structure of this Dissertation

The remainder of this dissertation is structured as follows. Chapter 2 presents the preliminaries of this dissertation. Chapter 3 describes the problems this dissertation handles. Chapter 4 discusses the method for parameter settings of personalized graph analysis. Chapter 5 provides the index-based algorithm to improve the processing speed of personalized graph analysis. Chapter 6 presents the index management scheme considering the dynamic graph. Chapter 7 concludes this dissertation.

Chapter 2

Preliminaries

This chapter introduces the preliminaries of this dissertation. Table 2.1 lists the frequently used notations.

2.1 Graph

$G = (V, E, W)$ is a weighted directed graph with n nodes and m edges. An undirected graph is represented by putting bi-directional directed edges on all edges. W is an $n \times n$ matrix that represents weights of each edge. $W_{v,w}$ is the weight of edge if $(v, w) \in E$. If $(v, w) \notin E$, $W_{v,w} = 0$. If G is undirected, $W_{v,w} = W_{w,v}$ is ensured. In the unweighted graph, weights of all edges $\in E$ are 1. If nodes $v, w \in V$ and $(v, w) \in E$, then w is an adjacent node of v, and $N_{out}(v)$ denotes the set of adjacent nodes of v. The degree of v, denoted as $\deg(v)$, is the sum of the weights of the edges from v, i.e., $\Sigma_{w \in N_{out}(v)} W_{v,w}$. Note that in unweighted graph, $\deg(v) = |N_{out}(v)|$. In the following part, W

Table 2.1: Notations.

Notation	Description
$G(V, E, W)$	A weighted directed graph with node set V, edge set E, and weight matrix W
n, m	The number of nodes and edges, respectively
$N_{out}(v)$	A set of adjacent nodes of v
$\deg(v)$	A degree of v
$G^t = (V^t, E^t, W^t)$	A dynamic graph at the timestamp t
$\pi(s, v)$	PPR value of node v with respect to source node s
$\boldsymbol{\pi}_s$	PPR vector storing all PPR values for all nodes
$\boldsymbol{\pi}$	PageRank vector storing all PageRank values for all nodes
α	A termination probability of a random walk
ω	The number of random walks used to compute PPR
Idx^t	The index at the timestamp t

will be omitted when it is clear whether G is weighted or not is irrelevant to the discussion. Figure 2.1 is a sample unweighted graph frequently used to explain the algorithms in the later part.

This dissertation handles dynamic graphs. Let $G^t = (V^t, E^t, W^t)$ be a dynamic, weighted, and directed graph at timestamp t. It is enough to consider only insertions and deletions of edges as graph updates. Note that a sequence of edge updates represents node insertions and deletions. This dissertation assumes that only one edge update occurs from G^{t-1} to G^t. Given that G^0 is an initial graph, G^t is a graph where t edges are updated. In the following part, the subscripts indicating timestamps will be omitted for simplicity if there is no danger of confusion.

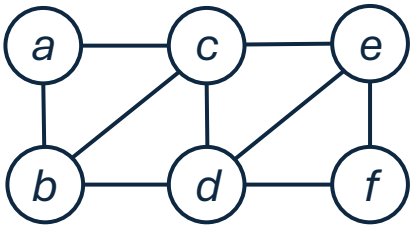


Figure 2.1: A sample unweighted graph ($n = 6, m = 9$).

2.2 Random Walk

A random walk is a graph computation that takes a source node s and a termination probability α as inputs. It starts from s and repeatedly moves to a random adjacent node of the current node until termination. On each current node, it terminates with a probability α and moves to the random adjacent node with a probability $1-\alpha$. Algorithm 1 shows the typical random walk algorithm called Monte Carlo.

Selecting the next node depends on whether the graph is weighted. If the graph is unweighted, the random walk randomly and equally selects the next node from all adjacent nodes. Otherwise, it randomly selects the next node with a probability proportional to the weight of the corresponding edge. Although several methods exist for randomly selecting from non-uniform distributions, this dissertation adopts the Alias method [22], which determines the next node with O(1) time complexity by indexing.

α controls the length of the random walk, and changing α affects the distributions of the visited nodes [23,24]. Therefore, α should be determined

Algorithm 1 Monte Carlo

Input: Source node s, termination probability α
Output: A random walk path *path*

```
path ← [s];
current_node ← s;
while random() > α do
    current_node ← random adjacent node of current_node;
    path.append(current_node);
end while
```

depending on the graph features and applications. The length of the random walk l_α is the number of nodes the random walk visits until termination.

The expected length of the random walk $\mathbb{E}[l_\alpha]$ is $1/\alpha$. The reason is that the probability that the length of the random walk is $k(k \geq 1)$ is expressed by Equation (2.1), and $\mathbb{E}[l_\alpha]$ is derived as shown in Equation (2.2). In addition, the time complexity of the random walk is $O(1/\alpha)$, considering that the time complexity of determining the next node is O(1).

$$P(l_\alpha = k) = \alpha(1-\alpha)^{k-1} \tag{2.1}$$

$$\mathbb{E}[l_\alpha] = \Sigma_{k=1}^{\infty} k \cdot P(l_\alpha = k) = \frac{1}{\alpha} \tag{2.2}$$

2.3 Personalized PageRank (PPR)

Given a graph G, a source node $s \in V$, a target node $v \in V$, and a termination probability α, PPR of node v with respect to s, denoted as $\pi(s, v)$, is the probability that a random walk starting from s terminates at v [9]. When Top-k PPR is queried, the k nodes with the highest PPR values are output.

An n-dimensional vector that stores PPR values of all nodes with respect to s is called the PPR vector and is denoted as $\boldsymbol{\pi}_s$.

PPR is computed by independently performing ω random walks from s and determining the ratio of random walks visiting each node [25–27]. Therefore, if the number of visits to node v in ω random walks is N_v and the sum of the length of ω random walks is N_ω, $\pi(s, v)$ is approximated by N_v/N_ω. Since the expected length of a random walk is $1/\alpha$, the expected N_ω is ω/α. Other PPR computation methods include Power Iteration [28–30], Forward Push [31–33], and Reverse Push [34–36].

In contrast, PageRank is a metric that quantifies the global importance of each node by the probability that a random walk equally starting from all nodes visits each node [9]. An n-dimensional vector that stores PageRank values of all nodes is called the PageRank vector and is denoted as $\boldsymbol{\pi}$.

2.4 State-of-the-art PPR computation method: FORA

FORA is a state-of-the-art method for computing PPR vector $\boldsymbol{\pi}_s$ with respect to the input source node s [10, 21]. When queried, FORA performs Forward Push [37] and Monte Carlo algorithms in two phases.

Algorithm 2 shows the details of FORA. FORA manages two variables for each node v: *estimate* $\hat{\pi}(s, v)$ and *residue* $r(s, v)$. $\hat{\pi}(s, v)$ represents the

number of random walks from s visiting v, and $r(s, v)$ represents the number of random walks from s remaining at v without termination. In the initial state, $\hat{\pi}(s, s)$ and $r(s, s)$ are ω, and $\hat{\pi}(s, v)$ and $r(s, v)$ where $v \neq s$ are zero because all random walks remain at s without terminating. In addition, *total_step* for normalization is initialized as ω (lines 1–3).

In the Forward Push phase, a push operation is performed on v as long as a node v whose *residue* is more than $\frac{|N_{out}(v)|}{\alpha}$ exists (lines 5–12). The push operation on node v moves $1 - \alpha \cdot r(s, v)$ random walks to adjacent nodes proportionally to the weight of the adjacent edge and terminates the remaining $\alpha \cdot r(s, v)$ random walks (lines 6–11).

In the Monte Carlo phase, all remaining random walks maintained by *residue* terminate by performing $\lceil r(s, v) \rceil$ random walks from each node v (lines 14–22). Here, $monte\_calro(v, \alpha)$ returns a random walk path by Algorithm 1. Note that when counting the visited times, normalization is required because the performed number of random walks is $\lceil r(s, v) \rceil$, not $r(s, v)$ (line 18).

The performance guarantee and algorithm correctness of FORA come from the fact that the PPR value is decomposed by Equation (2.3) [37].

$$\pi(s, v) = \hat{\pi}(s, v) + \sum_{w \in V} r(s, w) \cdot \pi(w, v) \tag{2.3}$$

Equation (2.3) means that $\pi(s, v)$ is computed by the Forward Push and Monte Carlo results. $\hat{\pi}(s, v)$ and $r(s, w)$ come from *estimate* and *residue*, respectively, and $\pi(w, v)$ comes from performed random walks in Monte Carlo

phase. The approximation error of FORA is caused in the Monte Carlo phase because the distribution of the visiting nodes cannot be computed exactly. Setting a larger ω will make the result more accurate [25].

2.5 Index-based FORA: FORA+

FORA+ accelerates FORA by referencing an index during the Monte Carlo phase [10, 21]. The index is generated before PPR queries occur and maintains the list of pre-performed random walk paths with each source node as a key.

In the index generation, $\lceil |N_{out}(v)|/\alpha \rceil$ random walks are performed from each node v. The pre-performed random walk paths are stored as $Idx[v]$. The space complexity is $\Theta(n + m/\alpha)$ because the number of keys of the index is n, and the number of random walks performed from node v is $\Theta(|N_{out}(v)|/\alpha)$.

In FORA, performing random walks from nodes with *residue* is required in the Monte Carlo phase. FORA+ omits on-the-fly random walks by referencing Idx. The push condition in the Forward Push phase guarantees that *residue* of each node v is less than $|N_{out}(v)|/\alpha$. Therefore, the required number of random walks from node v in the Monte Carlo phase is also guaranteed to be less than $\lceil |N_{out}(v)|/\alpha \rceil$. Consequently, FORA+ can completely omit performing on-the-fly random walks in the Monte Carlo phase.

Table 2.2: Datasets (M=10^6, B=10^9).

Name	Directed	Weighted	n	m	Dynamic
DBLP [38]	No	No	0.3M	1.1M	No
Web-BerkStan [38]	Yes	No	0.7M	7.6M	No
YouTube [39]	No	No	3.2M	9.4M	Yes
Flickr [40]	Yes	No	2.3M	33M	Yes
MovieLens [41]	No	Yes	418K	34M	Yes
LiveJournal [38]	No	No	4.0M	35M	No
Orkut [38]	No	No	3.1M	117M	No
Twitter [42]	Yes	No	41.7M	1.5B	No
Friendster [38]	No	No	65.6M	1.8B	No

2.6 Evaluation Environment

This section explains the evaluation environment used in the following part of this dissertation. All implementation is written in C++, and all evaluations are performed on a machine with Ubuntu 20.04.4 / Xeon Gold 6334 CPU @3.60GHz / 1TB DDR4-3200 memory. Table 2.2 lists the real-world graph datasets.

Algorithm 2 FORA

Input: A source node s, a termination probability α, the number of random walks ω

Output: $\boldsymbol{\pi}_s$

$\hat{\pi}(s,s) \leftarrow \omega$; $r(s,s) \leftarrow \omega$;
$\hat{\pi}(s,v) \leftarrow 0$; $r(s,v) \leftarrow 0$; for all $v \in V \setminus \{s\}$;
$total\_step \leftarrow \omega$;

while $\exists v \in V$ *such that* $r(s,v) > \frac{|N_{out}(v)|}{\alpha}$ **do**
  for $w \in N_{out}(v)$ **do**
    $walks\_to\_move \leftarrow (1-\alpha) \cdot r(s,v) \cdot \frac{W_{v,w}}{\sum_{u \in N_{out}(v)} W_{v,u}}$
    $r(s,w) \leftarrow r(s,w) + walks\_to\_move$;
    $total\_step \leftarrow total\_step + walks\_to\_move$;
  end for
  $\hat{\pi}(s,v) \leftarrow \hat{\pi}(s,v) + \alpha \cdot r(s,v)$; $r(s,v) \leftarrow 0$;
end while

for $v \in V$ **where** $r(s,v) > 0$ **do**
  $\omega_v \leftarrow \lceil r(s,v) \rceil$;
  for $i = 1$ **to** ω_v **do**
    $path \leftarrow monte\_carlo(v, \alpha)$;
    $\hat{\pi}(s,w) \leftarrow \hat{\pi}(s,w) + \frac{r(s,v)}{\omega_v}$;
    $total\_step \leftarrow total\_step + \frac{r(s,v)}{\omega_v}$;
  end for
  $r(s,v) \leftarrow 0$;
end for

$\pi(s,v) \leftarrow \frac{\hat{\pi}(s,v)}{total\_step}$; for all $v \in V$;

Chapter 3

Overview

As described in Section 1.2, the remainder of this dissertation will address the following problems: the guidelines for setting the termination probability α in personalized analysis using random walks, the generation of random walk paths for arbitrary α using the index, and the index management on dynamic graphs. This chapter provides an overview of each work and the relationships among these three works.

Firstly, regarding the guidelines for setting the termination probability α, it is important to set the source node s effectively and the termination probability α in the personalized analysis using random walks. While many methods for selecting s based on the user's activity history and input have been discussed, α has been set to a fixed value blindly. This dissertation focuses on the fact that the average path length of the random walk changes depending on α and quantitatively clarifies the influence of α on Personalized PageRank (PPR) result. In particular, while nodes with high PPR values

contain a mixture of nodes with high global importance and source proximity, this dissertation shows that α is a parameter that monotonically balances the influence that both of these factors have on the results of PPR. A case study using a movie rating dataset found that the shorter the average path length of the random walk, the higher the PPR value of the nodes directly related to the source node. In addition, it was revealed that by changing the average path length of random walks from 1.05 to 100, the cosine similarity between the PPR vector and the PageRank vector, which represents the global importance, monotonically changes from 0.001 to 0.78.

Secondly, regarding the index-based random walk path generation for arbitrary α, the index is a set of random walk paths, and α of the path obtained by simply referencing the index is restricted to a specific value, α_{index}. However, in personalized analysis, it is crucial to generate paths for arbitrary α, and the fast computation using the index inhibits analysis that is flexible with respect to α. Therefore, this dissertation proposes a method for generating random walk paths for arbitrary α using an index by manipulating indexed random walk paths. In particular, the proposed method focuses on the fact that the path length of a random walk changes stochastically when the termination probability changes from α_{index}, and a proposed method connects or cuts the indexed random walk paths. The evaluation compared the processing time of the proposed index-based methods with the existing index-free methods known to apply to arbitrary α. The results showed that the proposed methods are 11.2 times faster than the existing methods.

Thirdly, regarding the index management for dynamic graphs, it is necessary to re-generate some indexed paths to guarantee accuracy for each graph update after the index generation. However, existing methods are not scalable, as the time and space costs for identifying the paths that need to be re-generated and re-generating cost itself are significant. This dissertation proposes a method for eliminating index re-generation during graph updates while clarifying that index references in PPR computations are concentrated on nodes whose indexed paths require little re-generation during graph updates. Although eliminating index re-generation does not guarantee accuracy, evaluations have revealed that the accuracy of the proposed method is comparable to that of the guaranteed methods.

Figure 3.1 shows the relationships between the three problems addressed in this dissertation. In Figure 3.1, User A determines the value of α while referencing the guidelines for setting it (1, 2 in the figure). Chapter 4 establishes these guidelines. The PPR query, including the determined α, is then submitted to the existing FORA+ algorithm (3 in the figure). FORA+ generates random walk queries starting from each node in the Monte Carlo phase and submits them to αFlexWalk (4 in the figure). αFlexWalk generates random walk paths with a termination probability α while referencing the index (5, 6 in the figure). Chapter 5 proposes the path generation algorithm. FORA+ then completes PPR computation with the generated paths and returns the results to User A (7, 8, 9 in the figure). User B also submits an update query to the graph (a. in the figure). The graph notifies the index

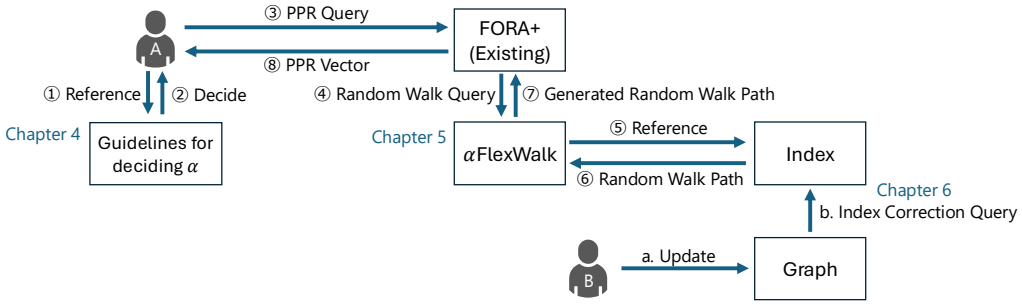


Figure 3.1: The relationships between the three problems addressed in this dissertation.

of the updated edge, and the index corrects the stored paths accordingly (b. in the figure). Chapter 6 proposes the lightweight correction method.

Chapter 4

Balancing Global Importance and Source Proximity in Personalized Graph Analysis

4.1 Overview

As discussed in Chapter 3, existing recommendations based on Personalized PageRank (PPR) blindly set the termination probability α of random walks to a fixed value. It is necessary to establish guidelines for setting α according to user requirements by clarifying how the results of personalized analysis change depending on α.

It is known that PPR determines nodes with high global importance and source proximity, and it is necessary to monotonically balance both influences on the PPR vector according to the user's requirements in PPR-based recommendations. When the influence of global importance is dominant,

nodes that are important from the perspective of the whole graph tend to have high PPR values, regardless of the source node. While it determines nodes considered important for many users, the drawback is that the source node has little impact on the PPR vector, and the results tend to be commonplace. In contrast, when the influence of source proximity is dominant, important nodes with respect to the source node tend to have high PPR values even if their global importance is not necessarily high. While it determines nodes strongly affected by the source node, the drawback is that it is possible to make the PPR vector minor from a global perspective. Therefore, a trade-off between global importance and source proximity exists. In real-world applications, users need to adjust the balance between them depending on the source node or their preferences. Users are also required to observe PPR vectors while gradually changing the parameters. In such a case, it is important to balance their influences monotonically according to the parameters.

However, existing work on PPR-based recommendations has discussed the way of setting the source node [43, 44], and has not sufficiently discussed the characterization of PPR vectors for a fixed source node. Although some methods utilize external information such as node labels to improve the recommendation quality [43, 45, 46], these methods are not general-purpose because they require determining the necessary external information and collecting it for each graph. In addition, various recommendation criteria including diversity [7, 45–47], novelty [43, 47], serendipity [44, 47], and fairness [7, 8]

have been proposed. To the best of the author's knowledge, no work focuses on balancing the influences between global importance and source proximity.

This chapter discusses how to monotonically balance the influences of global importance and source proximity on the PPR vector. In particular, the termination probability α, a parameter of PPR that controls the random walk length, changes the nodes that random walks are likely to visit. As the random walks get longer, they will likely visit nodes with high PageRank values, where PageRank indicates global importance. On the other hand, as the random walks get shorter, they will likely visit nodes close to the source node, regardless of their global importance.

This chapter observes the PPR vector while changing α using the real-world datasets in Table 2.2. A case study using a movie rating dataset confirmed that movies considered directly relevant to the source node got a higher PPR value by shortening the random walks. Furthermore, statistical evaluation confirmed that the similarity between PPR and PageRank vectors weakens by shortening the random walks. In particular, changing the expected random walk length from 1.05 to 100 resulted in a monotonic change in the cosine similarity of PPR and PageRank vectors from 0.003 to 0.76 at the maximum.

The main contributions of this chapter are summarized as follows.

- This chapter proposes a method to monotonically balance the influences of global importance and source proximity on the PPR vector by

the termination probability α that controls the expected random walk length.

- A case study using a movie rating dataset confirmed that α monotonically changes the ranking of movies considered to be directly related to the source movie.
- Evaluations using real-world datasets found that the similarity between the PPR vector and the PageRank vector, which represents global importance, changes monotonically with α.

The remainder of this chapter is structured as follows. Section 4.2 discusses related work. Section 4.3 presents the approach. Section 4.4 provides the case study using the movie rating dataset. Section 4.5 shows the statistical evaluations. Section 4.6 concludes this chapter.

4.2 Related Work

Personalized recommendations with respect to the source node are realized by quantifying how much each node is worthy of recommendation. This section discusses the existing quantification methods and the proposed method from three perspectives: the recommendation criteria, the information used for recommendations, and the controllability of the recommendation results.

Firstly, regarding the recommendation criteria, while typical recommendations focus only on source proximity to the source node [48], various

criteria, including diversity [7, 45–47], novelty [43, 47], serendipity [44, 47], and fairness [7, 8], have been focused. Each recommendation criteria focuses on different aspects besides a certain level of source proximity. In particular, diversity emphasizes that the similarity between nodes in the recommendation list is slight. Novelty emphasizes that the user has not checked the recommended nodes before. Serendipity emphasizes that the recommended nodes belong to a field the user has not checked before. Fairness emphasizes that every node will have the opportunity to be recommended. On the other hand, this dissertation focuses on the balance between global importance and source proximity.

Secondly, regarding the information used for recommendations, personalized recommendation methods are categorized into two types: those using external information such as node labels [43, 45, 46] and those using only graph topology [44, 48, 49]. For example, movie recommendations using external information recommend movies with the same director, cast, or genre as the user's favorite [43, 45, 46]. However, these methods are not general-purpose because they rely on external information. As a result, they must decide the necessary external information and collect it for each graph. On the other hand, the methods using only graph topology utilize only the adjacent information between nodes. For example, these methods exploit the PPR vector with respect to the interest node [44, 48] and the similarity between the embedding vectors [49]. These methods are general-purpose because the same analysis method can be applied to any graph.

Furthermore, comparing the methods focusing on the PPR vector and embedding vectors, the former requires computation only on the source node. In contrast, the latter requires computation on all nodes for the similarity computation, resulting in a lack of scalability. From the above, this dissertation focuses on the general-purpose and lightweight method of exploiting the PPR vectors.

Thirdly, regarding the controllability of the recommendation results, the existing methods are categorized into two types: those that can control how much the recommendation criteria are incorporated into the results by parameters [43, 45, 46] and those that cannot [44]. Furthermore, the controllable methods are divided into two types: those that can control monotonically [45] and those that cannot [43, 46]. Controllability is an important element for flexible recommendation according to the users' preferences and interest nodes. Moreover, the monotonic control is also an important element in finding the desired parameter settings with a small amount of computation, e.g., by binary search. Therefore, this dissertation aims to monotonically control the influence of global importance on the recommendation results by the length of the random walks performed in PPR computations. Here, a random walk has two parameters: the source node s and the termination probability α that controls the random walk length. However, existing work has mainly discussed the effect of s on the recommendation results [43, 44]. An existing PPR-based work, where novelty is the recommendation criteria and external information is used, reports that α affects the novelty of the recommenda-

tion results [43]. However, the influence is non-monotonic, highly dependent on the dataset, and small range. Therefore, the influence of α needs more discussion.

4.3 Approach

This section describes an approach for balancing the influences of global importance and source proximity on the PPR vector. First, it describes how the visited nodes of the random walks change by α. Second, it presents the hypothesis that this can be used to balance the influences of global importance and source proximity on the PPR vector. Note that the expected random walk length is $1/\alpha$, as described in Section 2.2.

If α changes, the visited nodes of the random walks are expected to change in terms of how close they are to the source node or how high their PageRank values are. In particular, it is expected that a smaller α and longer random walks decrease the influence of the source node on the visited nodes. As a result, nodes with high global importance, i.e., nodes with high PageRank values, are more likely to be visited even if they are distant from the source node. In other words, as α approaches 0, the visited nodes are no longer affected by the source node, and the PPR vector approaches the PageRank vector, which is independent of the source node. For example, when the graph is undirected and connected, $\lim_{\alpha \to 0} \boldsymbol{\pi}_s$ is known to converge to a vector proportional to the stationary distribution [50]. Here, some work

reports that the PageRank value of each node strongly correlates with its degree [50, 51].

On the other hand, when the random walks are shortened by increasing α, the source node has a more significant influence on the visited node. As a result, it is expected that nodes with high source proximity are more likely to be visited regardless of their global importance. This expectation means that as α approaches 1, the visited nodes of the random walks are no longer affected by PageRank value, and the PPR vector will approach the one-hot vector where only the element corresponding to the source node is 1.

Although the behavior of the PPR vector when α is close to 0 and 1 is obvious, the behavior when α is between 0 and 1 is unpredictable. Therefore, it is necessary to observe the monotonicity or speed of changes experimentally. The following sections of this chapter will present the case study and statistical evaluation using real-world datasets.

4.4 Case Study

This section uses the MovieLens dataset [41] and observes the top 20 nodes of PPR while changing the termination probability α. MovieLens dataset contains a total of 34 million ratings from 330,000 users for about 87,000 movies. At each rating, a user u gives a score s of [1.0, 5.0] to a movie m. These ratings construct a graph whose nodes are users and movies by putting

an undirected edge between u and m weighted by s [43–45]. The personalized recommendation is realized by computing the PPR vector whose source node is the user's interest movie. Although the MovieLens dataset includes metadata such as the genre of each movie, information other than graph topology is not used because this dissertation focuses on general-purpose personalized recommendation without depending on external information.

Table 4.1 shows the top 20 PPR movies for $\alpha = 0.2, 0.4, 0.6, 0.8$, and "Avengers: Infinity War Part I" is set to the source node. Note that the movies shown in bold font are produced by Marvel Studios. Table 4.1a – 4.1d confirms that Marvel movies, which are directly related to the source node, monotonically rise to the upper rankings as α increases. In particular, when $\alpha = 0.6, 0.8$, seven Marvel movies are ranked in the top 20, indicating that source proximity had a significant influence on the results. Specifically, the Marvel movies ranked in the top 20 are nodes with PageRank values between 91st and 706th place. This result indicates that as α increases, nodes that do not necessarily have high global importance but have high source proximity are more likely to be visited by random walks.

On the other hand, when $\alpha = 0.2$, only three Marvel movies, including the source node, are ranked in the top 20, and non-Marvel movies are nodes with high PageRank values. In particular, all non-Marvel movies ranked in the top 20 are nodes with PageRank values in the top 46. This result shows that when α is small, nodes with higher global importance are more likely to be visited by random walks.

A similar behavior was also observed when taking other movies as the source node. For example, when "Monsters, Inc." was taken as the source node, I found that Disney and Pixar movies monotonically rose to the higher ranks as α increased. In another case, when "Shoplifters" was taken as the source node, I confirmed that "Parasite" related in terms of Asian movies that won the Palme d'Or at the Cannes Film Festival rose to the top of the list by increasing α.

4.5 Statistical Evaluation

This section quantitatively evaluates the relationships between α and the characteristics of PPR vectors using nine real-world datasets listed in Table 2.2. Section 4.5.1 measures the relationships between α and the influence of the PageRank vector on the PPR vector, where PageRank represents the global importance. The results show that an increase in α causes a monotonic decrease in the influence of the PageRank vector on the PPR vectors, indicating that α can monotonically balance the influences of global importance and source proximity. In addition, Section 4.5.2 analyzes how PPR values of the high-ranking nodes change depending on α. As a result, the decrease in α causes the PPR values of the top-ranking nodes to be concentrated at large and close values, indicating that random walks tend to visit many nodes in a relatively uniform distribution.

Table 4.1: Top 20 PPR nodes when "Avengers: Infinity War Part I" is the source node.

(a) $\alpha = 0.2$

Rank	Title
1	**Avengers: Infinity War I**
2	The Shawshank Redemption
3	The Matrix
4	The Dark Knight
5	Inception
6	Star Wars: Episode IV
7	The Lord of the Rings III
8	The Lord of the Rings I
9	The Lord of the Rings II
10	Star Wars: Episode V
11	Fight Club
12	Forrest Gump
13	Raiders of the Lost Ark
14	**Thor: Ragnarok**
15	Star Wars: Episode VI
16	Pulp Fiction
17	Interstellar
18	**Avengers: Infinity War II**
19	Silence of the Lambs
20	Schindler's List

(b) $\alpha = 0.4$

Rank	Title
1	**Avengers: Infinity War I**
2	The Matrix
3	The Shawshank Redemption
4	The Dark Knight
5	Inception
6	Star Wars: Episode IV
7	The Lord of the Rings III
8	**Thor: Ragnarok**
9	Star Wars: Episode V
10	The Lord of the Rings II
11	The Lord of the Rings I
12	**Avengers: Infinity War II**
13	Fight Club
14	Raiders of the Lost Ark
15	Interstellar
16	Star Wars: Episode VI
17	**Guardians of the Galaxy**
18	Forrest Gump
19	**Iron Man**
20	**Logan**

(c) $\alpha = 0.6$

Rank	Title
1	**Avengers: Infinity War I**
2	The Matrix
3	The Shawshank Redemption
4	The Dark Knight
5	Inception
6	**Thor: Ragnarok**
7	Star Wars: Episode IV
8	The Lord of the Rings III
9	Star Wars: Episode V
10	**Avengers: Infinity War II**
11	The Lord of the Rings II
12	The Lord of the Rings I
13	Fight Club
14	Interstellar
15	**Guardians of the Galaxy**
16	Raiders of the Lost Ark
17	Star Wars: Episode VI
18	**Iron Man**
19	**Logan**
20	**Deadpool 2**

(d) $\alpha = 0.8$

Rank	Title
1	**Avengers: Infinity War I**
2	The Matrix
3	The Shawshank Redemption
4	The Dark Knight
5	**Thor: Ragnarok**
6	Inception
7	Star Wars: Episode IV
8	The Lord of the Rings III
9	**Avengers: Infinity War II**
10	Star Wars: Episode V
11	The Lord of the Rings II
12	The Lord of the Rings I
13	**Guardians of the Galaxy**
14	Interstellar
15	Raiders of the Lost Ark
16	Fight Club
17	Star Wars: Episode VI
18	**Iron Man**
19	**Logan**
20	**Deadpool 2**

4.5.1 α vs. the influences of global importance on PPR vectors

This section evaluates the influence of global importance on the PPR vector by measuring the similarity between the PPR and PageRank vector while changing α. In particular, PPR vectors for randomly selected 100 nodes are computed while changing α in 20 ways: [0.01, 0.05, 0.1, 0.15, 0.2, ..., 0.85, 0.9, 0.95]. Then, the similarity between the computed PPR vectors and the PageRank vector is measured. Here, α used to compute the PageRank vector is fixed at 0.2. The reason is that when comparing PPR vectors for different α with PageRank vector as a baseline, the baseline vector needs to be fixed. The setting of $\alpha = 0.2$ in PageRank computation follows many existing works [2–4, 9, 52].

The similarity is quantified by Normalized Discounted Cumulative Gain (NDCG) focusing on the high-ranking nodes [53] and cosine similarity focusing on the whole vectors. NDCG quantifies the similarity of the top-k nodes of both vectors as a [0, 1] value, where $k = 128$ in this chapter; the more nodes with high PageRank values are included in the top nodes of the PPR vector, the more significant NDCG value is. Equation (4.1) shows the definition of NDCG, where t_i and t'_i are the nodes with the i th highest PageRank and PPR values, respectively.

$$ndcg(\boldsymbol{\pi}_s, \boldsymbol{\pi}, k) = \frac{\sum_{i=1}^{k} \frac{2^{\pi(t'_i)}-1}{\log_2(i+1)}}{\sum_{i=1}^{k} \frac{2^{\pi(t_i)}-1}{\log_2(i+1)}} \tag{4.1}$$

Cosine similarity quantifies the correlation of the values in both vectors as

a [0, 1] value; the more nodes with high PageRank values have high PPR values, the larger the cosine similarity is.

Figure 4.1 shows the similarity between PageRank and PPR vectors for each α. Figure 4.1a and Figure 4.1b are the results with the similarity metrics as NDCG and cosine similarity, respectively. Note that in both figures, the y-axis represents the average of the 100 similarity values, and the scale of the y-axis is linear in Figure 4.1a and logarithmic in Figure 4.1b.

Figure 4.1a indicates that in NDCG, a similarity metric focusing only on the top nodes, an increase in α leads to a monotonic decrease in the similarity between PageRank and PPR vectors, increasing the source proximity in PPR vectors. In particular, by setting $\alpha = 0.01$, NDCG increases to [0.1, 0.94] for each dataset. Here, when NDCG is 0.94, the top nodes of PPR and PageRank vectors are almost identical, differing only in that their rankings are swapped. On the other hand, by setting $\alpha = 0.95$, NDCG is decreased to [0.06, 0.42] for each dataset. Here, when NDCG is 0.06, the top nodes in PPR and PageRank vectors hardly overlap. Moreover, some datasets, such as Web-BerkStan and Friendster, show a monotonous decrease in NDCG over α, but the decrease ranges are [0.20, 0.28] and [0.06, 0.1], and the decrease rates are about 29 and 40%, respectively. Note that, as described later, significant decrease rates are observed in Web-BerkStan and Friendster when the similarity metric is the cosine similarity.

Figure 4.1b also confirms a similar trend when the similarity metric is the cosine similarity. In particular, by setting $\alpha = 0.01$, the cosine similarity

increases to [0.03, 0.76] for each dataset, and by setting $\alpha = 0.95$, it decreases to [0.0001, 0.003]. In Web-BerkStan and Friendster, where the decrease rates are small in the case of NDCG, the decrease ranges of the cosine similarity are [0.003, 0.23], [0.0001, 0.03], and the decrease rates are both 99%. Moreover, the cosine similarity is strongly affected by the dimension of the vectors (in this case, the number of nodes). Thus, results across datasets should be compared carefully. For example, Figure 4.1b shows that datasets with a more significant number of nodes tend to have lower cosine similarity. On the other hand, it is worth noting that the shape of the decrease among datasets is similar.

4.5.2 α vs. the distribution of PPR values of high-ranking nodes

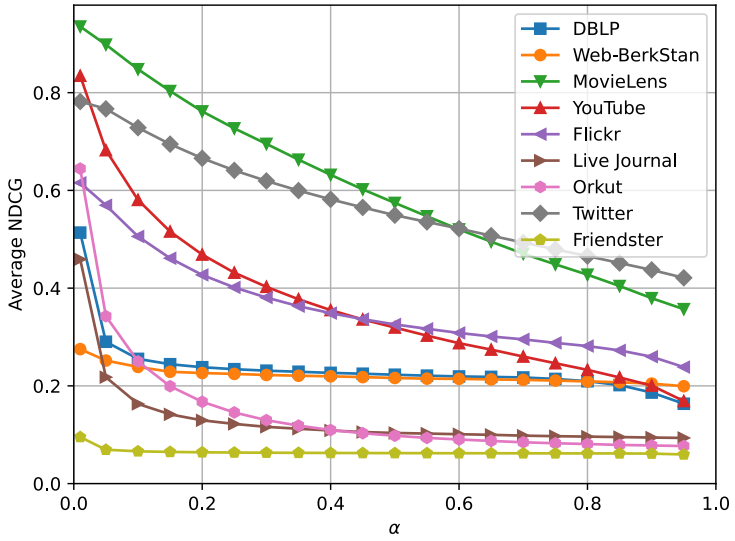
This section evaluates the distribution of PPR values of top-ranking nodes while changing α. In particular, PPR vectors for 100 randomly selected source nodes are computed while changing α to [0.01, 0.05, 0.1, 0.15, 0.2, ..., 0.85, 0.9, 0.95] and measure the average PPR value for each ranking from 1st to 128th. Here, based on the report that the average PPR value at each ranking is distributed to the power [54], the average PPR value y for the ranking x is fitted to the distribution of $y = ax^b$, and the scaling exponent b ($b < 0$) is observed. The reason for focusing on the scaling exponent b is that b quantifies the degree of concentration of PPR values in the high-ranking nodes. The larger b means that PPR values are concentrated in

close values, whereas the more minor b means that PPR values broadly differ among rankings.

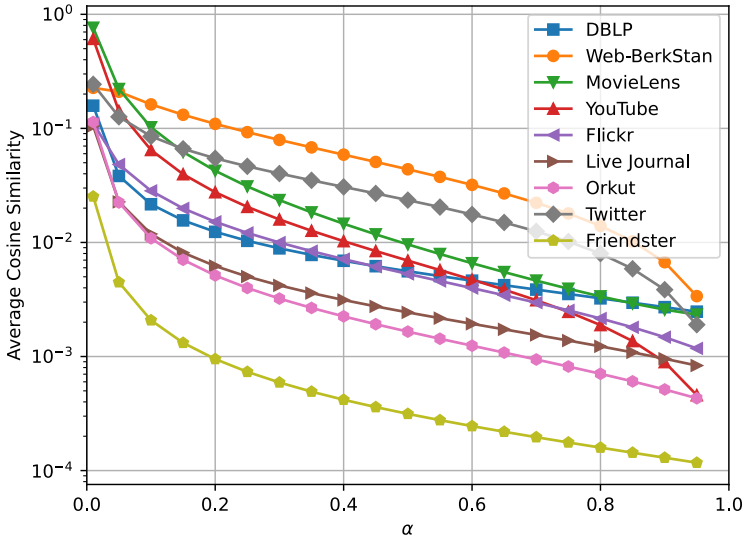
Figure 4.2 shows the relationships between α and the scaling exponents. Figure 4.2 indicates that the scaling exponent decreases monotonically against α. Therefore, by decreasing α, PPR values of top nodes are likely to be concentrated, indicating that random walks visit a large number of nodes in a relatively uniform distribution. The ratio of scaling exponents for $\alpha = 0.01, 0.95$ was [1.2, 2.6] for each dataset. Note that the coefficient of determination when calculating the scaling exponent was always greater than 0.9.

4.6 Conclusion of This Chapter

It is important to monotonically balance the influences of global importance and source proximity on PPR-based recommendations. This chapter proposes a method to balance them by the termination probability α, which is a parameter to control the random walk length. In particular, a case study using the MovieLens dataset showed that when α is increased, nodes with high source proximity rose to higher rankings. Moreover, statistical evaluation on nine real-world datasets listed in Table 2.2 revealed that changing α from 0.01 to 0.95 resulted in a monotonic decrease in the cosine similarity between PPR and PageRank vectors from 0.76 to 0.003 at the maximum.



(a) Similarity metric: NDCG.



(b) Similarity metric: Cosine similarity.

Figure 4.1: α vs. the similarity between PageRank and PPR vectors.

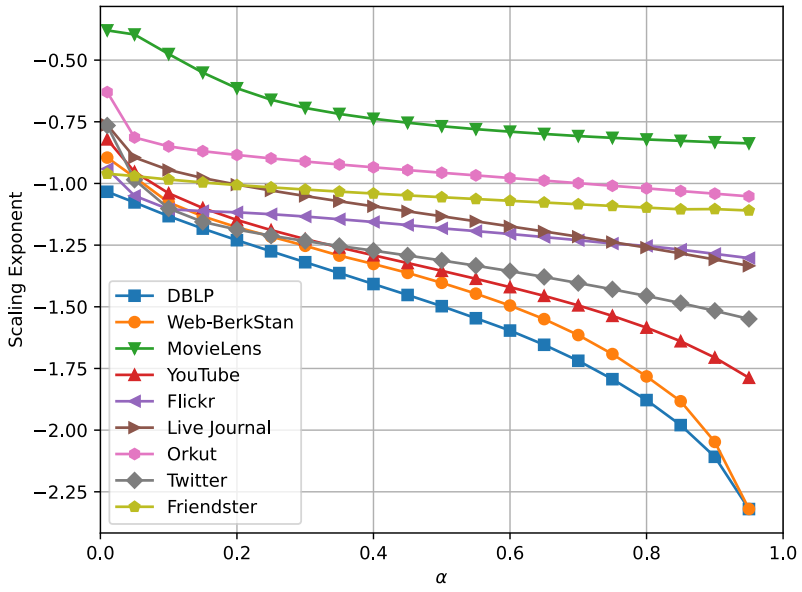


Figure 4.2: α vs. scaling exponent in the distribution of PPR values.

Chapter 5

Index-based Random Walks for Arbitrary Termination Probabilities

5.1 Overview

Index-based random walk path generation needs to be flexible for an arbitrary termination probability α. Here, the flexible path generation requires that α is easily changed with light overhead while using the index. The capability of fast and flexible random walk path generation allows a wide variety of graph analyses. Chapter 4 claims that α should be changed from 0.01 to 0.95 depending on the user's preferences. Additionally, α is set to 0.15 or 0.2 in Personalized PageRank (PPR) [9], 0.1 – 0.5 in graph neural network [55–60], and 0.02 – 0.7 in embedding [49, 61–63]. To be more specific, the Flickr dataset shows the best performance with $\alpha = 0.3$ for embedding [62], whereas α is set to 0.15 in PPR-based recommendations [30]. Moreover, existing work

has clarified the meaning of modifying α in an application [23, 24, 64, 65]. In particular, it is suggested to modify α based on the query source nodes [24,64] and the querying user [23]. Comparing random walk paths for different α realizes spam detection [65]. From the above, quickly generating random walk paths for the arbitrary α is essential for the flexible graph analysis.

Theoretically, there are two independent types of techniques for speeding up flexible random walk path generation: computational and algorithmic techniques. Note that the computational and algorithmic techniques can be coupled because they are for different processing layers. As computational techniques, many methods to generate random walk paths for any α [66–70] have been proposed. Each method removes the bottleneck of the specific environments, such as in-memory [69,70], disk-based [67,68], and distributed [66] settings. However, these methods perform random walks on the fly, and they do not contribute to decreasing computational complexity. In contrast, the existing algorithmic techniques [10–12, 21, 71] are not flexible because these methods only accept restricted α due to the application constraint. The existing algorithmic techniques exploit an index to omit the on-the-fly random walks. However, the acceptable α is constrained by α_{index} in these methods, where α_{index} is the termination probability used for the index generation.

This dissertation aims to develop an algorithmic technique for generating random walk paths for arbitrary α. This chapter proposes an index-based method called αFlexWalk to achieve the goal. Its index is the array of random walk paths for the termination probability α_{index} starting from

each node. In particular, αFlexWalk probabilistically manipulates the indexed paths based on the magnitude relationship between α and α_{index}. It connects and cuts the indexed paths when $\alpha < \alpha_{index}$ and $\alpha > \alpha_{index}$, respectively. As the manipulation is based on mathematical considerations, αFlexWalk generates guaranteed paths.

Evaluation measures the processing time of αFlexWalk while incorporating it into the Monte Carlo phase of FORA+ using nine real-world datasets. As existing methods cannot use the index for accepting arbitrary α, FORA+ using αFlexWalk is compared with index-free FORA. The results show that αFlexWalk improves the processing time of the index-free random walk method by up to 11.2 times.

The main contributions of this chapter are summarized as follows.

- This chapter developed an index-based algorithm αFlexWalk to generate random walk paths for arbitrary termination probabilities.
- As αFlexWalk is performed based on mathematical considerations, generated paths are guaranteed.
- Evaluations using nine real-world datasets showed that αFlexWalk improves the processing time of the index-free Monte Carlo by at most 11.2 times.

Finally, the remainder of this chapter is structured as follows. Section 5.2 discusses related works. Section 5.3 presents the approach. Sec-

tion 5.4 proposes a method called αFlexWalk. Section 5.5 shows the experimental results. Section 5.6 concludes this chapter.

5.2 Related Work

This section describes the existing methods to speed up random walk path generation and compares them with the proposed method. The existing methods are categorized into computational [66–70] and algorithmic methods [10–12, 21, 71]. The computational ones are useful for flexible path generation. However, they do not contribute to reducing time complexity. On the other hand, algorithmic ones speed up the random walk path generation. However, they cannot generate paths for arbitrary α due to the application constraint. I also show the characteristics of the proposed flexible algorithmic method compared with these methods.

The computational methods [66–70] speed up the typical random walk method Monte Carlo described in Section 2.2. Each method assumes a different execution environment and resolves bottlenecks specific to each environment. For example, in the in-memory settings, memory-access-latency-aware techniques have been proposed [69, 70]. In particular, ThunderRW [69] proposes a step interleaving using the software prefetch to hide memory access latency. Flashmob [70] proposes the cache-efficient graph management strategy by focusing on the fact that random walks concentrate on high-degree nodes. For the distributed settings, KnightKing [66] proposes the synchro-

nization technique to reduce server communication. For cases where I/O processing between the disk and the memory is the bottleneck, GraphWalker [67] and NosWalker [68] propose block-centric and scheduling techniques, respectively. As these methods generate random walk paths on the fly, they do not decrease the time complexity compared with Monte Carlo. Developing the algorithmic technique is necessary for improving more.

Although the existing algorithmic methods [10–12, 21, 71] quickly generate random walk paths while referencing the index, these methods only accept the restricted α. The acceleration method is described in Section 2.5. These methods use the indexed paths as they are. Therefore, the acceptable α is limited to α_{index}, where α_{index} is the termination probability used for the index generation.

The proposed method αFlexWalk is categorized as an algorithmic method. By manipulating the indexed paths, it generates guaranteed random walk paths for arbitrary α. As a result, αFlexWalk quickly generates a random walk path using the index and reduces the time complexity at the same time.

Finally, note that αFlexWalk is also effective on dynamic graphs. Several methods [11, 12, 71] propose efficient index management strategies while considering the graph updates. The strategies work for αFlexWalk, so it is robust to dynamic situations. The detailed discussion will be presented in Chapter 6.

5.3 Approach

This section explains the approach to generate the guaranteed random walk paths for the termination probability α using the indexed paths for the termination probability α_{index}. The following part will consider the generation of a random walk path starting at node s, denoted as $path$, from an indexed random walk path starting at node v ($v \in V$), denoted as $path_v$. Here, the termination probabilities of $path$ and $path_v$ are α and α_{index}, respectively. The visiting nodes of $path_v$ are represented as $[v, v_1, v_2, ..., v_l]$.

First, when generating $path_v$, a termination judgment is performed at each visiting node by repeatedly generating a uniform random number within $[0, 1]$. If a random number does not exceed α_{index}, the random walk terminates. Otherwise, the random walk moves to the next node. In the case of $path_v$, random numbers generated at $v, v_1, v_2, ..., v_{l-1}$ do not exceed α_{index}, and a random number generated at v_l exceeds α_{index}. In the following, the random numbers generated at $v, v_1, v_2, ..., v_l$ are denoted as $[r_v, r_{v_1}, r_{v_2}, ..., r_{v_l}]$, respectively.

To generate $path$ whose termination probability is α, αFlexWalk need to determine the magnitude relationships between α and each random number $r \in [r_v, r_{v_1}, r_{v_2}, ..., r_{v_l}]$. If $r \leq \alpha$, $path$ terminates at the corresponding node. In this case, αFlexWalk cuts $path_v$ to generate $path$. If all random numbers in $[r_v, r_{v_1}, r_{v_2}, ..., r_{v_l}]$ exceed α, $path$ needs to continue a random walk after v_l. In this case, to generate $path$, αFlexWalk connects another

indexed path from v_l to $path_v$.

5.4 Proposed Method: αFlexWalk

This section proposes an index-based method, αFlexWalk, to generate random walk paths for an arbitrary α by efficiently utilizing the random numbers generated during indexing. The following part will explain the details of the αFlexWalk algorithm by dividing the cases based on the magnitude relationship between α and α_{index}. Note that when $\alpha = \alpha_{index}$, αFlexWalk just outputs the indexed paths as they are, and hence, it is sufficient to describe the case of $\alpha \neq \alpha_{index}$. Moreover, αFlexWalk may run out of indexed paths in some cases. Therefore, how to deal with the index shortage will be described. The detailed algorithm of αFlexWalk for the case of $\alpha < \alpha_{index}$ and $\alpha > \alpha_{index}$ are shown in Algorithms 3 and 4, respectively. In Algorithms 3 and 4, $Idx.get(v)$ returns the random walk path longer than one starting from node v while referencing the index. The processing of $Idx.get(v)$ is related to the problem of the index shortage. The details will be discussed in detail in Section 5.4.3.

5.4.1 When $\alpha < \alpha_{index}$

When $\alpha < \alpha_{index}$, a random walk path $path$ for the termination probability α does not terminate at the intermediate nodes of $path_v$, and αFlexWalk only considers connecting the indexed paths. The reason is that when gen-

erating $path_v$, all random numbers used for the termination judgment at $v, v_1, v_2, ..., v_{l-1}$, denoted as $r_v, r_{v_1}, r_{v_2}, ..., r_{v_{l-1}}$ exceed α_{index}. In this case, it is ensured that $\alpha < \alpha_{index}$. Therefore, it is clear that $[r_v, r_{v_1}, r_{v_2}, ..., r_{v_{l-1}}]$ exceed α, too. From the above, it is guaranteed that $path$ does not terminate at node v to v_{l-1}.

On the other hand, $path$ probabilistically terminates at the tail node of $path_v$, denoted as v_l. When $path_v$ is generated, it is ensured that $r_{v_l} \leq \alpha_{index}$. However, it is not ensured that $r_{v_l} \leq \alpha$, because $\alpha < \alpha_{index}$. Here, the key point is that r_{v_l} can be treated as the uniform random number within $[0, \alpha_{index}]$, and the probability that the uniform random number within $[0, \alpha_{index}]$ does not exceed α is α/α_{index}. Therefore, αFlexWalk terminates $path$ at v_l with the probability α/α_{index} and does not terminate and continue to move with the probability $1-\alpha/\alpha_{index}$. If $path$ terminates at v_l, αFlexWalk outputs $path_v$ as it is. Otherwise, αFlexWalk connects one of the indexed paths starting from v_l, denoted as $path_{v_l}$, to $path_v$. After connecting $path_v$, it is also ensured that $path$ does not terminate at intermediate nodes of $path_{v_l}$. As with the same case of $path_v$, αFlexWalk determines whether $path$ terminates at the tail nodes of $path_{v_l}$ with the probability α/α_{index}. By repeating this process until $path$ terminates with the probability α/α_{index}, αFlexWalk completes to generate the guaranteed $path$ for the termination probability α.

The length of $path$ needs to increase after $path_{v_l}$ is connected. However, if the length of $path_{v_l}$ is one, $path$ will not get longer after the connec-

tion. To solve this problem, αFlexWalk restricts the length of the indexed paths to greater than one.

In this restriction, the length of the generated paths is always greater than one. Therefore, to ensure that the paths generated by αFlexWalk include paths of length one, this chapter proposes a method to generate paths of length one and others separately. If αFlexWalk can determine the guaranteed number of paths of length one and more respectively, the accuracy of αFlexWalk will be ensured.

When generating ω paths for the termination probability α, αFlexWalk determines the number of paths of length one with the binomial distribution. The binomial distribution is the probability distribution that the number of successes follows when a Bernoulli trial with a success probability p is repeated k times. The probability that a random walk length is one is α, and each path generation is considered a Bernoulli trial. Consequently, the number of length one paths also follows the binomial distribution. Note that the random numbers following the binomial distribution are generated in O(1) time complexity by the inverse transform method [72]. By performing this processing, αFlexWalk generates paths of length one and others separately.

An example of αFlexWalk when $\alpha < \alpha_{index}$ on the sample graph shown in Figure 2.1 is presented below. In this example, $s = a$, $\alpha = 0.1$, $\alpha_{index} = 0.2$, $\omega = 3$, and assume that the indexed paths at nodes a, b, c are as shown in Table 5.1. Note that all indexed paths are longer than one, as

Algorithm 3 αFlexWalk (when $\alpha < \alpha_{index}$)

Input: Source node s, termination probability α ($< \alpha_{index}$), the number of random walks ω, index Idx
Output: List of the random walk paths *paths*

```
1: paths ← empty list;
2: α_index ← Idx.α;
3: length_one_path_count ← bin_dist(α, ω);
4: for i = 1 to length_one_path_count do
5:    paths.append([s]);
6: end for
7:
8: for i = 1 to ω − length_one_path_count do
9:    path ← Idx.get(s);
10:   terminate_prob ← α / α_index;
11:   while random() > terminate_prob do
12:      current_node ← path.back();
13:      path.pop();
14:      path.connect(Idx.get(current_node));
15:   end while
16:   paths.append(path);
17: end for
```

shown in Table 5.1.

First, αFlexWalk determines the number of length-one paths out of three paths. By generating a random number following the binomial distribution with $p = \alpha = 0.1$ and $k = \omega = 3$, αFlexWalk gets the guaranteed number, assuming the random number is one in this case. Therefore, the first *path* is $[a]$.

Second, αFlexWalk generates the remaining two random walk paths longer than one. As the second *path* generation, αFlexWalk references the index of node a and gets the indexed path $[a, b, d]$. αFlexWalk then de-

Table 5.1: Sample Indexed Paths (Excerpt).

Source node	Indexed paths
a	$[a, b, d], [a, b, c]$
b	$[b, d, f], [b, c, a, b], [b, c, e]$
c	$[c, d], [c, e], [c, e, f, e], [c, a]$

termines whether *path* terminates at the tail node d with the probability $\alpha/\alpha_{index} = 0.5$. For the judgment, αFlexWalk generates a uniform random number within [0, 1] and checks whether it exceeds 0.5. Assuming that the uniform random number is 0.4, which does not exceed 0.5, *path* terminates at node d. Consequently, the second *path* is $[a, b, d]$.

As the third *path* generation, αFlexWalk references the index of node a and gets the indexed path $[a, b, c]$. Note that it cannot reference $[a, b, d]$ because every random walk must be performed independently, and an indexed path must not be referenced multiple times during the query. αFlexWalk then determines whether *path* terminates at the tail node c with the probability of 0.5. Assuming that the uniform random number for the judgment is 0.8, which exceeds 0.5, *path* continues to move from node c. Then, αFlexWalk references the index of node c and gets the indexed path $[c, d]$. Afterward, it connects $[c, d]$ to $[a, b, c]$, resulting in $[a, b, c, d]$. Similarly, αFlexWalk determines whether *path* terminates at the tail node d with the probability 0.5. Assuming that the uniform random number is 0.2, which does not exceed 0.5, *path* terminates at node d. Consequently, the third *path* is $[a, b, c, d]$. From the above, αFlexWalk generates three guaranteed paths $[a], [a, b, d], [a, b, c, d]$.

At last, the time complexity of αFlexWalk when $\alpha < \alpha_{index}$ is $O(\omega \cdot \alpha_{index}/\alpha)$. The reason is that the process of αFlexWalk is constructed with the generating length-one paths and other paths. Clearly, the time complexity of the former process is $O(\omega \cdot \alpha)$. For the latter one, the expected number of generated paths is $\omega \cdot (1 - \alpha)$, and the expected number of index references is α_{index}/α. Therefore, the time complexity of the latter process is $O(\omega \cdot (1-\alpha) \cdot \alpha_{index}/\alpha)$. To conclude, the total time complexity of αFlexWalk becomes $O(\omega \cdot \alpha_{index}/\alpha)$.

5.4.2 When $\alpha > \alpha_{index}$

When $\alpha > \alpha_{index}$, a random walk path *path* for the termination probability α always terminates until the tail node v_l. The reason is that when generating $path_v$, it is ensured that $r_{v_l} \leq \alpha_{index}$. Considering $\alpha > \alpha_{index}$, it is clear that $r_{v_l} < \alpha$. Therefore, *path* always terminates until v_l. Consequently, αFlexWalk only needs to consider cutting $path_v$.

On the other hand, *path* probabilistically terminates at each intermediate node of $path_v$. When generating $path_v$, all random numbers $r \in [r_v, r_{v_1}, r_{v_2}, ..., r_{v_{l-1}}]$ exceeds α_{index}. The key point is that r can be treated as the uniform random number within $(\alpha_{index}, 1]$. The probability that the uniform random number within $(\alpha_{index}, 1]$ does not exceed α is $(\alpha - \alpha_{index})/(1 - \alpha_{index})$. Therefore, αFlexWalk generates *path* by judging the magnitude relationships between a uniform random number within $[0, 1]$, denoted as r, and $(\alpha - \alpha_{index})/(1 - \alpha_{index})$ at each next node of $path_v$. It cuts

$path_v$ when $r < (\alpha - \alpha_{index})/(1 - \alpha_{index})$. If no random number less than $(\alpha - \alpha_{index})/(1 - \alpha_{index})$ is generated until node v_{l-1}, αFlexWalk output $path_v$ as it is because $path$ is ensured to terminate until node v_l.

The problem here is that if αFlexWalk judges whether $path$ terminates at each node in $path_v$, it performs $|path_v| - 2$ judgments at most, resulting in the same time complexity of the index-free method. To solve this problem, αFlexWalk determines the timing such that the random number does not exceed $(\alpha - \alpha_{index})/(1 - \alpha_{index})$ in a single random number generation by leveraging the geometric distributions. The geometric distribution is the probability distribution followed by the number of repeated Bernoulli trials with a success probability p until the first successful trial. As the generation of the uniform random numbers is the Bernoulli trial, the timing such that the random number does not exceed the threshold also follows the geometric distribution. Note that the inverse transform method [72] generates the random number following the geometric distribution in O(1) time complexity.

αFlexWalk generates length-one paths and others separately when $\alpha < \alpha_{index}$, and this technique can be applied when $\alpha > \alpha_{index}$. In this case, αFlexWalk generates length-one paths by the binomial distribution and other paths by cutting the indexed paths. Therefore, the judgment of the magnitude relationships between a uniform random number and $(\alpha - \alpha_{index})/(1 - \alpha_{index})$ are performed from node v_1, the second node of $path_v$. By doing this, αFlexWalk generates length-one paths and others separately.

An example of αFlexWalk when $\alpha > \alpha_{index}$ on the sample graph

Algorithm 4 αFlexWalk (when $\alpha > \alpha_{index}$)

Input: Source node s, termination probability α ($> \alpha_{index}$), the number of random walks ω, index Idx
Output: List of the random walk paths *paths*

```
1: paths ← empty list;
2: α_index ← Idx.α;
3: length_one_path_count ← bin_dist(α, ω);
4: for i = 1 to length_one_path_count do
5:    paths.append([s]);
6: end for
7:
8: for i = 1 to ω − length_one_path_count do
9:    path ← Idx.get(s);
10:   terminate_prob ← (α − α_index) / (1 − α_index);
11:   trial_count ← geo_dist(terminate_prob);
12:   if trial_count + 1 < path.size() then
13:      path ← path.resize(trial_count + 1);
14:   end if
15:   paths.append(path);
16: end for
```

shown in Figure 2.1 is presented below. In this example, $s = a$, $\alpha = 0.6$, $\alpha_{index} = 0.2$, $\omega = 3$, and assume that the indexed paths are the same as in the previous example, as shown in Table 5.1. First, αFlexWalk determines the number of length-one paths out of three paths, as is the same case with $\alpha < \alpha_{index}$. Assuming that the number of length-one paths is one in this case, the first *path* is $[a]$.

Second, αFlexWalk generates the remaining two random walk paths longer than one. As the second *path* generation, αFlexWalk references the index of node a and gets the indexed path $[a, b, d]$. αFlexWalk then determines when a uniform random number within [0, 1] does not exceed

$(\alpha - \alpha_{index})/(1 - \alpha_{index}) = 0.5$ with the geometric distribution. Assuming that the random number gained from the geometric distribution is one, indicating that a uniform random number within [0, 1] does not exceed α at the second node. Consequently, the second *path* is $[a, b]$.

As the third *path* generation, αFlexWalk references the index of node a and gets the indexed path $[a, b, c]$. In this case, assuming that the random number gained from the geometric distribution is 5, indicating that a uniform random number within [0, 1] consistently exceeds α until the tail node. Consequently, αFlexWalk does not cut the path and determines that the second *path* is the same as the indexed path $[a, b, c]$. From the above, αFlexWalk generates three guaranteed paths $[a], [a, b], [a, b, c]$.

At last, the time complexity of αFlexWalk when $\alpha > \alpha_{index}$ is $O(\omega)$. The reason is that the process of αFlexWalk is constructed with the generating length-one paths and other paths. Clearly, the time complexity of the former process is $O(\omega \cdot \alpha)$. For the latter one, the expected number of generated paths is $\omega \cdot (1-\alpha)$. The time complexity of generating each path is $O(1)$ because the number of index references is always one, and the cutting point is determined with $O(1)$ time complexity. To conclude, the total time complexity of αFlexWalk becomes $O(\omega)$.

5.4.3 How to Deal with the Index Shortage

An indexed path must not be referenced multiple times during a query because all random walks must be performed independently. However, it is difficult to expect the number of index references for each node before the query. For example, αFlexWalk may reference the index of a specific node v many times when $\alpha < \alpha_{index}$. The reason is that the number of index references while connecting the indexed paths has no upper limit, and the index references may concentrate on v. Therefore, αFlexWalk may use up the indexed paths during a query. It is necessary to consider what to do in the case of index shortages.

When index shortages occur during a query, αFlexWalk performs a part of the random walks on the fly. If there are no indexed paths that are unused in the query, αFlexWalk performs a random walk until it reaches a node with unused indexed paths. As soon as αFlexWalk obtains unused indexed paths, it does not perform the subsequent random walks and references them instead. For example, when the index shortage occurs at node s, αFlexWalk moves the random walk to the s's randomly selected adjacent node v. Afterward, αFlexWalk performs the termination judgment with the probability α_{index}. If it does not terminate, αFlexWalk checks whether v has the unused indexed paths. When v has the ones, generating the path for the termination probability α_{index} is completed by referencing the indexed path starting from v and connecting it to the path $[s, v]$. On the other hand, when v does not have, αFlexWalk moves the random walk to the random adjacent

node of v and continues to perform the same process. This process minimizes the number of on-the-fly random walk transitions.

Algorithm 5 shows the detail of the $Idx.get()$ function. It returns the random walk path longer than length-one, starting from s, for the termination probability α_{index}, using Idx while considering the index shortage. In Algorithm 5, ref_count_map is a map that manages the number of index references for each node v during the query with the node v as the key. It prevents a specific indexed path from being referenced multiple times in a query. It initializes *current_node* with s and *path* with $[s]$ (lines 1–2). Afterward, it continues the random walk for the termination probability α_{index} until it reaches the node that has the unused path (lines 11–12), connects that unused path to *path* (lines 6–8), and completes the generation of *path* (line 9).

Moreover, in FORA+, the default number of indexed paths starting from node v is $\lceil |N_{out}(v)|/\alpha_{index} \rceil$. By changing this size, αFlexWalk can balance the frequency of index shortages and the space overhead of the index. The coefficient c is introduced to decide the balance; the number of indexed paths starting from node v is set as $\lceil c \cdot |N_{out}(v)|/\alpha_{index} \rceil$. Increasing c results in decreasing the likelihood of index shortages, and vice versa. Note that if $c \geq 1$, the index shortage occurs only when $\alpha < \alpha_{index}$. The reason is that when $\alpha > \alpha_{index}$, the index re-references to connect paths never occur, and it is ensured that the number of index references for each node v is less than or equal to $\lceil c \cdot |N_{out}(v)|/\alpha_{index} \rceil$. At last, although increasing c reduces the

Algorithm 5 $Idx.get()$

Input: Index Idx, source node s, map that manages the number of the index references for each node ref_count_map
Output: A random walk path $path$

```
1: current_node ← s;
2: path ← [s];
3: repeat
4:    ref_count ← ref_count_map[current_node];
5:    if ref_count < Idx[current_node].size() then
6:       path.pop();
7:       path.connect(Idx[current_node][ref_count]);
8:       ref_count_map[current_node]++;
9:       break;
10:   else
11:      current_node ← random adjacent node of current_node;
12:      path.append(current_node);
13:   end if
14: until random() < α_index;
```

likelihood of index shortages, index shortages cannot be eliminated because of the unpredictability of the index referencing pattern. Section 5.5.4 will show the experimental results of the trade-off.

5.4.4 Accuracy Guarantee

This section proves that Algorithms 3, 4, and 5 generate the guaranteed random walk paths for the termination probability α. In particular, this section shows that the $Idx.get()$ function presented in Algorithm 5 returns a path for the termination probability α_{index} and its length greater than one. Afterward, it will be proved that Algorithms 3 and 4 generate guaranteed paths for the termination probability α.

Algorithm 5 splits at lines 5–9 and 11–12 depending on whether the index of *current_node* has been used up. In both cases, it is clear that the output is longer than one at this time. Moreover, lines 6–7 can be replaced with Monte Carlo because the indexed path is connected here. In this case, it is evident that the random walk terminates with probability α_{index} at every second or later node, without depending on whether the index for each node is exhausted. From the above, Algorithm 5 returns the random walk path for the termination probability α_{index} and its length greater than one.

Next, considering the correctness of Algorithms 5, Algorithms 3 and 4 return the guaranteed paths for the termination probability α. Obviously, the distribution of the number of length-one paths and others is ensured by the binomial distribution. Therefore, it is enough to show that each random walk with a length of at least two terminates with probability α at every second or later node.

When $\alpha < \alpha_{index}$, Algorithm 3 is considered to perform a two-phase termination judgment at every second or later next node. In particular, in each visiting node, Algorithm 3 performs the termination judgments for the termination probability α_{index} and α/α_{index} twice. When both judgments are satisfied, the random walk terminates. As these two judgments are performed independently, the probability such that the random walk terminates at each visiting node is $\alpha_{index} \cdot \alpha/\alpha_{index} = \alpha$. Consequently, each longer than one random walk path terminates with probability α at every second or later next node.

When $\alpha > \alpha_{index}$, the path generated by Algorithm 4 terminates at the lth visiting node with the probability $(1-\alpha)^{l-2}\alpha$ ($l \geq 2$). Here, the case that a path terminates at the lth visiting node is divided into the two following cases.

1. The length of the indexed path is greater than or equal to l, and the output of the *geo_dist*() function in line 11 is $l-1$.

2. The length of the indexed path is l, and the output of the *geo_dist*() function in line 11 is at least l.

Firstly, Equation (5.1) represents the probability such that the length of the indexed path becomes $x(\geq 2)$, denoted as $P_1(x)$.

$$P_1(x) = (1-\alpha_{index})^{x-2} \cdot \alpha_{index} \tag{5.1}$$

Secondly, Equation (5.2) represents the probability such that the output of the *geo_dist*() function becomes $y(\geq 1)$, denoted as $P_2(y)$.

$$P_2(y) = \left(1 - \frac{\alpha - \alpha_{index}}{1-\alpha_{index}}\right)^{y-1} \cdot \frac{\alpha - \alpha_{index}}{1-\alpha_{index}} \tag{5.2}$$

Therefore, the occurrence probability of case 1 described above, denoted as P_3, is represented by Equation (5.3).

$$\begin{aligned} P_3 &= \Sigma_{x=l}^{\infty} P_1(x) P_2(l-1) \\ &= (1-\alpha)^{l-2} \cdot \frac{\alpha - \alpha_{index}}{1-\alpha_{index}} \end{aligned} \tag{5.3}$$

On the other hand, Equation (5.4) represents the occurrence probability of case 2, denoted as P_4.

$$\begin{aligned} P_4 &= \Sigma_{y=l}^{\infty} P_1(l) P_2(y) \\ &= \frac{\alpha_{index}(1-\alpha)^{l-1}}{1-\alpha_{index}} \end{aligned} \tag{5.4}$$

Consequently, when $\alpha > \alpha_{index}$, the probability P that the path generated by Algorithm 4 terminates at the lth visiting node is represented by Equation (5.5), indicating the correctness of Algorithm 4.

$$P = P_3 + P_4 = (1-\alpha)^{l-2}\alpha \tag{5.5}$$

Finally, when $\alpha = \alpha_{index}$, αFlexWalk obviously returns the path for the termination probability α because it returns the indexed path as it is.

5.5 Evaluation

5.5.1 Index Size

αFlexWalk uses the parameter index size coefficient c to control the index size. This evaluation shows the relationships between c and the total index size when $\alpha_{index} = 0.2$. Note that the number of indexed paths starting from each node v is $\lceil c \cdot |N_{out}(v)|/\alpha_{index} \rceil$, and when $c = 1$, the index size is the same as default FORA+.

Figure 5.1 shows the relationships between the index size coefficient c and the index size. For the sake of clarity, the nine datasets are divided into

three groups according to their size, and the results for each group are shown in Figure 5.1a, Figure 5.1b, and Figure 5.1c, respectively. As expected in Section 5.4.3, the index size is proportional to c.

To compare the index size among nine datasets, Figure 5.2 shows the bar graph indicating the index size when $\alpha_{index} = 0.2$ and $c = 1$. As shown in Figure 5.2, it is clear that even for the most extensive dataset, the total index size is less than 1000 GB. From the above, the scalability of the indexing scheme is confirmed.

5.5.2 Processing Time of Random Walk Query

This evaluation compares αFlexWalk with Monte Carlo, which is the typical index-free method, in terms of the processing time to show the algorithmic contribution. I measure the processing time of αFlexWalk and the index-free Monte Carlo.

Regarding the parameters of random walk queries, the source nodes are randomly selected 100 nodes, $\omega = 10^3$, and $\alpha \in [0.0125, 0.025, ..., 0.9375, 0.95]$. Regarding the parameters of αFlexWalk, $c = 1$, and $\alpha_{index} \in [0.1, 0.15, 0.2, 0.25, 0.3]$. However, for Twitter and Friendster, only a part of α_{index} is evaluated due to the memory budgets. $\alpha_{index} \in [0.2, 0.25, 0.3]$ for Twitter, and $\alpha_{index} = 0.3$ for Friendster.

Figure 5.3–Figure 5.5 show the processing time of αFlexWalk and Monte Carlo. The x-axis represents the queried α, and the y-axis represents

the average processing time for the 100 source nodes. The series indicated by the solid line and circle plots show the results of αFlexWalk, and the color of each plot represents α_{index}. The series indicated by dashed lines and square plots show the results of Monte Carlo.

According to Figure 5.3–Figure 5.5, the processing time decreases as α_{index} decreases and α increases. In particular, the processing time of αFlexWalk improves by up to [2.3, 11.2] times compared with that of Monte Carlo for each dataset. To be more specific, the processing time of αFlexWalk behaves differently when $\alpha < \alpha_{index}$ and $\alpha \geq \alpha_{index}$. When $\alpha < \alpha_{index}$, the decrease speed in processing time against α is fast in the case of $\alpha > \alpha_{index}$. The reason is that the number of index references decreases at different speeds against α when $\alpha < \alpha_{index}$ and $\alpha \geq \alpha_{index}$.

When $\alpha < \alpha_{index}$, the expected number of index references to generate a path is α_{index}/α. On the other hand, when $\alpha > \alpha_{index}$, the expected number of index references to generate a path is always one in the case of generating a path longer than one. Although the number of paths longer than one decreases against α, the decreasing speed in processing time against α is less than that in the case of $\alpha < \alpha_{index}$. As the number of index references does not differ by α_{index}, the processing time is also the same among different α_{index} when $\alpha > \alpha_{index}$.

Moreover, as shown in Figure 5.3, the processing times of the index-based αFlexWalk and the index-free Monte Carlo are almost the same for minor α in DBLP and Web-BerkStan. The reason is that their graph size is

trivial, and most of the graph data can be handled in the cache memory. As a result, the merit of using the index in these small datasets is not significant.

5.5.3 Processing Time of PPR Query

This evaluation compares the processing time of the Monte Carlo phase in the FORA framework to show the practical contribution. I incorporate αFlexWalk into FORA+ and measure the processing time of the Monte Carlo phase during PPR computation. I also measure the processing time of the Monte Carlo phase in FORA, not using an index, to clarify the difference in processing time between both methods.

Regarding the parameters of PPR queries, the source nodes are randomly selected 100 nodes, ω is $\alpha \cdot 10^6$, and α is changed to [0.0125, 0.025, ..., 0.9375, 0.95]. Here, I determine ω based on α because it has been reported that the computational accuracy in PPR depends on the total path length of the random walks, and by setting $\omega = \alpha \cdot 10^6$, the total path length is fixed at 10^6. The parameters of αFlexWalk are the same as the evaluation in Section 5.5.2. Note that there is no randomness in the Forward Push phase of the FORA framework, so if the parameters of the PPR query are the same, the random walk queries generated in the Monte Carlo phase will also be the same.

Figure 5.6–Figure 5.8 show the results. The representation of the figures is the same as in Figure 5.3–Figure 5.5. The whole trend is similar to

the evaluation in Section 5.5.2: αFlexWalk is faster than index-free Monte Carlo in most cases, and the processing time decreases as α_{index} decreases and α increases. The processing time of FORA+ with αFlexWalk improves by up to [1.3, 14.4] times compared with that of FORA with Monte Carlo for each dataset.

Furthermore, although this evaluation fixes total random walk length among α, the results show that the processing time decreases against α. The reason is that the number of paths that terminate in the Forward Push phase of the FORA framework decreases against α.

Moreover, in directed datasets such as Web-BerkStan, Flickr, and Twitter, the processing time rapidly decreases when α approaches zero. The reason is that a long random walk reaches dangling nodes, whose degree is zero, before it terminates with the probability α. When it reaches the dangling node, it automatically terminates. As a result, the processing time behaves differently compared with the results of undirected graphs.

5.5.4 The Frequency of the Index Shortages

αFlexWalk performs a random walk without using the index until it reaches a node where an unused indexed path exists when the index shortages occur. In addition, the frequency of index shortages is controlled by a parameter called the index size coefficient c. Therefore, I measure the frequency of index shortages and the number of random walks performed in the Monte

Carlo phase of FORA+ while changing c. I then obtain the frequency of index shortages in a single random walk. Regarding the parameters of PPR queries, the source node is randomly selected 100 nodes, and I set α to 0.0125 and ω to $\alpha \cdot 10^6$. Regarding the parameters of αFlexWalk, I set α_{index} to 0.3 and change c to [0.2, 0.4, ..., 1.8, 2]. In order to make it most likely that index shortages will occur, I fix both α and α_{index} to the minimum and maximum values in the evaluation in Section 5.5.3, respectively. The reason why index shortages are more likely to occur is that the smaller α leads to the longer the path, and the larger α_{index} is, the more often the index is referenced. With such parameter settings, I can evaluate the worst case for the overhead of the αFlexWalk.

Figure 5.9 shows the relationship between the index size coefficient c and the average frequency of index shortages in a single random walk for datasets other than Web-BerkStan. Web-BerkStan showed a significantly different trend from the other datasets, so I will discuss it separately later. Figure 5.9 shows that the number of index shortages is four or less for each random walk in the datasets other than DBLP. Even in DBLP, where the number of index shortages is the highest in the figure, the number is at most around 10. Considering that the average path length is 80 when α is 0.0125, the number of index shortages accounts for less than 1/8 of the number of random walk transitions, indicating that most random walk transitions are achieved by the index. As mentioned above, in this evaluation, the parameters are set so that index shortages are most likely to occur; therefore, it is

expected that index shortages will be less likely to occur in other parameter settings.

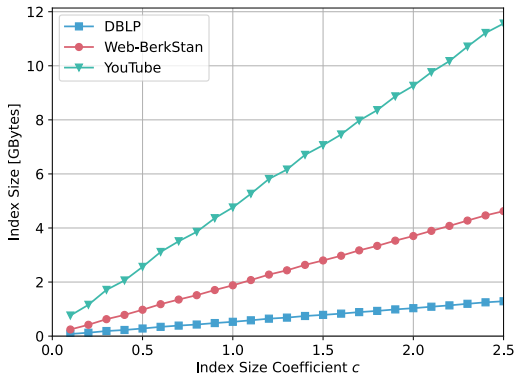
In addition, for datasets other than DBLP, the decrease in the number of index shortages when the index size coefficient is changed from 0.2 to 2 is two or less. From these results, it is apparent that when the memory budget is limited, reducing the index size coefficient to reduce the index space cost has a limited impact on the computation time. This is because, in FORA+, the index size is determined based on the maximum number of paths that can be required in the Monte Carlo phase when $\alpha = \alpha_{index}$ and $c = 1$, and the number of paths required during actual operation is sufficiently smaller than the worst-case scenario FORA+ assumes. On the other hand, in DBLP, by changing the index size coefficient from 0.2 to 2, the frequency of index shortages is reduced by almost half. The reason for this is supposed to be that in DBLP, index references tend to concentrate on specific nodes.

Furthermore, Figure 5.10 shows the results for Web-BerkStan. For detailed analysis, I change α to [0.0125, 0.1, 0.2, 0.4, 0.6]. In the case where $\alpha = 0.0125$ and $c = 0.2$, the number of index shortages exceeds 40, and about half of the transitions are performed without an index. The reason for this is thought to be that Web-BerkStan is a directed graph that is not strongly connected. In other words, it is thought that frequent index shortages are caused by the fact that index accesses are concentrated on nodes in a particular subgraph when a long random walk is performed. Note that, in Web-BerkStan, the change in the number of index shortages for c is relatively

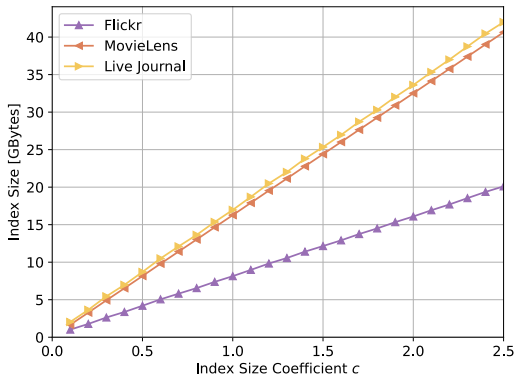
large. Therefore, it is recommended that the index size should be increased in environments where high-speed computation is required.

5.6 Conclusion of This Chapter

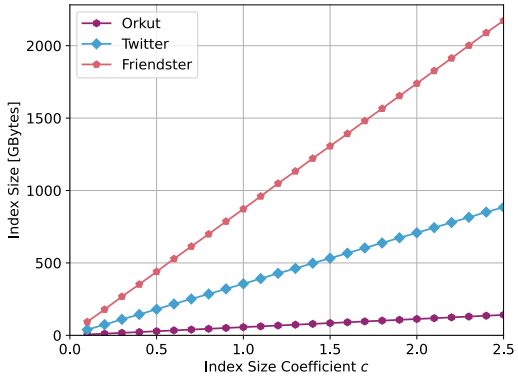
Existing index-based random walk path generation methods accept the specific termination probability α, resulting in the inflexible graph analysis. This chapter focused on the fact that modifying α causes the probabilistic change in the length of random walks. It proposed αFlexWalk that connects and cuts the indexed paths to generate guaranteed paths for any α. Evaluations with nine real-world datasets found that αFlexWalk achieved at most 11.2 times speedup compared to the index-free random walk method.



(a) Small datasets.



(b) Medium datasets.



(c) Large datasets.

Figure 5.1: Index size coefficient vs. Index size.

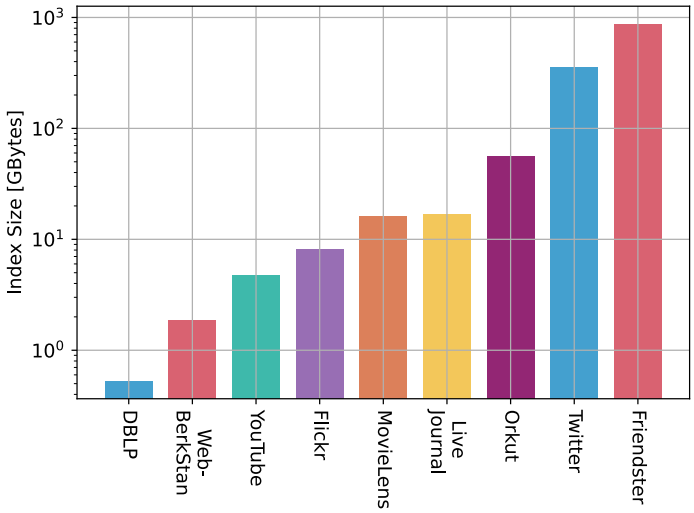
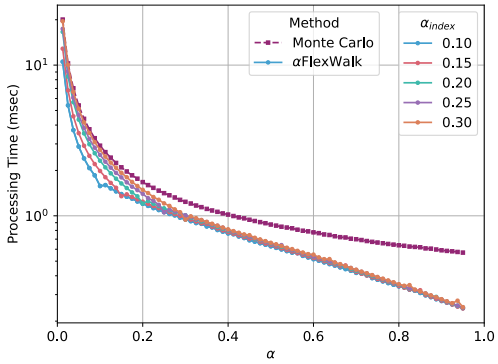
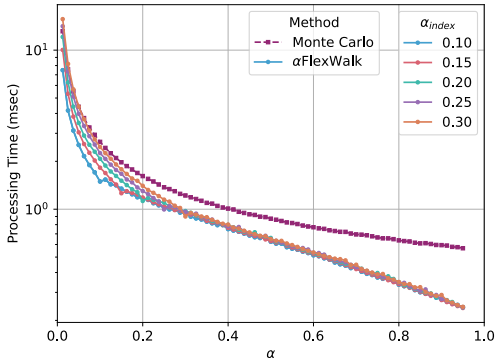


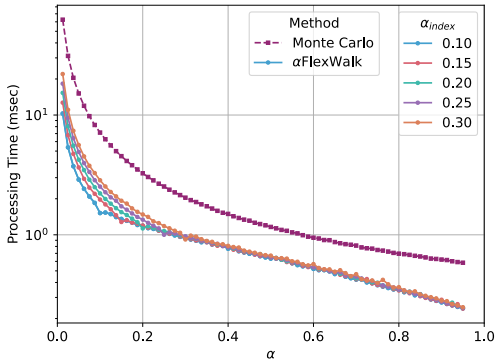
Figure 5.2: Index size ($\alpha_{index} = 0.2$, $c = 1$).



(a) DBLP

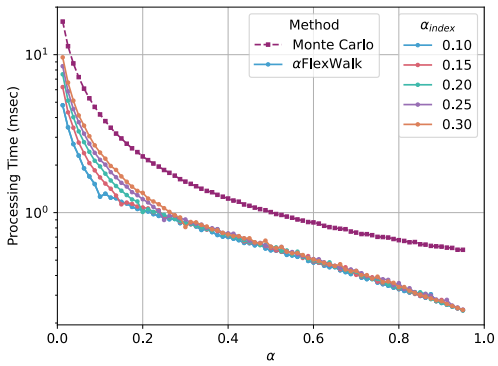


(b) Web-BerkStan

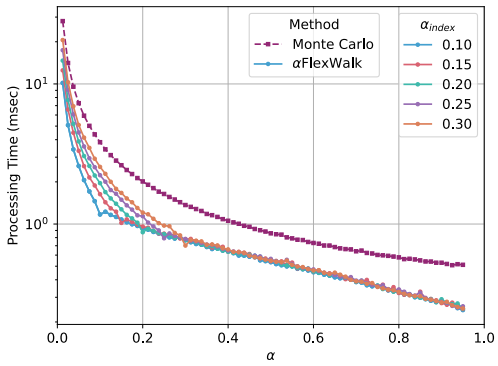


(c) YouTube

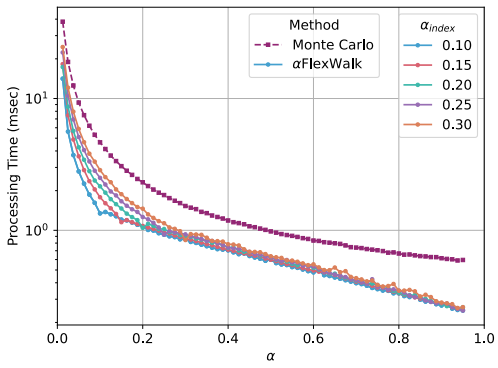
Figure 5.3: α vs. average processing time of random walks for each α_{index} (small datasets).



(a) Flickr

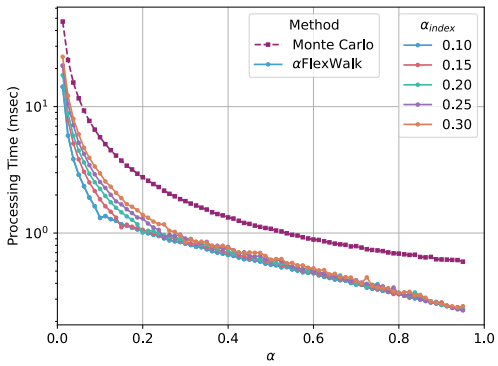


(b) MovieLens

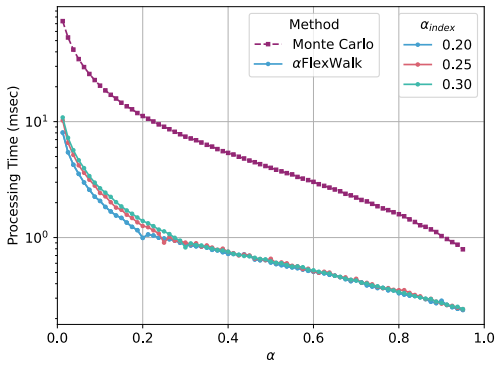


(c) Live Journal

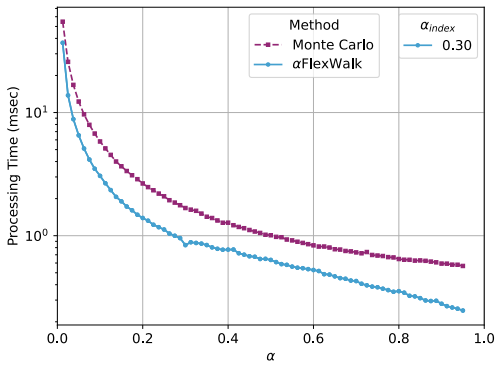
Figure 5.4: α vs. average processing time of random walks for each α_{index} (medium datasets).



(a) Orkut

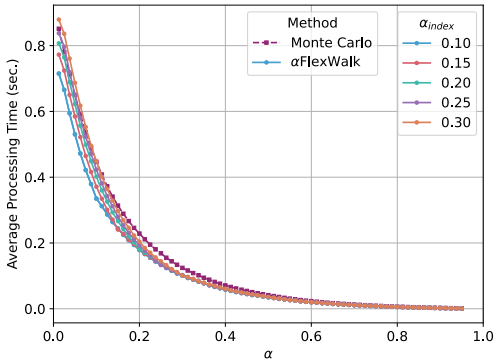


(b) Twitter

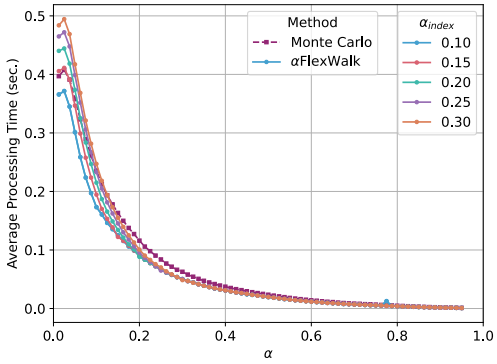


(c) Friendster

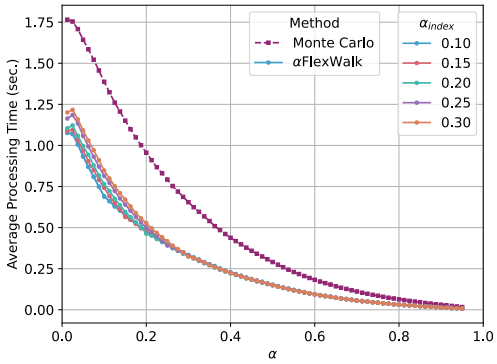
Figure 5.5: α vs. average processing time of random walks for each α_{index} (large datasets).



(a) DBLP

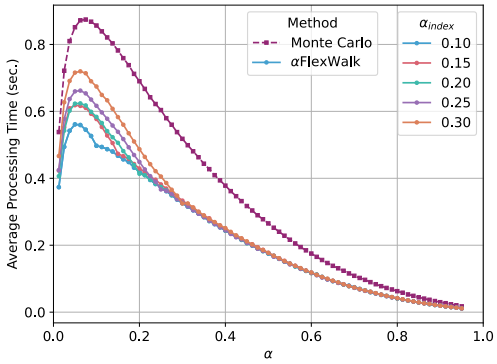


(b) Web-BerkStan

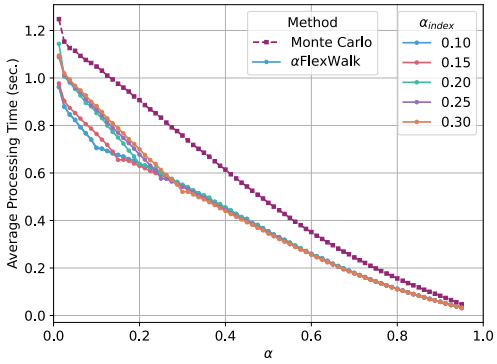


(c) YouTube

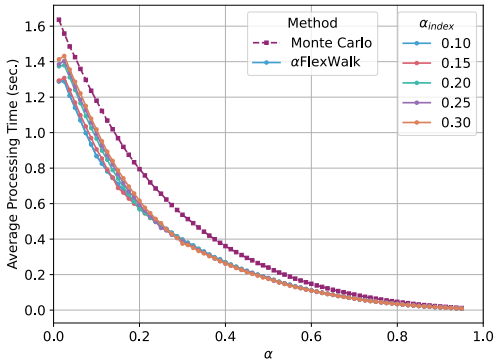
Figure 5.6: α vs. average processing time of PPR for each α_{index} (small datasets).



(a) Flickr

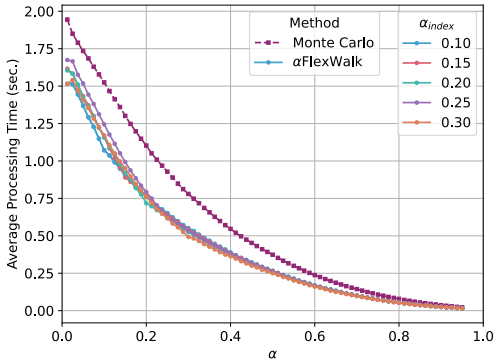


(b) MovieLens

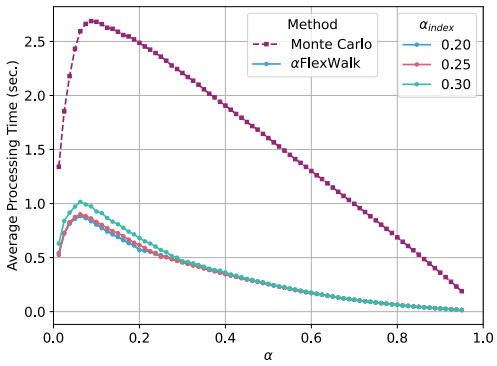


(c) Live Journal

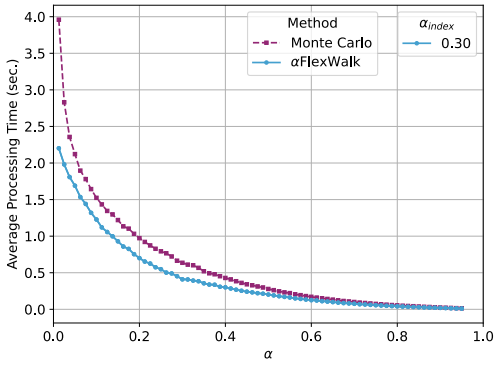
Figure 5.7: α vs. average processing time of PPR for each α_{index} (medium datasets).



(a) Orkut



(b) Twitter



(c) Friendster

Figure 5.8: α vs. average processing time of PPR for each α_{index} (large datasets).

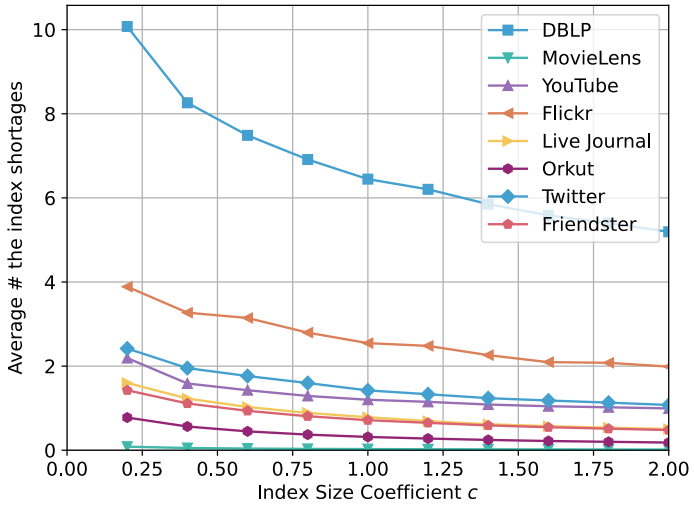


Figure 5.9: Index size coefficient vs. Average number of the index shortages in a path generation when $\alpha_{index} = 0.3$ and $\alpha = 0.0125$ (datasets are other than Web-BerkStan).

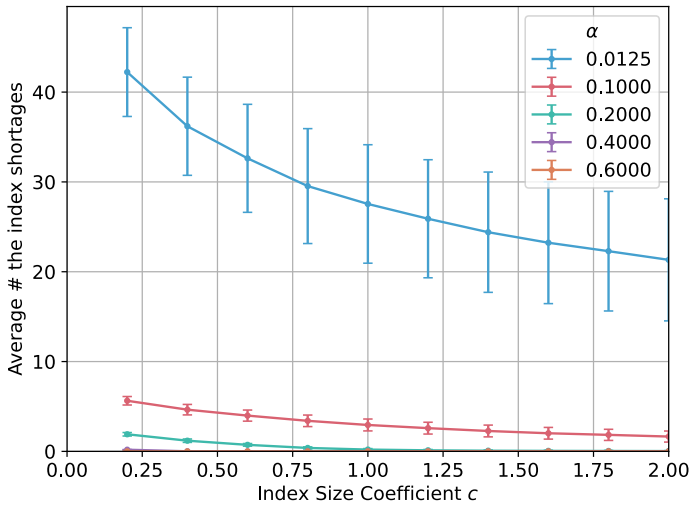


Figure 5.10: Index size coefficient vs. Average number of the index shortages in a path generation when $\alpha_{index} = 0.3$ (dataset is Web-BerkStan).

Chapter 6

Reducing Re-Indexing on Dynamic Graphs

6.1 Overview

Although an index-based FORA+ is effective for fast Personalized PageRank (PPR) computations, as discussed in Chapter 3, computational overhead exists when applying FORA+ to dynamic graphs. In existing methods, an index correction, consisting of index resizing and subsequent re-indexing, is necessary to guarantee accuracy because the index becomes stale as the graph is updated [11–20]. The staleness is caused by the difference between the distribution of the indexed random walks in the before-update and that of the on-the-fly random walks in the after-update graph. In general, PPR accuracy is not guaranteed if the stale index is referenced. Thus, to guarantee accuracy, index resizing and subsequent re-indexing must be performed at the nodes whose index becomes stale whenever an edge update occurs. Index

resizing is an operation that deals with changes in the degree of the updated node. Re-indexing is an operation that re-performs random walks at the nodes whose index is stale.

Index resizing, the first process of the index correction, is a light $O(1/\alpha)$ operation. However, re-indexing, the second process, involves either heavy computation [11, 13, 14] or significant memory overhead [12, 14–20]. The state-of-the-art method that suffers from heavy computation is Agenda- [11, 13]. It quantifies the staleness of the index by performing the reverse push algorithm [73] for each edge update. The time complexity of the reverse push is $O(n)$, where n is the number of edges in the graph. Experimentally, Agenda's index correction time is up to about 1000 seconds, even for a single edge update, with the settings described in Section 6.5.3. On the other hand, the state-of-the-art method that incurs significant memory overhead is Firm [12]. It maintains indexes other than random walk paths to complete re-indexing with $O(1/\alpha)$. The evaluation shows that the Firm's index size is up to 5 TB, which is prohibitively large for practical use (see Section 6.5.2).

This chapter shows that it is possible to achieve comparable accuracy to the guaranteed methods such as Agenda and Firm, both with light computation and low memory overhead in dynamic graphs. To realize comparable accuracy while avoiding heavy re-indexing, an approach that ignores edge updates, causing a negligible impact on accuracy, is adopted. Firstly, Section 6.3 analyzes FORA+ to quantify the impact of edge updates on accuracy and the amount of index references for each node. Secondly, based on

the analysis, Section 6.4 proposes an efficient method to realize comparable accuracy while significantly reducing re-indexing and achieving low memory overhead. Finally, Section 6.5 clarifies the effectiveness of the proposed method in evaluations.

The analysis of FORA+ indicates that the index references concentrate on the nodes whose index is robust to edge updates. As a result, reducing re-indexing for such nodes does not necessarily lead to a loss of accuracy. In particular, I analyze relationships between the number of index references in FORA+ and the likelihood of the index becoming stale at each node. The likelihood of the index becoming stale is quantified by the change in Top-k PPR before and after edge updates. As a result, the more frequently a node's index is referenced, the less likely it is to become stale.

Based on the analysis, this chapter proposes a method that avoids the heavy re-indexing computation until the graph changes to a specific size while achieving comparable accuracy and low memory overhead. When an edge update occurs, the proposed method only performs $O(1/\alpha)$ index resizing and does not perform re-indexing. Consequently, the proposed method can achieve $O(1/\alpha)$ index correction. In addition, the proposed method does not require an index other than the random walk paths. Therefore, the memory overhead of the proposed method is the smallest among existing methods. The space complexity is $O(n + m/\alpha)$. Note that this chapter assumes that $\alpha_{index} = \alpha$, so α_{index} will be represented as α for the simplicity. During PPR computations, the proposed method preferentially refers to the newer index

to minimize the loss of accuracy.

The evaluations clarify the effectiveness of the proposed method and the condition that the proposed method can achieve comparable accuracy using nine real-world datasets. Its index correction time and index size are minimal compared to the state-of-the-art index-based methods. In particular, the index correction time of the proposed method is up to six orders of magnitude faster than that of Agenda, which suffers from heavy re-indexing computation. The index size of the proposed method is also more than 3.3 times smaller than that of the Firm, which incurs significant memory overhead. In addition, the proposed method achieves accuracy comparable to the guaranteed methods. I evaluate accuracy using Normalized Discounted Cumulative Gain (NDCG) [53], which quantifies PPR accuracy with a value in the range $[0, 1]$. As a result, even when edge insertions/deletions occur until the number of edges in the graph increases/decreases by 20% from the index generation, NDCG is more significant than 0.999, comparable to the state-of-the-art methods.

The main contributions of this chapter are summarized as follows.

- The analysis of FORA+ observes that index references concentrate on the nodes whose index is unlikely to become stale.
- Based on the observation, the proposed method ignores edge updates and significantly reduces heavy re-indexing while achieving accuracy comparable to the guaranteed methods.

- Moreover, the only index of the proposed method is the random walk paths, which is the minimum memory overhead among existing methods.
- Evaluations show that the proposed method achieves fast index correction and low memory overhead at the same time.
- Evaluations also show that the proposed method achieves 0.999 NDCG on average until 20% of the edges are updated on all datasets, indicating that the heavy re-indexing of the guaranteed methods contributes only a slight improvement in accuracy.

The remainder of this chapter is structured as follows. Section 6.2 presents related work. Section 6.3 analyzes the characteristics of FORA+'s index references on dynamic graphs. Section 6.4 describes the proposed method. Section 6.5 shows the experimental results. Section 6.6 concludes this chapter.

6.2 Related Work

This section describes the characteristics of the existing index-based methods to compute PPR on dynamic graphs [11–20] and the differences between these methods and the proposed method. The existing methods and the proposed method are categorized into five types: Agenda [11, 13], Firm [12], local push [15–20], Bahmani [14], and the proposed method. These five types

are discussed from three perspectives: computation accuracy, the index that each method maintains, and the time complexity of re-indexing, as shown in Table 6.1.

All existing methods [11–20] aim to guarantee accuracy. As a result, these methods must correct the stale index after every edge update. Index corrections involve either significant memory overhead or heavy re-indexing computation. In contrast, the proposed method aims to achieve accuracy comparable to the guaranteed methods while significantly reducing the re-indexing and memory overhead of the index.

In terms of the index memory overhead, the total space complexity of Agenda [11, 13], Firm [12], and the proposed method are $\Theta(n + m/\alpha)$. Although the space complexities are the same among the three methods, the details of the index are different, and the proposed method is the smallest. All three methods maintain the random walk paths, the same index as FORA+. While the proposed method only maintains that, the other two methods maintain other indexes. Note that the space complexity of the random walk paths is $\Theta(n+m/\alpha)$, as described in Section 2.5. Other indexes of the Agenda and the Firm will be described in detail below.

Agenda maintains one index other than the random walk paths. The first index is the random walk paths, which is the same index as FORA+. Therefore, the space complexity of this index is $\Theta(n + m/\alpha)$. The second index is the inverse graph used in the reverse push algorithm described below. The space complexity of the inverse graph is the same as the space complexity

of the graphs, i.e., $\Theta(n + m)$.

Firm maintains two indexes other than the random walk paths. The first index is a map identifying random walk paths passing through each node. The key is a node, and the value is a list of pointers to paths. The second index is auxiliary data to enable O(1) manipulations to the map's value. The space complexities of these indexes are both $\Theta(n + m/\alpha)$.

The index's memory overhead of local push [15–20] and Bahmani [14] is far more significant than the three methods described above. Local push manages the results of Forward Push for each node as the index to answer queries with O(1) time complexity. The results of Forward Push contain PPR results for all nodes. As a result, the space complexity of local push is $O(n^2)$, which is unacceptable for large graphs. Bahmani maintains similar indexes to the Firm. However, the number of paths maintained for each node is $\Theta(n \log n)$, compared to $\Theta(|O_{out}(v)|)$ in Firm. As a result, the total space complexity is $\Theta(n^2 \log n/\alpha)$, which is also prohibitive for large graphs.

From the viewpoint of the re-indexing's time complexity, that of Firm and local push are light. Firm identifies the stale index after an edge update with O(1) time complexity by referencing the expensive index described above. The time complexity of re-generating the stale index is $O(1/\alpha)$ by exploiting probability distributions. As a result, Firm's total time complexity of re-indexing for an edge update is $O(1/\alpha)$. Note that Firm is required to perform the index resizing as the index correction in addition to re-indexing. As mentioned above, local push maintains the results of Forward Push as

the index. It does not need to redo the process, which leads to light time complexity. Consequently, local push also completes re-indexing with O(1) time complexity.

On the other hand, Agenda and Bahmani suffer from heavy re-indexing. Agenda performs the reverse push algorithm [73] for every edge update as a part of re-indexing. The reverse push is utilized to quantify the staleness of the index for each node. Its time complexity is $O(n)$, making it difficult for dynamic graphs to perform the reverse push for every edge update. Agenda does not complete re-indexing immediately after an edge update. When queried, Agenda performs re-indexing on the nodes whose index is not sufficiently accurate to guarantee accuracy. In particular, Agenda determines the nodes to perform re-indexing by the staleness computed by the reverse push and the number of index references in the Monte Carlo phase of FORA+. The expected time complexity of re-indexing for each query is $O(m^2/\omega\alpha)$. Therefore, Agenda's total time complexity of re-indexing is $O(n + m^2/\omega\alpha)$. Quota [13] is an Agenda-based method. This method optimizes the parameter settings of Agenda while assuming an environment where PPR queries and graph updates are mixed in a dynamic ratio. Since the algorithm for graph updates is equivalent to Agenda, this method also faces the same re-indexing problem. Bahmani needs to generate a lot of random walks for every edge update as re-indexing. The number of paths needed to generate is $\Theta(n \log n)$. Consequently, Bahmani's time complexity of re-indexing is $\Theta(n \log n/\alpha)$, which is heavy for real-world large dynamic graphs.

Table 6.1: The characteristics of existing index-based methods and the proposed method.

Type	Accuracy	Indexes		Time Complexity of Re-Indexing
		Details	Space Complexity	
Proposed method	Comparable to guaranteed methods	Random walks paths	$\Theta(n + m/\alpha)$	0
Agenda [11, 13]	Guaranteed	Random walks paths Inverse graph	$\Theta(n + m/\alpha)$ $\Theta(n + m)$	$O(n + m^2/\omega\alpha)$
Firm [12]		Random walks paths Auxiliary data for re-indexing Auxiliary data for re-indexing	$\Theta(n + m/\alpha)$ $\Theta(n + m/\alpha)$ $\Theta(n + m/\alpha)$	$O(1/\alpha)$
Local Push [15–20]		Results of Forward Push	$O(n^2)$	$O(1)$
Bahmani [14]		Random walks paths Auxiliary data for re-indexing Auxiliary data for re-indexing	$\Theta(n^2 \log n/\alpha)$ $\Theta(n^2 \log n/\alpha)$ $\Theta(n^2 \log n/\alpha)$	$\Theta(n \log n/\alpha)$

In contrast, the proposed method does not perform re-indexing while achieving accuracy comparable to the guaranteed methods. When an edge is updated, it only performs O(1) index resizing as the index correction, which Agenda and Firm also perform.

Finally, the proposed method's characteristics are summarized as follows: it achieves comparable accuracy to the guaranteed methods while avoiding both heavy re-indexing and significant memory overhead; re-indexing is omitted as long as comparable accuracy is achieved; the index is only random walk paths, which is the smallest among existing methods; the space complexity of index is $\Theta(n + m/\alpha)$.

6.3 Analyzing FORA+'s Index References on Dynamic Graphs

This section analyzes the characteristics of FORA+'s index references to show that comparable accuracy to guaranteed methods is achieved even when FORA+ uses the stale index as it is. In particular, *stability* of a node is defined to quantify the unlikelihood of the index becoming stale and show that index references concentrate on the nodes with high *stability*.

The staleness of a node's index is quantified using Normalized Discounted Cumulative Gain (NDCG) [53]. In general, NDCG is used to quantify the approximation accuracy of Top-k PPR computations with the ground truth and the approximated PPR as inputs, as described in Section 4.5. Equation (4.1) defines NDCG. This chapter uses NDCG to quantify the unlikelihood of the index becoming stale for each node, denoted as *stability*, in the following part. In particular, Equation (6.1) defines *stability*. In Equation (6.1), $\boldsymbol{\pi}_v^0$ and $\boldsymbol{\pi}_v^t$ are PPR vectors with respect to node v before and after the edge updates, respectively. Consequently, higher *stability* means that the index of the node is unlikely to become stale because Top-k PPR nodes are unlikely to change due to edge updates. The range of *stability* is also $[0, 1]$ since the range of NDCG is $[0, 1]$. Note that *stability* is a symmetric measure for $\boldsymbol{\pi}_v^0$ and $\boldsymbol{\pi}_v^t$. Therefore, even if the graph before and after the edge updates are treated inversely, *stability* will have the same value.

$$stability(v,k) = \frac{1}{2}\{ndcg(\boldsymbol{\pi}_v^0, \boldsymbol{\pi}_v^t, k) + ndcg((\boldsymbol{\pi}_v^t, \boldsymbol{\pi}_v^0, k)\} \tag{6.1}$$

The analysis method is as follows. First, I set G^0 as the graph where $m/2$ edges are inserted and generate the index Idx^0 on G^0. Here, m is the number of edges contained in each dataset. Second, I obtain $G^{m/2}$ by inserting the remaining $m/2$ edges. Third, *stability* of all nodes are calculated using G^0 and $G^{m/2}$, and the nodes are classified into groups according to their *stability* values. The k in Equation (6.1) is 128. Fourth, I perform FORA+ and record the number of index references for each node and its group. The source nodes are 100 randomly selected nodes. The order of edge insertions is randomized if the dataset does not contain timestamps. Otherwise, I have two insertion methods: timestamped and randomized order to quantify the effect of the locality of insertions. Although only the edge insertions are considered as graph updates, it is not necessary to consider the edge deletions. The reason is that the same results are obtained when $G_{2/m}$ and G_0 are treated inversely due to the symmetry of *stability*.

Figure 6.1 shows the ratio of index references for each node group as a cumulative graph. The number of groups is 21, and the *stability* of nodes in each group is within [0, 0.8], (0.8, 0.81], (0.81, 0.82], ..., (0.99, 1], respectively. The ratio of index references is normalized so that the sum for all groups is 100%. A plot (x, y) in the figure indicates that y% of all index references are performed on the nodes whose *stability*, which indicates the unlikeliness

of the index change, is less than x. For the dynamic datasets, the solid and dashed lines represent the results for the randomized and timestamped insertions, respectively.

According to Figure 6.1, index references in FORA+ tend to concentrate on the nodes with high *stability*. The ratio of index references to the nodes with $stability \leq 0.9$ is less than 12% on all datasets. In contrast, the ratio of index references to the nodes with $stability > 0.95$ is more than 43%. Moreover, more than 76% of index references concentrate on nodes with *stability* greater than 0.97 on YouTube (timestamped) and Flickr (timestamped). Therefore, it is expected that the accuracy of FORA+ is unlikely to decrease even if re-indexing is omitted and the stale index is used as it is.

Notably, the plots of YouTube (timestamped) and Flickr (timestamped) tend to be smaller than those of the other datasets. This indicates that the index references concentrate at nodes with higher *stability*. However, the plots of randomized YouTube and Flickr approach the plots of the other datasets without timestamps. This suggests that the order of edge updates in the dynamic datasets contributes to this tendency. Note that this means that the index references of the timestamped results are more concentrated on the nodes with higher *stability* than those of the randomized results. It also implies that more positive results are expected with the timestamped data on other datasets.

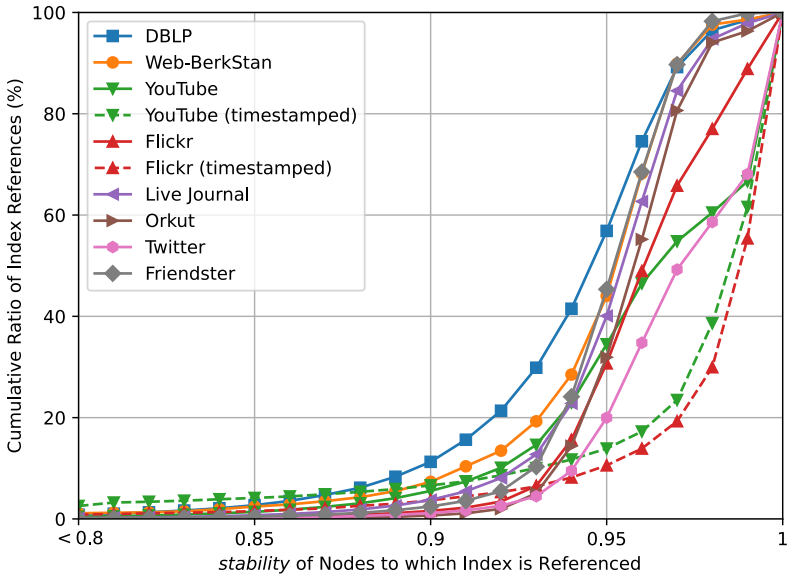


Figure 6.1: The cumulative ratio of index references for each node group of *stability*.

6.4 Proposed Method

This section describes the proposed method in four parts: index generation, index correction for an edge insertion, index correction for an edge deletion, and Top-k PPR computation. First, index generation performs random walks from each node and stores paths. Second, only additional random walks are performed as the index correction when inserting an edge. Third, the proposed method only shrinks the index when an edge deletion occurs. Fourth, the index referencing method is the key to achieving high accuracy in PPR computation. Note that the proposed method references the stale index as long as comparable accuracy can be achieved. It re-generates the index

when comparable accuracy cannot be achieved. The experimental results show the relationship between the number of edge updates and accuracy (see Section 6.5.4).

6.4.1 Index Generation

The index generation method is almost the same as that described in Chapter 5. A key point is to adopt deque as the data structure of the index for node v, denoted as $Idx[v]$. Using the deque, add/remove paths to/from the head/tail of $Idx[v]$ is realized with O(1) time complexity. The index with deque also maintains the generation order. These are suitable for the proposed index correction and PPR computation algorithms described below.

The index is re-generated when it no longer achieves comparable accuracy. The timing of re-generation is determined in two ways. The first way contributes to a coarse computation with a light computational overhead. It is to re-generate the index based on the number of edge updates that occurred from index generation. The evaluation indicates the relationship between the number of edge updates and accuracy. Therefore, by quantitatively defining comparable accuracy, the results experimentally determine the period over which comparable accuracy is achieved. The second way contributes to a more accurate computation at the expense of computational overhead. It is to re-generate the index based on the periodic comparison of the index-based PPR results with the index-free results. The index-free method provides the exact results at the cost of computation time. Therefore, the accuracy of the

stale-index-based results can be checked by comparing the two results. By periodically checking accuracy, the proposed method detects when the index has become too stale. Although the periodic comparison involves some overhead, it is reduced by checking accuracy for only a few random nodes.

6.4.2 Index Correction for an Edge Insertion

The index correction algorithm for an edge insertion is shown in Algorithm 6. It only performs additional random walks from the inserted node and pushes random walk paths back to the deque $Idx[v]$ (lines 4–6). Unlike existing methods, it does not perform any re-indexing. In particular, the number of additional random walks performed from node v is $\lceil c \cdot |N_{out}(v)|/\alpha \rceil$ minus the index size of node v before the edge insertion (line 3). Here, the required number of paths for node v, $\lceil c \cdot |N_{out}(v)|/\alpha \rceil$, is described in detail in Section 5.4. Considering the degree of node v before the edge insertion is $|N_{out}(v)| - 1$, the number of additional random walks is about $1/\alpha$, which means the index correction of the proposed method is completed with $\mathrm{O}(1/\alpha)$ time complexity.

An example of the index correction using the sample graph shown in Figure 2.1 will be presented below. Assume that edge (a, d) is inserted like in Figure 6.2, and the index of node a before the insertion is the paths shown in the row with timestamp 0 in Table 6.2. Here, let α and c be 0.5 and 1, respectively. In this case, the required number of paths maintained in Idx^0 is $\lceil c \cdot |N_{out}(a)|/\alpha \rceil = 4$. After the edge insertion, the number changes to 6

Algorithm 6 Index Correction for an Edge Insertion

Input: Graph $G^t(V^t, E^t)$, inserted edge (v, w), index Idx^{t-1}
Output: Index Idx^t

```
1: Idx^t ← Idx^{t-1};
2: ω_v ← ⌈c · |N_out(v)|/α⌉;
3: ω'_v ← ω_v − Idx^t[v].size();
4: for i = 1 to ω'_v do
5:    path ← random_walk(G^t, v);
6:    Idx^t[v].push_back(path);
7: end for
```

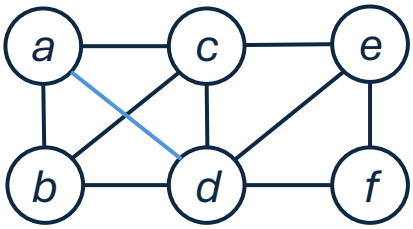


Figure 6.2: The sample graph that edge (a, d) is inserted.

because the degree of node a becomes 3. Therefore, the algorithm performs two additional random walks and adds paths to the tail of Idx^1, as shown in the row with timestamp 1 in Table 6.2.

6.4.3 Index Correction for an Edge Deletion

The index correction algorithm for an edge deletion is shown in Algorithm 7. It only deletes the small number of paths maintained as the deleted node's index. As with edge insertion, it does not perform any re-indexing. In particular, the number of random walk paths that need to be stored at node v decreases when an edge deletion occurs because the degree of node v is decremented. Therefore, it deletes paths in the index so that the index

Table 6.2: Sample Indexed Paths for node a before and after an edge insertion in the graph shown in Figure 6.2 ($\alpha = 0.5$).

Timestamp	Indexed paths
0	$[a, b, d], [a, b, c], [a, c, e], [a, c, d, f]$
1	$[a, b, d], [a, b, c], [a, c, e], [a, c, d, f], \boldsymbol{[a, b, c], [a, d, e]}$

size of node v becomes $\lceil c \cdot |N_{out}(v)|/\alpha \rceil$ (lines 4–5). As a result, the time complexity is $O(1/\alpha)$ because the number of paths to be deleted is about $1/\alpha$.

The critical point is to delete the index in order from the earliest to the latest (line 5). The reason is that the paths generated in the past are more likely to be stale. Note that a newer index may be more stale than an older one if the newer index frequently passes the updated region of the graph. However, it is difficult to determine which index is more stale with light overhead. Therefore, the approach of preferentially deleting the older index, which is more likely to be stale, is reasonable. The experimental results show the effect of the index reference order on accuracy in Section 6.5.6.

An example of the index correction using the sample graph shown in Figure 6.2 will be presented below. Assume that edge (a, c) is deleted like in Figure 6.3, and the index of node a before the deletion is the paths shown in the row with timestamp 1 in Table 6.3. In this case, the number of paths maintained in Idx^1 is 6. After the edge insertion, the required number of paths changes to 4 because the degree of node a becomes 2. Therefore, the

Algorithm 7 Index Correction for an Edge Deletion

Input: Graph $G^t(V^t, E^t)$, deleted edge (v, w), index Idx^{t-1}
Output: Index Idx^t
1: $Idx^t \leftarrow Idx^{t-1}$;
2: $\omega_v \leftarrow \lceil c \cdot |N_{out}(v)|/\alpha \rceil$;
3: $\omega'_v \leftarrow Idx^t[v].size() - \omega_v$;
4: **for** $i = 1$ **to** ω'_v **do**
5: $\quad Idx^t[v].pop\_front()$;
6: **end for**

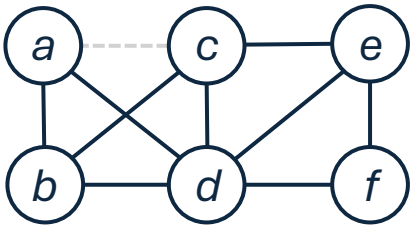


Figure 6.3: The sample graph that edge (a, c) is deleted.

algorithm deletes two indexed paths from the tail of Idx^1, as shown in the row with timestamp 2 in Table 6.3.

6.4.4 PPR Computation

The PPR computation algorithm is based on FORA+. In the Monte Carlo phase, it requires the indexed paths. αFlexWalk, presented in Chapter 5, returns the paths. The key point is that αFlexWalk references paths in order of generation time, from the latest to the earliest, considering that the proposed method uses the stale index as it is. Here, the proposed method assumes that the staleness increases monotonically with the number of edge updates. This means that paths added to the index that have been more recently added are

Table 6.3: Sample Indexed Paths for node a before and after an edge deletion in the graph shown in Figure 6.3 ($\alpha = 0.5$).

Timestamp	Indexed paths
1	$[a, b, d], [a, b, c], [a, c, e], [a, c, d, f], [a, b, c], [a, d, e]$
2	$[a, c, e], [a, c, d, f], [a, b, c], [a, d, e]$

supposed to be less stale. For example, assuming the case that the indexed paths at the queried time are $[a, c, e], [a, c, d, f], [a, b, c], [a, d, e]$ as shown in the row with timestamp 2 in Table 6.3, and two paths are referenced by αFlexWalk. In this case, $[a, b, c], [a, d, e]$, which are the latest paths, are referenced.

From the viewpoint of the algorithm correctness and the performance guarantee compared to the guaranteed methods based on FORA+, only referencing the stale index leads to a loss of accuracy. First, the algorithm termination is obviously guaranteed, as is FORA+. Second, regarding the performance guarantee, the loss of accuracy is caused by the second term of Equation (2.3) in Section 2.4, where the index is referenced. In particular, the expected absolute error of the PPR value of node v with respect to node s, denoted by $err(s, v)$, is represented by Equation (6.2), where $\tilde{\pi}(w, v)$ is a PPR value computed by the stale index.

$$err(s, v) = |\sum_{w \in V} r(s, w) \cdot \{\pi(w, v) - \tilde{\pi}(w, v)\}| \tag{6.2}$$

Note that it is difficult to estimate the expected absolute error for the partial re-indexing with light overhead. Estimating the error requires heavy

processing, such as the reverse push in Agenda [11], which takes O(n) time complexity for each graph update, as described in Section 6.2.

6.5 Evaluation

6.5.1 Settings

Firm [12] and Agenda [11] are adopted as the state-of-the-art PPR computation methods for dynamic graphs. Regarding the parameter of Agenda, $r_b^{max} = 1/n$, following the original paper [11]. The parameters of FORA+ are the same among all evaluations: $\alpha = \alpha_{index} = 0.2$, $\omega = 10^5$. Note that due to the memory budget, α_{index} is 0.3 for the Friendster dataset.

6.5.2 The Index Size

This evaluation compares the index size among the three methods to show the light space overhead of the proposed method. Two existing methods maintain additional indexes other than the random walk paths. Firm maintains a map to identify paths that pass through each node and auxiliary data for operations on the value of the map. Agenda maintains an inverse graph for the reverse push algorithm. I report the total memory size of the indexes for each method. Note that in large datasets, the index size is more significant than the memory budget. Therefore, the index size is estimated from the total number of indexed edges.

Figure 6.4 shows the index size for each method. According to Figure 6.4, the proposed method maintains the minimum index among the three methods. For the most significant dataset, Friendster, it is about 1 TB. The index size of Agenda is about the same as the proposed method, but it is slightly more significant due to the inverse graph for the reverse push algorithm.

On the other hand, the index size of Firm is from 3.36 to 3.65 times larger than that of the proposed method. For large datasets such as Twitter and Friendster, Firm's index is prohibitively significant. In particular, it is 2.0 TB and 4.8 TB, compared to 539 GB and 1.3 TB for the proposed method, respectively. This result shows that Firm has prohibitive memory overhead to achieve O(1) index corrections.

6.5.3 The Index Correction Time

This evaluation measures the index correction time of the proposed and existing methods. It assumes the two types of graph updates: edge insertions and deletions. In particular, the index is generated on the graph G^0. The two types of G^0 are considered to evaluate edge insertions and deletions. For edge insertions, G^0 has $(m - 1000)$ edges, where m is the number of edges a dataset contains. On the other hand, for edge deletions, G^0 has m edges. Afterward, the remaining 1000 edges are inserted or deleted one by one. For each update, the index correction time of each method is measured. Here, the index correction time of the existing methods is the sum of the time for

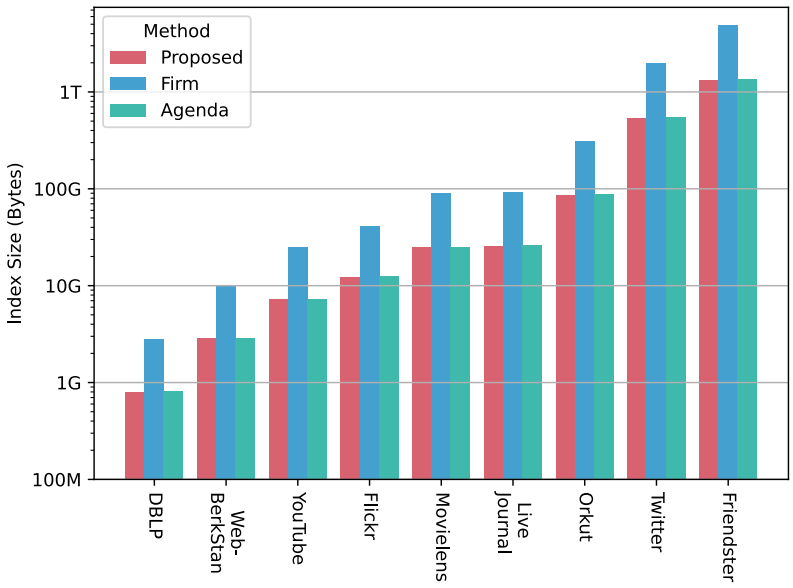


Figure 6.4: The index size of the proposed method and the existing methods.

index resizing and re-indexing, and that of the proposed method is only the time for index resizing. In Agenda, re-indexing is performed when updating an edge and processing the PPR query. However, only the time of the former will be reported because the computation cost of the re-indexing performed at the PPR query differs depending on the query. As a result, the actual index correction time of the Agenda is greater than the reported one.

Figure 6.5 shows the distribution of the index correction time. Figure 6.5a and Figure 6.5b give the correction time for edge insertions and deletions, respectively. Note that the Firm's correction times for Twitter and Friendster are omitted because their index size is more significant than the memory budget.

According to Figure 6.5, the index correction time of the proposed method for datasets other than the MovieLens is almost constant regardless of the updated edges. In datasets other than the MovieLens, all index correction times of the proposed method are within $[10^{-6}, 10^{-2}]$ seconds. These fast index corrections result from the proposed approach of omitting re-indexing.

The reason why results for the MovieLens show a relatively higher time is that it is a weighted dataset. A weighted graph maintains a data structure called an alias table to decide the next node of random walks in O(1) time complexity. When the weighted graph is updated, updating the alias table is required in addition to updating the adjacent list. As a result, updating the edge for a weighted graph involves an overhead.

On the other hand, Agenda's index correction times are within $[10^{-3}, 10^{3}]$ seconds, which means that the index correction times vary widely depending on a dataset and an updated edge. The median index correction time is $[10^{2}, 10^{6}]$ times slower than the proposed method. This indicates that Agenda's re-indexing, which performs the reverse push with $O(n)$ time complexity, is too heavy.

Furthermore, the Firm's index correction times are within $[10^{-5}, 10^{-1}]$ seconds and are nearly constant across all datasets and updated edges. However, the median index correction time is $[10^{1.5}, 10^{2.4}]$ times slower than the proposed method. The reason is that the Firm performs re-indexing while the proposed method does not.

Finally, the median index correction time of the proposed method for edge deletions is slightly faster than that for edge insertions. The reason is that the proposed method performs random walks when an edge is inserted, whereas it simply deletes the elements of the index from the deque when an edge is deleted.

6.5.4 Updated Ratio of Edges vs. Accuracy

This evaluation measures the accuracy loss against the number of edge updates to indicate that the proposed method achieves comparable accuracy even without re-indexing. The number of edge updates is quantified by an inserted/deleted ratio of edges. The inserted/deleted ratio of edges is defined as the ratio of the number of edges inserted/deleted between the index generation and the queried time to the number of edges the dataset contains. Therefore, it is $\frac{|m_q - m_0|}{m} \times 100$ (%), where m_0 and m_q are the number of edges at the index generation and queried times, respectively. Specifically, m_0 is $m/2$ and m in the evaluation of edge insertions and deletions, respectively. Then, $m/20$ edges of the remaining $m/2$ edges are inserted/deleted at a time. The total number of updates is $m/2$, i.e., the inserted/deleted ratio of edges is changed to $0, 5, 10, ..., 50\%$. Note that when the inserted/deleted ratio of edges is 50%, the number of edges at query time is doubled/halved from the index generation. The order of edge updates is randomized if the dataset does not contain timestamps. Otherwise, this evaluation follows the timestamp. The queries are Top-128 PPR with respect to randomly selected 100

nodes. NDCG, described in Section 4.5.1, quantifies accuracy.

Figure 6.6 shows the relationships between the inserted/deleted ratio of edges and NDCG. The x-axis represents the inserted/deleted ratio of edges at the query time in Figure 6.6a and Figure 6.6b, respectively. The y-axis represents the average NDCG for 100 source nodes.

According to Figure 6.6, the average NDCG is more than 0.999, while the inserted and deleted ratio of edges is less than 20% for all datasets. This result means that to maintain more than 0.999 NDCG, periodic index re-generation is required by the edge updated ratio reaches 20%. If users accept a lower accuracy, the index can be used for a longer time. These results also indicate that the re-indexing of the guaranteed methods, with significant overhead, contributes only a slight improvement in accuracy.

Figure 6.6a shows that NDCG is more than 0.9985, even when the inserted ratio of edges is 50%, i.e., the number of edges is doubled from the index generation. In all datasets, the decrease in NDCG against the inserted ratio of edges is slight. The relatively notable decrease in NDCG for the cases of 0% and 50% inserted ratio of edges is 0.005 and 0.004 for MovieLens and YouTube, respectively. In other datasets, the decreases are also less than 0.003, indicating the effectiveness of the proposed method.

In the case of edge deletions, Figure 6.6b shows almost the same results as the edge insertions, except for the result on YouTube. For YouTube, the decrease in NDCG is about 0.0027. The reason for the relatively large

decrease in NDCG for edge deletions than for edge insertions is that some indexed random walks are performed on the updated graph in the case of edge insertions. On the other hand, in the case of edge deletions, all indexed random walks are performed on the initial graph, which accounts for the difference in NDCG.

Moreover, considering Figure 6.1 and Figure 6.6, it is suggested that index references to nodes with particularly low *stability*, such as those below 0.8, cause a loss of accuracy. According to Figure 6.1, the index references to the nodes whose *stability* is less than 0.8 are 2.6% in YouTube. On the other hand, that for other datasets, including randomized YouTube, is less than 1%. Therefore, this tendency may result in YouTube's relatively lower accuracy compared to other datasets.

For a more detailed analysis, I investigate the performance of YouTube. As discussed in Section 6.3, accuracy is expected to be determined by the *stability* of the nodes to which the index is referenced. Therefore, I will clarify the characteristics of the nodes with low *stability* on YouTube. In particular, I focus on the relationships between the degree and the *stability*. Note that the degree is the feature of a node that is highly relevant to the other complex features and is easy to obtain. When computing the *stability*, I set the graph before and after updates as the one with $m/2$ and m edges, respectively. For these graphs, I randomly sample 1000 nodes and measure *stability* and degree in the graph with m edges for all sampled nodes.

Figure 6.7 shows the relationships between degree and *stability*. Ac-

cording to Figure 6.7, I find that most of the nodes with low *stability* (less than 0.8) have a degree of less than 50. Therefore, the relatively low accuracy of YouTube is expected to be caused by the index staleness of the nodes with a low degree. However, it is also observed that nodes with low degrees do not necessarily exhibit low *stability*. The same trend is observed in the other datasets, although the decrease in *stability* is not as significant as in YouTube. To estimate the staleness, heavy computation during the index correction, such as Agenda's reverse push, is required. From these observations, it is challenging to realize partial re-indexing focusing on the nodes expected to be low *stability*, avoiding heavy computations.

6.5.5 Effect of the Timestamp

To confirm the effect of the timestamp, I compare accuracy when edges are updated in timestamped order with that in randomized order. The evaluation method other than updating order is the same as the evaluation in Section 6.5.4. Figure 6.8 shows the results. In Figure 6.8, the solid and dashed lines represent the results for the randomized and timestamped updates, respectively.

According to Figure 6.8, the edge update order does not make a difference in Flickr. In contrast, accuracy improves by up to 0.0005 and 0.0025 by randomizing the edge insertions and deletions on YouTube, respectively. Similarly, in the case of MovieLens, accuracy improves by up to 0.0005 and 0.0008 when the edge updates are insertions and deletions, respectively. From

these results, the locality of the edge updates may degrade accuracy.

6.5.6 Effect of the Index Reference Order

I check the effect of the index reference order on accuracy. I compare the two types of referencing orders: from the latest to the earliest and from the earliest to the latest. The method is the same as the one described in Section 6.5.4. When computing Top-128 PPR, I calculate NDCG for each referencing pattern. Note that edge deletions are not considered in this evaluation. When an edge is deleted, the proposed method does not perform any random walks. Therefore, all indexed random walks are performed on the initial graph. As a result, accuracy is not affected by the index reference order.

Figure 6.9 shows the effect of the index reference order. The x-axis represents the inserted ratio of edges at query time, and the y-axis represents the average difference between NDCGs when the index is referenced in the order of the latest to the earliest and the earliest to the latest. Thus, a more significant value on the y-axis indicates that the proposed referencing order improves NDCG.

According to Figure 6.9, NDCG is improved by referencing the index in the inverse order of generation time on all datasets because all values on the y-axis are more than 0. In particular, the improvement in NDCG is up to 0.01 on YouTube, and the effect becomes significant as the ratio of inserted

edges increases. As described in Section 6.5.4, YouTube shows relatively low accuracy. Therefore, it is suggested that the benefit of the index reference order becomes significant for the dataset sensitive to the edge updates.

On the other hand, the improvement is negligible in some datasets, such as Movielens, Twitter, and Friendster. The difference in NDCG is less than 0.0006 even when the ratio of inserted edges is 50% in these datasets. In these datasets, the fact that Top-k PPR is unlikely to change on index-referenced nodes contributes more to improving accuracy than the index reference order. However, as described in Section 6.4, the proposed way of index reference does not involve any computational overhead. Therefore, although the proposed method is generally robust to reference orders, referencing the index in the inverse order, as proposed, can help achieve further prevention of loss of accuracy.

6.5.7 Processing Time of The Proposed Method vs. Index-Free Method

To show the advantage of using the stale index, I compare the processing time of the proposed method with that of the index-free method, setting ω so that both methods achieve the same accuracy. Although the proposed method is fast due to the index, accuracy decreases by the stale index. Considering that ω is a parameter that balances the processing time and accuracy, a smaller ω is enough in the index-free method to achieve the same accuracy as the proposed method.

This evaluation compares the proposed method's processing time with that of the index-free method, whose input ω is smaller than the proposed method. In particular, I confirm ω_{fora+} and ω_{fora} enough to achieve 0.999 NDCG with the proposed method and the index-free FORA, respectively. Afterward, the processing time of the proposed method and FORA are measured while setting ω to ω_{fora+} and ω_{fora}, respectively. The queried graph is the one whose inserted ratio of edges is 20%. Note that Section 6.5.4 clarified that 0.999 NDCG could be achieved on the graph whose inserted ratio of edges is 20%.

Figure 6.10 shows the PPR processing time of the index-free FORA divided by that of the proposed method for each dataset. A value on the y-axis larger than 1 indicates that the proposed method is faster than the index-free method even though the index-free method sets a smaller ω. The box-and-whisker plot represents the processing time ratio with respect to 100 source nodes on each dataset. As a result, the proposed is faster than the index-free method for more than about 60% of source nodes in datasets other than DBLP and Web-BerkStan. Therefore, using the stale index is justified to speed up the PPR computation for most source nodes.

The reason that the results of DBLP and Web-BerkStan show relatively minor values is that they are small datasets. The major bottleneck of the random walks is the random access to the memory. However, their graph size is tens to a hundred MB, and most of the graph data can be handled in the cache memory. As a result, the merit of using the index in these small

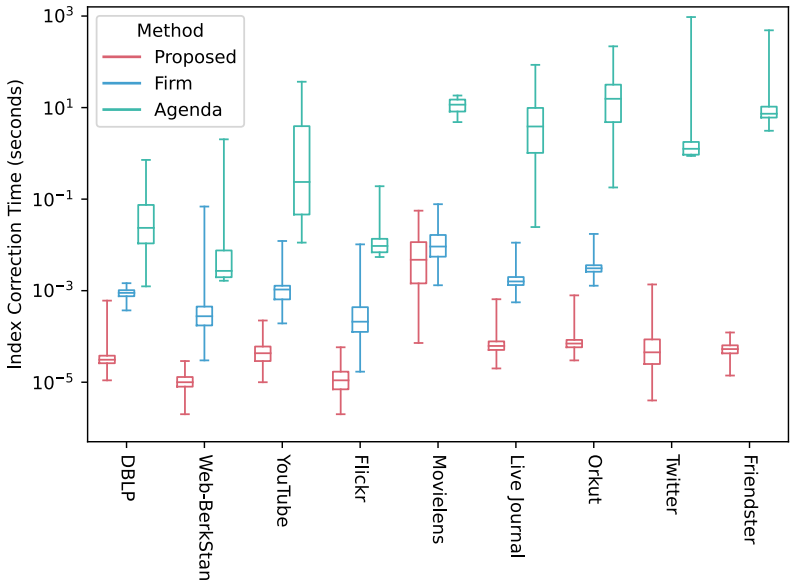
datasets is not significant.

Moreover, for some source nodes, the index-free method is faster than the proposed method. A detailed analysis of the characteristics of these nodes shows that they can be explained by one of the following two reasons: First, the source node is located in a small disconnected component of the graph and can achieve sufficient NDCG with a tiny number of random walks. The second case is when graph updates are concentrated around the source node, and the NDCG of the proposed method decreases relatively significantly. For these reasons, it is clear that index-free can be advantageous for a limited number of nodes.

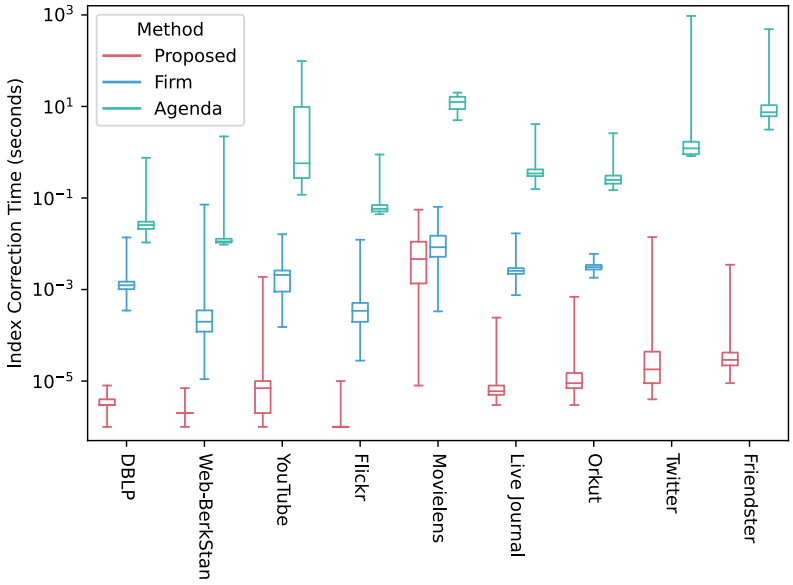
6.6 Conclusion of This Chapter

Re-indexing for each edge update, performed to guarantee accuracy, is a bottleneck for Top-k Personalized PageRank (PPR) computations on dynamic graphs. This chapter focuses on index references, concentrating on nodes whose index is unlikely to change due to edge updates. The proposed method omits re-indexing as long as comparable accuracy to state-of-the-art methods is achieved. While comparable accuracy is achieved, the proposed method requires minimum index correction time and memory overhead compared to the existing methods. The evaluations using nine real-world datasets show the effectiveness of the proposed method and the time period in which the proposed method achieves comparable accuracy. In particular, by quan-

tifying accuracy with Normalized Discounted Cumulative Gain (NDCG) of Top-128 PPR, the proposed method achieves more than 0.999 average NDCG until 20% of edges are updated.

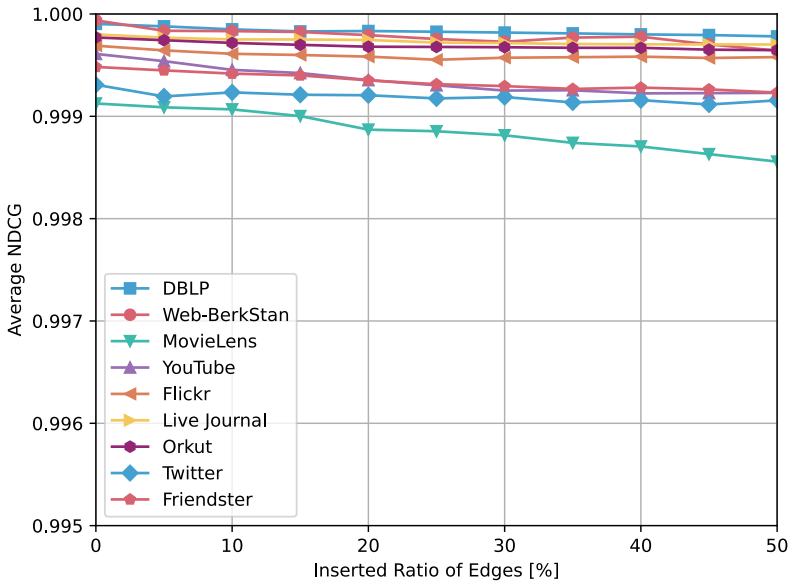


(a) Edge insertions.

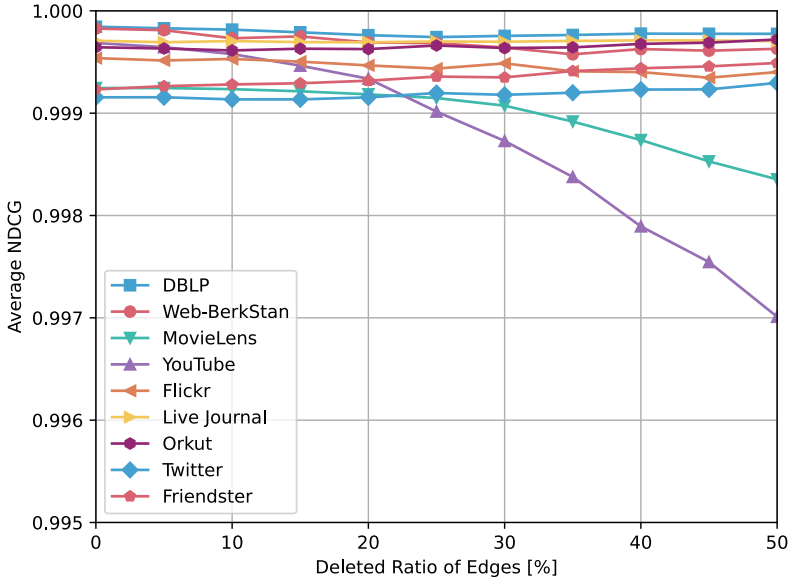


(b) Edge deletions.

Figure 6.5: The distribution of index correction time.



(a) Edge insertions.



(b) Edge deletions.

Figure 6.6: The updated ratio of edges vs. NDCG.

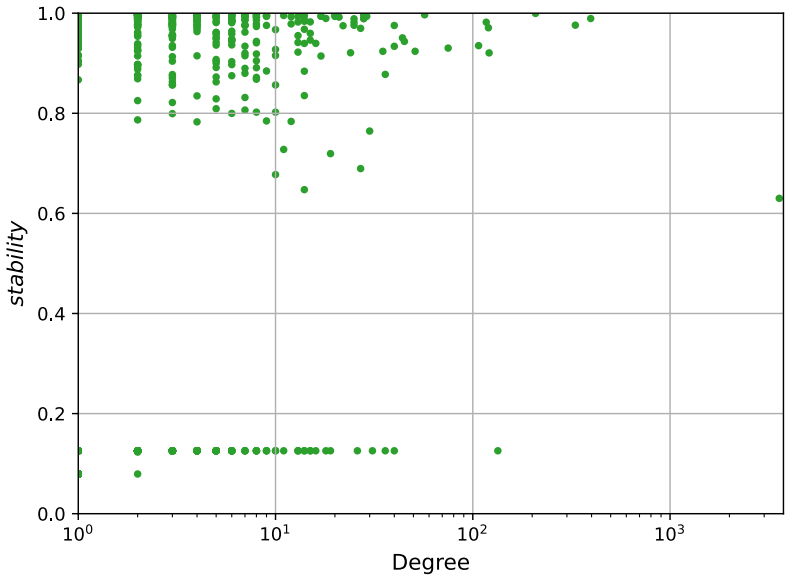
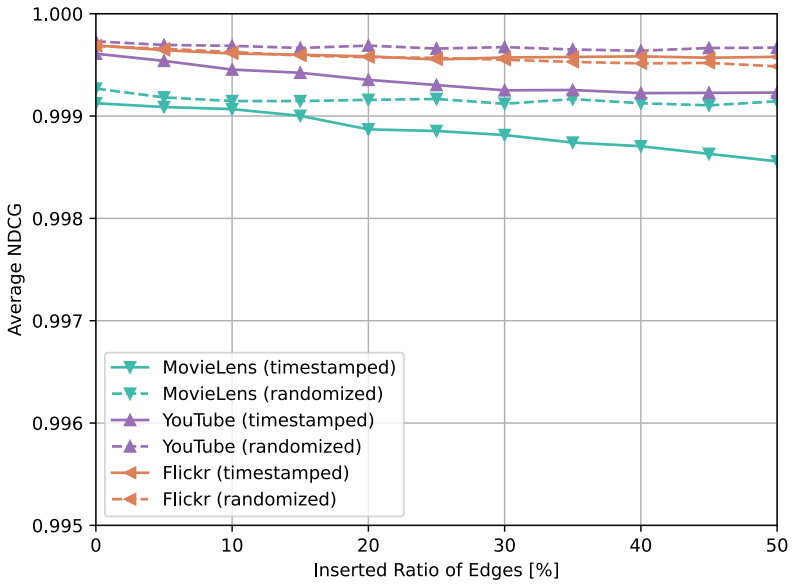
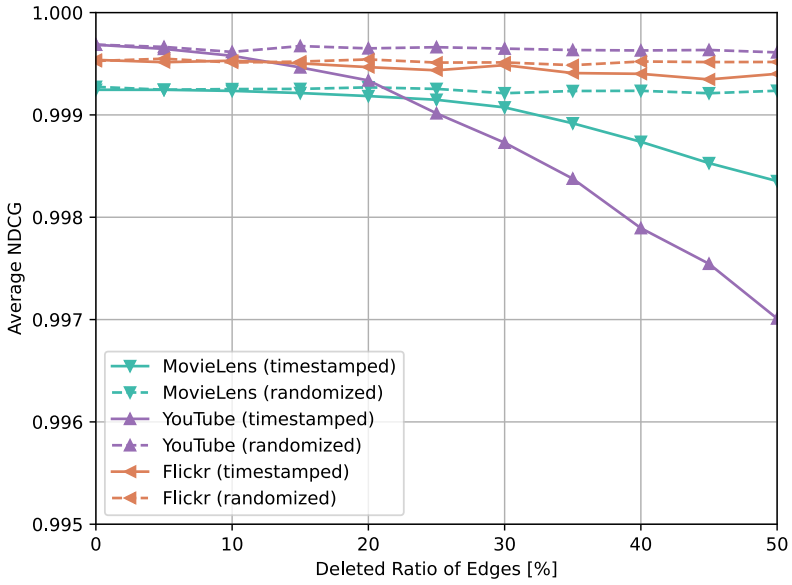


Figure 6.7: The relationships between degree and *stability* in YouTube.



(a) Edge insertions.



(b) Edge deletions.

Figure 6.8: The updated ratio of edges vs. NDCG when the graph is updated in the timestamped and randomized order.

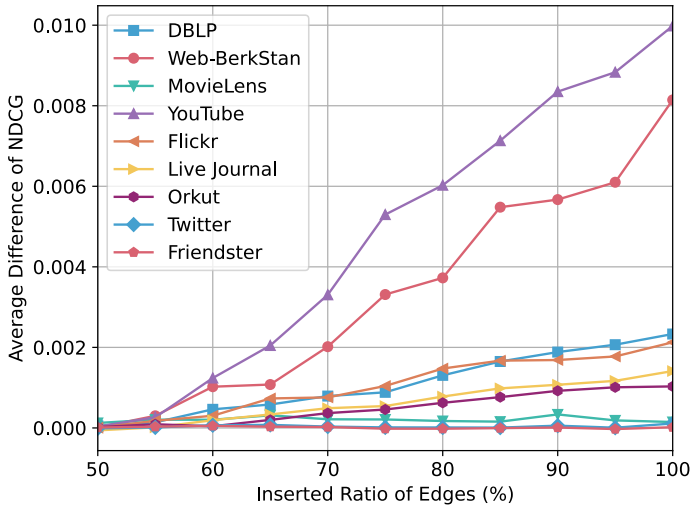


Figure 6.9: The average difference between NDCGs when the index is referenced in the order of the latest to the earliest and the earliest to the latest.

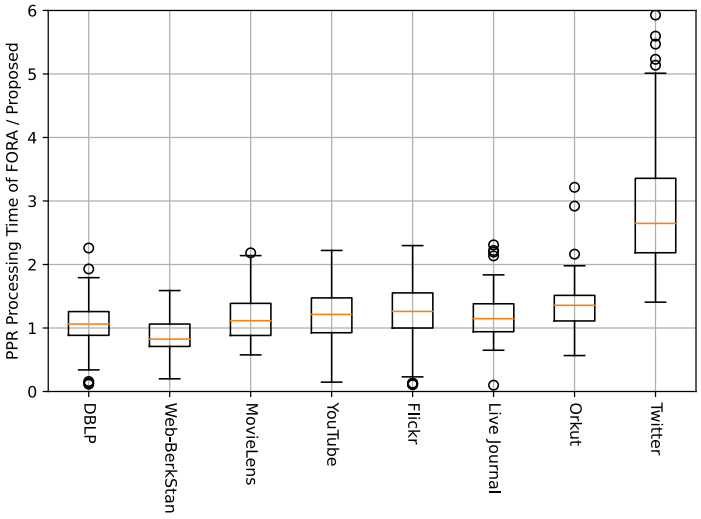


Figure 6.10: PPR processing time of index-free method divided by that of the proposed method.

Chapter 7

Conclusion

To utilize various types of digital data, personalized graph analysis gathers attention. In particular, Personalized PageRank (PPR), which uses random walks from the user's interest node, is effective for personalized graph analysis. However, PPR-based graph analysis has three problems. Firstly, there are no parameter-setting guidelines for the termination probability α to control the length of the random walks. This dissertation clarifies the influence of α on the personalized analysis results. In particular, it confirms that α monotonically balances the influences of global importance and source proximity on the PPR vector using real-world datasets. As a result, in the movie rating dataset, the movies directly related to the source node get a higher PPR value by shortening random walks. Moreover, evaluations show that the cosine similarity between PPR and PageRank vectors, which represents global importance, changes monotonically from 0.003 to 0.76 at the maximum by changing the expected random walk length from 1.05 to 100.

Secondly, although random walks using an index are helpful for fast PPR computation, existing index-based random walk methods only accept a specific α. This dissertation proposes an index-based algorithm αFlexWalk to generate random walk paths for arbitrary α. In particular, it focuses on the fact that the random walk length differs probabilistically when α changes. αFlexWalk generates the guaranteed random walk paths by connecting and cutting the indexed paths. Evaluations show that αFlexWalk improves the processing time by up to 11.2 times compared with the existing index-free method. Thirdly, on dynamic graphs, re-indexing for each graph update is required to guarantee accuracy. However, re-indexing involves either significant time or memory overhead. This dissertation proposes a method that omits re-indexing while accuracy comparable to the guaranteed method is achieved. The proposed method is based on the observation that the index references concentrate on nodes whose index is stable to graph updates. The evaluations show that the proposed method's graph update time and index size are the best among the existing index-based methods. It also clarifies that the proposed method achieves 0.999 Normalized Discounted Cumulative Gain (NDCG) until 20% of edges are updated from the index generation.

I will present future work for each of Chapter 4 – Chapter 6. First, Chapter 4 balanced the influence of the global importance vector on the PPR vector according to the value of α. However, the control range differed among datasets, and the influence of the global importance vector could not be predicted until the computation was actually performed. Therefore, it is

considered necessary to control the influence from a theoretical perspective, as well as the statistical investigation conducted in this dissertation.

Next, Chapter 5 clarifies the algorithmic contribution of the proposed method. However, as mentioned in Section 5.2, existing random walk acceleration methods use computational techniques to remove bottlenecks in each execution environment. Since these computational techniques can be applied at the same time as the proposed algorithmic technique, it is considered necessary to clarify the effect of the proposed method in such cases.

Finally, Chapter 6 proposed a method for continuing to use stale index as long as comparable accuracy is achieved. I have considered the case where all indexes are regenerated when graph updates accumulate and sufficient accuracy is no longer achieved. However, in real-world situations, it is possible to partially regenerate indexes at times when the query load is low. Therefore, there is room to consider partially re-performing random walks that have passed a certain amount of time since they were generated to replace indexes.

References

[1] David Reinsel, John Gantz, and John Rydning. The digitization of the world - From edge to core. Technical report, May 2020.

[2] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. WTF: The Who to Follow Service at Twitter. In *Proceedings of the International Conference on World Wide Web*, pages 505–514, Rio de Janeiro, RJ,Brazil, May 2013.

[3] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time. In *Proceedings of the International Conference on World Wide Web*, pages 1775–1784, Lyon, France, April 2018.

[4] David C. Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. Related pins at Pinterest: The evolution of a real-world recommender system. In *Proceedings of the International Conference on World Wide Web Companion*, pages 583–592, Perth, WA, Australia, April 2017.

[5] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the International Conference on World Wide Web*, pages 271–279, Budapest, Hungary, 2003.

[6] Taher H. Haveliwala. Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796, July 2003.

[7] Yuying Zhao, Yu Wang, Yunchao Liu, Xueqi Cheng, Charu C. Aggarwal, and Tyler Derr. Fairness and Diversity in Recommender Systems: A Survey. *ACM Transactions on Intelligent Systems and Technology*, 2024.

[8] Yifan Wang, Weizhi Ma, Min Zhang, Yiqun Liu, and Shaoping Ma. A Survey on the Fairness of Recommender Systems. *ACM Transactions on Information Systems*, 41(3):1–43, February 2023.

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

[10] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 505–514, Halifax, NS, Canada, August 2017.

[11] Dingheng Mo and Siqiang Luo. Agenda: Robust Personalized PageRanks in Evolving Graphs. In *Proceedings of the ACM International Con-*

ference on Information and Knowledge Management, pages 1315–1324, Online and Queensland, Australia, October 2021.

[12] Guanhao Hou, Qintian Guo, Fangyuan Zhang, Sibo Wang, and Zhewei Wei. Personalized PageRank on Evolving Graphs with an Incremental Index-Update Scheme. *Proceedings of the ACM International Conference on Management of Data*, 1(1):1–26, May 2023.

[13] Zulun Zhu, Siqiang Luo, Wenqing Lin, Sibo Wang, Dingheng Mo, and Chunbo Li. Personalized PageRanks over dynamic graphs - The case for optimizing quality of service. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 409–422, Utrecht, Netherlands, May 2024.

[14] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized PageRank. *VLDB Endowment*, 4(3):173–184, December 2010.

[15] Naoto Ohsaka, Takanori Maehara, and Ken Ichi Kawarabayashi. Efficient PageRank Tracking in Evolving Networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 875–884, Sydney, NSW, Australia, August 2015.

[16] Minji Yoon, Woojeong Jin, and U Kang. Fast and Accurate Random Walk with Restart on Dynamic Graphs with Guarantees. In *Proceedings of the World Wide Web Conference*, pages 409–418, Lyon, France, April 2018.

[17] Zexing Zhan, Ruimin Hu, Xiyue Gao, and Nian Huai. Fast Incremental PageRank on Dynamic Networks. In *Proceedings of the 19th International Conference on Web Engineering*, pages 154–168, Daejeon, South Korea, June 2019.

[18] Tao Guo, Xin Cao, Gao Cong, Jiaheng Lu, and Xuemin Lin. Distributed Algorithms on Exact Personalized PageRank. In *Proceedings of the ACM International Conference on Management of Data*, pages 479–494, Chicago, IL, USA, May 2017.

[19] Hongyang Zhang, Peter Lofgren, and Ashish Goel. Approximate Personalized PageRank on Dynamic Graphs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1315–1324, San Francisco, CA, USA, August 2016.

[20] Zihao Li, Dongqi Fu, and Jingrui He. Everything Evolves in Personalized PageRank. In *Proceedings of the World Wide Web Conference*, pages 3342–3352, Austin, TX, USA, May 2023.

[21] Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. Efficient Algorithms for Approximate Single-Source Personalized PageRank Queries. *ACM Transactions on Database Systems*, 44(4):1–37, December 2019.

[22] Alastair J. Walker. An Efficient Method for Generating Discrete Random Variables with General Distributions. *ACM Transactions on Mathematical Software*, 3(3):253–256, September 1977.

[23] David F Gleich, Paul G Constantine, and Abraham D Flaxman. Tracking the Random Surfer : Empirically Measured Teleportation Parameters in PageRank. In *Proceedings of the International Conference on World Wide Web*, pages 381–390, Raleigh, NC, USA, April 2010.

[24] Konstantin Avrachenkov, Nelly Litvak, and Kim Son Pham. A Singular Perturbation Approach for Choosing the PageRank Damping Factor. *Internet Mathematics*, 5(1):47–69, 2008.

[25] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments. *Internet Mathematics*, 2(3):333–358, January 2005.

[26] Qin Liu, Zhenguo Li, John C.S. Lui, and Jiefeng Cheng. PowerWalk: Scalable Personalized PageRank via Random Walks with Vertex-Centric Decomposition. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 195–204, Indianapolis, IN, USA, October 2016.

[27] Wenqing Lin. Distributed Algorithms for Fully Personalized PageRank on Large Graphs. In *Proceedings of the World Wide Web Conference*, pages 1084–1094, San Francisco, CA, USA, May 2019.

[28] Jinhong Jung, Lee Sael, Namyong Park, and U. Kang. BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart. In *Proceedings of the ACM International Conference on Management of Data*, pages 789–804, Chicago, IL, USA, May 2017.

[29] Tao Guo, Xin Cao, Gao Cong, Jiaheng Lu, and Xuemin Lin. Distributed Algorithms on Exact Personalized PageRank. In *Proceedings of the ACM International Conference on Management of Data*, pages 479–494, Chicago, IL, USA, May 2017.

[30] Takanori Maehara, Takuya Akiba, Yoichi Iwata, and Kenichi Kawarabayashi. Computing Personalized PageRank Quickly by Exploiting Graph Structures. *VLDB Endowment*, 7(12):1023–1034, August 2014.

[31] Siqiang Luo, Xiaokui Xiao, Wenqing Lin, and Ben Kao. BATON: Batch One-Hop Personalized PageRanks with Efficiency and Accuracy. *IEEE Transactions on Knowledge and Data Engineering*, 32(10):1897–1908, April 2019.

[32] Guanhao Hou, Xingguang Chen, Sibo Wang, and Zhewei Wei. Massively Parallel Algorithms for Personalized PageRank. *VLDB Endowment*, 14(9):1668–1680, May 2021.

[33] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. In *Proceedings of the ACM International Conference on Management of Data*, pages 1996–2008, Online and Shaanxi, China, June 2021.

[34] Peter A. Lofgren, Siddhartha Banerjee, Ashish Goel, and C. Seshadhri. FAST-PPR: Scaling Personalized PageRank Estimation for Large Graphs. In *Proceedings of the ACM SIGKDD International Con-*

ference on Knowledge Discovery and Data Mining, pages 1436–1445, New York, NY, USA, August 2014.

[35] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 163–172, San Francisco, CA, USA, February 2016.

[36] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. HubPPR: Effective Indexing for Approximate Personalized PageRank. *VLDB Endowment*, 10(3):205–216, November 2016.

[37] Reid Andersen, Fan Chung, and Kevin Lang. Local Graph Partitioning Using PageRank Vectors. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 475–483, Berkeley, CA, USA, October 2006.

[38] Jure Leskovec and Andrej Krevl. Stanford large network dataset collection.

[39] Alan Mislove. *Online social networks: Measurement, analysis, and applications to distributed information systems*. PhD thesis, Rice University, Department of Computer Science, 2009.

[40] Alan Mislove, Hema Swetha Koppula, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Growth of the Flickr Social Network. In *Proceedings of the Workshop on Online Social Networks*, pages 25–30, Seattle, WA, USA, August 2008.

[41] F. Maxwell Harper and Joseph A. Konstan. The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):1–19, December 2015.

[42] Jérôme Kunegis. KONECT: The Koblenz network collection. In *Proceedings of the International Conference on World Wide Web*, pages 1343–1350, Rio de Janeiro, Brazil, May 2013.

[43] Makoto Nakatsuji, Yasuhiro Fujiwara, Akimichi Tanaka, Toshio Uchiyama, Ko Fujimura, and Toru Ishida. Classical music for rock fans?: Novel recommendations for expanding user interests. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 949–958, Toronto, ON, Canada, October 2010.

[44] Kensuke Onuma, Hanghang Tong, and Christos Faloutsos. Tangent: A novel, "Surprise-me", recommendation algorithm. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 657–665, Paris, France, June 2009.

[45] André Levi Zanon, Leonardo Chaves Dutra da Rocha, and Marcelo Garcia Manzato. Balancing the trade-off between accuracy and diversity in recommender systems with personalized explanations based on Linked Open Data. *Knowledge-Based Systems*, 252:109333, September 2022.

[46] Zihong Wang, Yingxia Shao, Jiyuan He, Jinbao Liu, Shitao Xiao, Tao Feng, and Ming Liu. Diversity-aware Deep Ranking Network for Recommendation. In *Proceedings of the ACM International Conference on*

Information and Knowledge Management, pages 2564–2573, Birmingham, United Kingdom, October 2023.

[47] Marius Kaminskas and Derek Bridge. Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-Accuracy objectives in recommender systems. *ACM Transactions on Interactive Intelligent Systems*, 7(1):1–42, December 2016.

[48] Yifei Zhou and Conor Hayes. Graph-Based Diffusion Method for Top-N Recommendation. In *Artificial Intelligence and Cognitive Science*, pages 292–304, Munster, Ireland, December 2023.

[49] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. VERSE: Versatile graph embeddings from similarity measures. In *Proceedings of the World Wide Web Conference*, pages 539–548, Lyon, France, April 2018.

[50] Flavio Chierichetti, Anirban Dasgupta, Ravi Kumar, Silvio Lattanzi, and Tamás Sarlós. On Sampling Nodes in a Network. In *Proceedings of the International Conference on World Wide Web*, pages 471–481, 2016.

[51] Kotaro Fujii, Tsuyoshi Yamashita, Andrew Shin, and Kunitake Kaneko. A Flexible Weighting Framework for Converting Relational Database to Hypergraphs. In *Proceeding of the International Conference on Information Networking*, Chiang Mai, Thailand, January 2025.

[52] Mingji Yang, Hanzhi Wang, Zhewei Wei, Sibo Wang, and Ji-rong Wen. Efficient Algorithms for Personalized PageRank Computation: A Sur-

vey. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, March 2024.

[53] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to Rank Using Gradient Descent. In *Proceedings of the International Conference on Machine Learning*, pages 89–96, Bonn, Germany, August 2005.

[54] N Litvak, W R W Scheinhardt, and Y Volkovich. In-Degree and PageRank: Why Do They Follow Similar Power Laws? *Internet Mathematics*, 4(2-3):175–198, 2007.

[55] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet Personalized PageRank. In *Proceedings of the International Conference on Learning Representations*, pages 1–15, New Orleans, LA, USA, May 2019.

[56] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling Graph Neural Networks with Approximate PageRank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2464–2473, Online and CA, USA, August 2020.

[57] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji Rong Wen. Scalable graph neural networks via bidirec-

tional propagation. *Advances in neural information processing systems*, 33:14556–14566, 2020.

[58] Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, and Ji Rong Wen. Approximate Graph Propagation. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1686–1696, Online and Singapore, August 2021.

[59] Xingyi Zhang, Shuliang Xu, Wenqing Lin, and Sibo Wang. Constrained Social Community Recommendation. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 5586–5596, Long Beach, CA, USA, August 2023.

[60] Qiuchen Zhang, Jing Ma, Jian Lou, Carl Yang, and Li Xiong. Towards Training Graph Neural Networks with Node-Level Differential Privacy. In *Proceedings of the World Wide Web Conference*, volume 1, Singapore, Singapore, May 2024.

[61] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710, New York, NY, USA, August 2014.

[62] Qiuchen Zhang, Jing Ma, Jian Lou, Carl Yang, and Li Xiong. Towards Training Graph Neural Networks with Node-Level Differential Privacy. In *Proceedings of the ACM Web Conference*, volume 1, Singapore, Singapore, May 2024.

[63] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S. Bhowmick. Homogeneous network embedding for massive graphs via reweighted personalized pagerank. *VLDB Endowment*, 13(5):670–683, January 2020.

[64] Konstantin Avrachenkov, Nelly Litvak, and Kim Son Pham. Distribution of PageRank mass among principle components of the Web. In *Proceedings of the Workshop On Algorithms And Models For The Web-Graph*, pages 16–28, San Diego, CA, USA, 2007.

[65] Hui Zhang, Ashish Goel, Ramesh Govindan, Kahn Mason, and Benjamin Van Roy. Making Eigenvector-Based Reputation Systems Robust to Collusion. In *Proceedings of the International Workshop on Algorithms and Models for the Web-Graph*, pages 92–104, Rome, Italy, October 2004.

[66] Ke Yang, MingXing Zhang, Kang Chen, Xiaosong Ma, Yang Bai, and Yong Jiang. KnightKing: A Fast Distributed Graph RandomWalk Engine. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 524–537, Huntsville, ON, Canada, October 2019.

[67] Rui Wang, Yongkun Li, John C S Lui, Hong Xie, and Yinlong Xu. GraphWalker: An I/O-efficient and resource-friendly graph analytic system for fast and scalable random walks. In *Proceedings of the USENIX Annual Technical Conference*, pages 559–571, Online, July 2020.

[68] Shuke Wang, Mingxing Zhang, Ke Yang, Kang Chen, Shaonan Ma, Jinlei Jiang, and Yongwei Wu. NosWalker: A Decoupled Architecture for Out-of-Core Random Walk Processing. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 466–482, Vancouver, BC, Canada, March 2023.

[69] Shixuan Sun, Yuhang Chen, Shengliang Lu, Bingsheng He, and Yuchen Li. ThunderRW: An In-Memory Graph Random Walk Engine. *VLDB Endowment*, 14(11):1992–2005, July 2021.

[70] Ke Yang, Xiaosong Ma, Saravanan Thirumuruganathan, Kang Chen, and Yongwei Wu. Random walks on huge graphs at cache efficiency. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 311–326, Online, 2021.

[71] Tsuyoshi Yamashita, Naoki Matsumoto, and Kunitake Kaneko. Reducing re-indexing for top-k Personalized PageRank computation on dynamic graphs. *IEEE Transactions on Big Data*, pages 1–13, 2025. to appear.

[72] Luc Devroye. *Non-uniform random variate generation*. Springer Science+Business Media, LLC, 1986.

[73] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S. Mirrokni, and Shang Hua Teng. Local computation of PageRank

contributions. In *Proceedings of the International Workshop on Algorithms and Models for the Web-Graph*, pages 150–165, San Diego, CA, USA, December 2007.