Efficient ad-hoc search for Personalized PageRank

Conference Paper · June 2013

DOI: 10.1145/2463676.2463717

Makoto Nakatsuji
Nippon Telegraph and Telephone
31 PUBLICATIONS **474** CITATIONS
SEE PROFILE

Makoto Onizuka
Osaka University
103 PUBLICATIONS **1,311** CITATIONS
SEE PROFILE

Efficient Ad-hoc Search for Personalized PageRank

Yasuhiro Fujiwara†, Makoto Nakatsuji‡, Hiroaki Shiokawa†,
Takeshi Mishima†, Makoto Onizuka†
†NTT Software Innovation Center, 3-9-11 Midori-cho Musashino-shi, Tokyo, Japan
‡NTT Service Evolution Laboratories, 1-1 Hikarinooka Yokosuka-shi, Kanagawa, Japan
{fujiwara.yasuhiro, nakatsuji.makoto, shiokawa.hiroaki,
mishima.takeshi, onizuka.makoto}@lab.ntt.co.jp

ABSTRACT

Personalized PageRank (PPR) has been successfully applied to various applications. In real applications, it is important to set PPR parameters in an ad-hoc manner when finding similar nodes because of dynamically changing nature of graphs. Through interactive actions, interactive similarity search supports users to enhance the efficacy of applications. Unfortunately, if the graph is large, interactive similarity search is infeasible due to its high computation cost. Previous PPR approaches cannot effectively handle interactive similarity search since they need precomputation or approximate computation of similarities. The goal of this paper is to efficiently find the top-k nodes with exact node ranking so as to effectively support interactive similarity search based on PPR. Our solution is *Castanet*. The key Castanet operations are (1) estimate upper/lower bounding similarities iteratively, and (2) prune unnecessary nodes dynamically to obtain top-k nodes in each iteration. Experiments show that our approach is much faster than existing approaches.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining

General Terms

Algorithms, Theory

Keywords

Graph database, Personalized PageRank, Top-k search

1. INTRODUCTION

Large scale, highly interconnected networks, such as social networks and communication networks, have pervaded our society [26, 27]. There is a growing need for data management systems that can support the searching and mining of such network data. Since network data is naturally represented as a graph, graph databases have attracted significant attention in the database community. In recent years, various papers have been published on reachability queries [21], parallel algorithms [22], and graph clustering [32].

In the database community, link-based similarity measures have become popular since the publication of the PageRank paper [29]. The objective of link-based similarity measures is to extract similarities among nodes from the link structures of the graph; link-based similarity measures are used so as to find, classify, and analyze objects on graph databases [5]. Since then, there has been a surge of papers on link-based similarity measures such as ObjectRank [5] and SimRank [15]. From among them, *Personalized PageRank (PPR)* [4, 12], also known as Random Walk with Restart (RWR)[1] [14], has emerged as the most popular link-based similarity measure due to its effectiveness and solid theoretical foundation. Informally, PPR scores correspond to the stationary probabilities of random walks. At each step of PPR, an outgoing edge from the current node is randomly selected, and, with a certain probability, c, the walk is randomly restarted from a given query node. The probability c is called a scaling parameter [23]. The objective of this work is to enhance the efficacy of the following applications by finding top-k nodes with exact node ranking in an ad-hoc manner where PPR is used as the similarity measure:

Web prefetching. Web prefetching is a technique to download web pages before users demand them to reduce user-perceived latencies; this demands predicting which pages will be accessed by the users [10]. If most prefetched web pages are not visited by users in their subsequent accesses, the prefetching technique cannot effectively reduce access delays. Therefore, the success of the prefetching technique heavily relies on the accuracy of predictions. The approach proposed by Guo et al. offers effective predictions based on PPR [17]. Their approach sets user-specific query nodes in an ad-hoc manner according to the user's web logs, such as access frequency, which are updated by browsing actions. Their approach provides the user with top-5 prediction lists based on the PPR scores of the web pages and yields 23% higher accuracy than usage-based PageRank approach [11].

Word sense disambiguation. Queries for search engines are typically short, consisting of just 2.2 words on average [6]. Search engines need to derive what users want from this limited source of information. However, many words have multiple meanings and the correct meaning depends on the context in which they are used. Word sense disambiguation is the task of identifying the correct meaning of words [28]; it is a key technique in improving a search en-

[1] RWR is a special case of PPR as described in Section 3.

gine's performance. The PPR-based approach proposed by Agirre et al. performs word sense disambiguation by exploiting WordNet[2] as a graph [1]. This approach performs the disambiguation for arbitrarily given words to match the users' intentions; it applies PPR over the graph and assigns the meaning with the highest rank to the query word. Their approach outperforms the previous TextRank approach [25].

Automatic image annotation. The explosion in the number of images on the web necessitates an effective tool to search them. The traditional approach to finding images, practiced by librarians, is to annotate each image manually with keywords and then search on those keywords using a search engine; thus queries can be naturally specified by the user. However, the disadvantage with this approach is the cost and infeasibility of scaling it to large numbers of images. Automatic image annotation is the process by which metadata, in the form of keywords, is automatically assigned to a digital image [31]. It is achieved by finding high ranking keywords to the query image from a given collection of images and associated keywords on the fly. Pan et al. proposed a graph-based annotation method in which images and keywords are treated as nodes in a mixed media graph [30]. They utilized RWR to obtain top-5 ranking keywords for each query image from the large image collection. Their method can provide 10 percent higher annotation accuracy than a fine-tuned method.

Interactive similarity search is an important process to enhance the effectiveness of the above applications; interactive similarity search finds similar nodes in an ad-hoc manner for an arbitrary graph with the scaling parameter. The reasons are twofold. First, the graph can be fully known only after the query is submitted [8]. For example, in web prefetching, graphs are generated on the fly from the user's web logs, and mixed media graphs are created in an ad-hoc manner based on the arbitrarily given image for automatic image annotation. Second, the scaling parameter c has an impact on similarity scores and effectiveness of the applications. For example, Pan et al. reported that, for automatic image annotation, high quality results were achieved with $c = 0.1$ or 0.2 [30] while $c = 0.85$ is recommended for word sense disambiguation [1]. This indicates that the users dynamically change the scaling parameter c through their interactive actions based on search results [8]. However, this interactive approach has difficulty in achieving real-time response if the graph is large. This is because the computation cost of PPR is $O((N + M)T)$ where N, M, and T are the numbers of nodes, edges, and iterations until convergence, respectively.

Several approaches have been proposed to reduce the computation cost of PPR [2, 3, 4, 12, 14, 16, 33]. These approaches are distinguished as matrix-based approaches and Monte Carlo-based approaches. However, they are not appropriate for interactive similarity search. Matrix-based approaches utilize the property that the definition of PPR similarities permits their factorization into some canonical form. For example, Fujiwara et al. proposed efficient approaches for PPR and RWR that are based on QR and LU decompositions, respectively [14, 16]. However, these approaches cannot handle interactive similarity search. This is because these approaches need the precomputation step of QR/LU decomposition before the search process; QR/LU decomposition must be repeated once the graph structure changes. In Monte Carlo-based approaches, random walks are performed to approximate similarity scores efficiently. The approximation approach proposed by Fogaras et al. [12] is one example of the Monte Carlo-based approach. They used random walks from the query nodes, which are stored in a disk; a fixed value of the scaling parameter c is used to compute approximate scores. However, approximate similarity scores are not desirable since it is difficult for any approximation to enhance the quality of real applications [14, 16]. Furthermore, the Monte Carlo-based approaches cannot effectively cope with on-the-fly changes of the given graph and the scaling parameter values since they precompute random walks before computing the similarity scores. Therefore, a fast approach that takes a different approach is demanded.

[2]http://wordnet.princeton.edu/

1.1 Problem statement

We address the following problem in this paper:

Problem (Ad-hoc top-k search for PPR).

Given: *Graph G, the scaling parameter c, required number of answer nodes k, and set of query nodes $\mathbb{Q}$.*

Find: *Top-k nodes exactly where k nodes are arranged in descending order with respect to their PPR scores.*

This problem is designed not to compute exact similarities of top-k nodes but to rank top-k nodes exactly with respect to their similarity scores. This is because, in real applications, exact node ranking is more important than the exactness of the similarity scores as described in [23].

1.2 Contributions

In this paper, we propose an efficient approach called *Castanet* that identifies the top-k nodes with exact node ranking. In order to reduce search cost, the proposal utilizes the following two approaches: (1) Estimate upper/lower similarity bounds in iterative style and (2) Dynamically prune unnecessary nodes for top-k search in each iteration. Based on the two approaches, Castanet terminates the iterative computations if the exact ranking of top-k nodes is obtained. Castanet has the following attractive characteristics:

- **Fast**: Compared with the original iterative approach [20], Castanet achieves high speed search by using the above approaches to construct subgraphs for similarity computations; Castanet can avoid computing similarities from the whole graph (Section 4.5 and 5.1).
- **Exact**: While our approach achieves efficient similarity search, it theoretically guarantees not to sacrifice accuracy even though it exploits an estimation approach to prune unnecessary nodes (Section 4.5 and 5.2); it returns the top-k nodes with exact ranking.
- **No-precomputations**: Our approach does not require any precomputation step unlike the previous approaches (Section 4.4). Therefore, it can effectively handle ad-hoc search for any arbitrary graphs, the scaling parameter, and number of answer nodes.
- **Parameter-free**: The proposed approach dispenses with the need for the user to set inner-parameters (Section 4.4). Thus, it provides the user with a simple solution to PPR-based applications.

Even though interactive similarity search based on PPR is effective to enhance application quality, it has been difficult to utilize due to the limitations of the previous approaches. However, by providing a sophisticated approach that suits interactive similarity search, Castanet will help to improve the effectiveness of future PPR-based applications.

The remainder of this paper is organized as follows: Section 2 describes related work. Section 3 gives an overview of the background. Section 4 introduces the main ideas and details of the approaches. Section 5 reviews the results of our experiments. Section 6 details case-studies of Castanet. Section 7 provides our conclusions.

2. RELATED WORK

The previous approaches for PPR fall into two categories: matrix-based and Monte Carlo-based approaches. However, neither is really suitable for interactive similarity search since matrix-based approaches need precomputation and Monte Carlo-based approaches compute approximate similarities.

Tong et al. studied two types of matrix-based approaches; B_LIN and NB_LIN [33]. They used the idea that eigenvalue decomposition can yield approximate similarity scores for RWR. They showed the proof of an error bound for NB_LIN. However, their proof is based on the assumption the normalized graph Laplacian [34] can be exploited to represent a graph. Furthermore, their approaches require setting of the target rank of eigen-decomposition, which impacts the approximation quality. It is impractical to subject a large graph to eigen-decomposition since it is time-consuming as they themselves pointed out. Fujiwara el al. proposed efficient approaches for PPR and RWR [14, 16]. Their approaches utilize the property that similarity computations can be factorized by QR/LU decompositions, so their approaches first compute QR/LU decompositions before commencing the search process. Even though their approaches can find top-k nodes exactly, they cannot effectively handle interactive similarity search since they need precomputation with the specified scaling parameter c.

Fogaras et al. proposed a Monte Carlo-based approach for PPR [12]. To compute approximate scores, they used fingerprints from query nodes; a fingerprint of a node is a random walk starting from the node. They precompute fingerprints and store them on disk to approximate similarity scores from the ending nodes of these random walks. They exploited the linearity property of PPR to approximate similarity scores. Bahmani et al. described an efficient Monte Carlo algorithm that also utilizes precomputed random walks to find top-k nodes [4]. To approximate a node, they count total number of times the node is visited by random walks. Even though their approach can incrementally update similarity scores, and so supports dynamically changing graphs, their approach assumes that the scaling parameter has a fixed value; it cannot effectively support interactive similarity search. A subsequent study by Bahmani et al. investigated a fast MapReduce algorithm for Monte Carlo approximation [3]. However, this algorithm suffers high I/O cost since MapReduce is a disk-based approach; it cannot efficiently handle interactive similarity search. Avrachenkov et al. proposed two Monte Carlo-based approaches, MC End Point and MC Complete Path, to find top-k nodes for PPR where probability $1 - c$ is used to restart from a query node [2]. To compute approximate similarities, MC End Point uses part of a random walk while MC Complete Path exploits the entire random walk. That is, MC End Point computes the similarity score of a node from the random walks that end at the node, and MC Complete Path evaluates the similarity score from the total number of visits to the node. Since MC Complete Path utilizes random walks more effectively than MC End Point, MC Complete Path theoretically requires approximately c times fewer random walks than MC End Point; MC Complete Path is faster than MC End Point. Since these approaches perform random walks with a given scaling parameter on the fly, they can compute top-k nodes in ad-hoc style even though they output approximate similarities. However, all Monte Carlo-based approaches require that the number of random walks be set to approximate similarity scores, which induces a trade-off between efficiency and approximation quality.

Table 1: Definition of main symbols.

Symbol	Definition
G	Given graph
N	Number of nodes in the graph
M	Number of edges in the graph
T	Number of iterations by the original iterative approach
c	Scaling parameter, $0 < c < 1$
k	Number of required answer nodes for top-k search
$\mathbb{V}$	Set of nodes in G
$\mathbb{E}$	Set of edges in G
$\mathbb{Q}$	Set of query nodes
$\mathbb{S}$	Set of selected nodes
$\mathbb{R}$	Set of reachable nodes to selected nodes
$\mathbb{L}$	Set of answer likely nodes
$\mathbb{D}$	Set of ranking determined nodes
$\mathbb{A}$	Set of answer nodes
$\mathbb{C}[u]$	Set of nodes adjacent to node u
$\mathbf{q}$	$N \times 1$ query node vector where $\sum q[i] = 1$
$\mathbf{s}$	$N \times 1$ similarity vector by PPR
$\mathbf{W}$	$N \times N$ column normalized adjacent matrix of G

Many other approaches have been proposed. For example, Cho et al. proposed the approach of computing lower bounding estimations for PPR-based web crawling [9]. To achieve this, they applied breadth-first search; they enumerate every possible path originating from the query nodes and compute the estimations through the enumeration. Even though their approach does not yield exact results, their lower bounding approach is acceptable for web crawling since their approach guarantees *not* to miss the high PPR pages (i.e. important pages). Jeh et al. suggested a framework that, for nodes that belong to a highly linked subset of hub nodes, provides a scalable solution for PPR [20]. Their approach precomputes similarity scores from hub nodes, and they approximately compute similarity scores from query nodes as a linear combination of the similarity scores from hub nodes. Basic Push Algorithm, studied by Gupta et al., exploits precomputed similarity scores of all nodes from hub nodes to find the top-k nodes [18]. The number of answer nodes yielded by this algorithm can be more than k since this algorithm uses upper similarity bounds. Unfortunately, the above approaches do not suit interactive similarity search since they compute approximate similarities or need precomputation with a fixed value scaling parameter.

3. PRELIMINARY

In this section, we formally define the notations and introduce the background of this paper. Table 1 lists the main symbols and their definitions. PPR is theoretically based on an intuitive "random surfer model", as same as for Google's PageRank [29]. Google's PageRank exploits the link structure of the graph to compute global importance, which can be used to rank nodes in the graph. PPR, introduced by Jeh and Widom [20], is based on the idea that this global notation of importance can be specialized to create personalized views of importance; importance scores of a node can be biased according to a user-specified set of nodes[3] (i.e., query nodes) and can be thought as the similarity to the nodes. RWR is a special case of PPR in that it has only a single node as a query node. Informally speaking, PPR is the steady-state probabilities of random walks; at each step in a random walk, it randomly selects an outgoing edge from the current node with probability c, and, with restart probability $1-c$, it jumps to a query node in accordance with its preference score. Let $\mathbb{V}$ and $\mathbb{E}$ be the set of nodes and edges in the given graph, respectively. That is, the given graph is represented as $G = \{\mathbb{V}, \mathbb{E}\}$. And let $\mathbf{s}$ be an $N \times 1$ PPR vector where the u-th element $s[u]$ denotes the PPR score of node u. $\mathbf{W}$ is a column-normalized adjacent matrix of the graph where its element $W[u, v]$ is the probability of node u being the next state given that the current state is node v. Computationally, the recursive PPR iteration propagates similarity scores until convergence as follows [20]:

$$\mathbf{s} = c\mathbf{W}\mathbf{s} + (1-c)\mathbf{q} \tag{1}$$

where $\mathbf{q}$ represents the query node vector in which the u-th element $q[u]$ corresponds to its preference score as a query node. If node u is not a query node, $q[u] = 0$. Elements in query nodes are normalized such that their sum is 1.

However, this iterative computation is not efficient if the top-k nodes are the target. This is because the similarity scores of all nodes are updated in the iterations. Furthermore, to find top-k nodes with exact node ranking, the exact similarity scores of all nodes must be obtained by applying the recursive procedures until convergence even though exact scores are not so important in top-k search [23]. This method takes $O((N+M)T)$ time to obtain exact similarities, which incurs excessive computational cost for large graphs and makes a fast solution essential as pointed out in [20].

4. PROPOSED METHOD

In this section, we present our proposal, Castanet, that efficiently finds top-k nodes with exact node ranking. First, Section 4.1 overviews the ideas underlying Castanet. That is followed by a full description including top-k search algorithm in Sections 4.2, 4.3, and 4.4. We also theoretically analyze its performance in Section 4.5.

4.1 Main ideas

The original iterative approach computes the similarities as the steady-state probabilities in the given graph as described in Section 3. This approach incurs high computational cost since it computes the similarities of all nodes by using the whole graph. To find top-k nodes efficiently, we update the similarity estimations of only selected nodes instead of all nodes in each iteration. Thus, we can avoid repeatedly processing the whole graph to find the top-k nodes.

In order to efficiently update the similarity estimations, each iteration dynamically extracts a subgraph by pruning unnecessary nodes and edges from the entire graph. Since the random walk probabilities that must be computed for the estimations are limited to existing edges in subgraphs, we can efficiently update similarity estimations. However, this raises a question; how can we dynamically identify the sets of necessary nodes and edges to find top-k nodes with exact node ranking? We dynamically estimate lower and upper similarity bounds in the iterations to identify unnecessary nodes and edges for top-k search [13]. The subgraphs are much smaller than the whole graph. As a result, we can obtain exact top-k nodes efficiently.

The subgraph approach requires fewer iterations than the original iterative approach. The original iterative approach applies recursive procedures until convergence to compute exact similarities. In contrast, our approach effectively utilizes lower and upper bounding estimations to obtain exact node ranking; if the ranking of a node is determined in an iteration, we do not reestimate the node in subsequent iterations. We terminate the search process once the node ranking is determined for all top-k nodes. This means that the search process is terminated without waiting for convergence; the number of iterations needed by our approach is smaller than that of the original iterative approach.

Our estimation approach provides a further advantage; this approach allows us to automatically determine the structures of subgraphs and the number of iterations. While previous approaches such as [2, 3, 4, 12, 33] require the setting of inner-parameters, which can impact search efficiency and approximation quality, our approach does not require any inner-parameter setting.

4.2 Similarity estimation

To realize efficient top-k search, we dynamically prune unnecessary nodes and edges by utilizing the lower and upper similarity estimations. We first introduce the notations used in the estimation in Section 4.2.1. We next formally define the lower and upper estimations in Section 4.2.2. We also show the theoretical aspects of the estimations.

4.2.1 Notation

To estimate lower and upper similarity bounds, we utilize $p_i[u]$, the random walk probability of length i which starts at a query node and ends at node u. In the random walk, we *do not* restart from a query node. Let $\mathbf{p}_i$ be the $N \times 1$ probability vector of i length random walk where u-th element corresponds to $p_i[u]$; $\mathbf{p}_i$ is computed as $\mathbf{p}_i = \mathbf{W}^i\mathbf{q}$ by exploiting i-th power of the adjacent matrix. Since it is clear that $\mathbf{p}_i = \mathbf{W}\mathbf{p}_{i-1}$, we can incrementally compute the random walk probability $\mathbf{p}_i$ from $\mathbf{p}_{i-1}$; the random walk probability of length i is computed as follows:

$$p_i[u] = \begin{cases} q[u] & (i = 0) \\ \sum_{v \in \mathbb{C}[u]} W[u, v] p_{i-1}[v] & (i \neq 0) \end{cases} \tag{2}$$

where $\mathbb{C}[u]$ is a node set whose edges are incident node u, i.e., $\mathbb{C}[u]$ is a set of directly adjacent nodes to node u in the original graph. In the i-th iteration ($i = 0, 1, 2, \ldots$), we select a set of nodes $\mathbb{S}_i$ for which we update lower and upper similarity bounds. The set of selected nodes initialized as the set of all nodes in the graph, i.e., $\mathbb{S}_0 = \mathbb{V}$, and nodes in $\mathbb{S}_i (i = 1, 2, \ldots)$ are selected as $\mathbb{S}_i$ is included in $\mathbb{S}_{i-1}$. Therefore $\mathbb{V} = \mathbb{S}_0 \supseteq \mathbb{S}_1 \supseteq \ldots \supseteq \mathbb{S}_i$. Details of the node selection are given in Section 4.3. To compute the upper similarity bound, we utilized $\mathbb{R}_i$, the set of nodes that is reachable [21] to any nodes in $\mathbb{S}_i$. Node u is reachable to node v if there is a path from node u to v in the original graph. Note that $\mathbb{S}_i \subseteq \mathbb{R}_i$, and, if $u \in \mathbb{S}_i$, it is obvious that $\mathbb{C}[u] \subseteq \mathbb{R}_i$. And $p_i[\mathbb{R}_i]$ indicates the probability of length i random walks that start at a query node and reaches a node in $\mathbb{R}_i$. That is, $p_i[\mathbb{R}_i] = \sum_{u \in \mathbb{R}_i} p_i[u]$. In addition, we use $W_{max}[u]$, the maximum probability incident to node u

[3] Called "preferred page" in [20].

in the matrix $\mathbf{W}$, i.e., $W_{max}[u] = \max\{W[u,v] : v \in \mathbb{V}\}$ so as to estimate the upper similarity bound.

4.2.2 *Definition*

We compute the lower similarity bound in each iteration by exploiting the random walk probabilities as follows:

Definition 1 (Lower bound). *The lower estimation of node u in the i-th iteration, $\underline{s}_i[u]$, is defined as follows:*

$$\underline{s}_i[u] = \begin{cases} (1-c)p_i[u] & (i=0) \\ \underline{s}_{i-1}[u] + (1-c)c^i p_i[u] & (i \neq 0) \end{cases} \tag{3}$$

We show that Equation (3) has the lower bounding property in the next paragraph. This definition implies that (1) if $i = 0$, a lower estimate of node u is obtained by random walk probability $p_i[u]$ and the scaling parameter, and (2) otherwise, we can incrementally update the lower estimation $\underline{s}_i[u]$ by the random walk probability of the node and the scaling parameter. This definition also indicates that we can compute the lower estimate of a node in $O(1)$ time if the random walk probability $p_i[u]$ is already obtained.

We introduce the following lemma to show the property of the above estimation approach.

Lemma 1 (Lower bound). *For any node in the graph, $\underline{s}_i[u] \leq s[u]$ holds in all iterations.*

Proof Prior to proving Lemma 1, we first prove that $\lim_{i\to\infty}(c\mathbf{W})^i = \mathbf{0}$ holds, i.e., the matrix $(c\mathbf{W})^i$ equal to a zero matrix after convergence. It is obvious $\lim_{i\to\infty} c^i = 0$ because $0 < c < 1$. Since matrix $\mathbf{W}$ is the column normalized adjacent matrix of the graph, the i-th power of adjacent matrix $\mathbf{W}^i$ corresponds to the probabilities of random walks of length i; none of the elements in $\mathbf{W}^i$ can be larger than 1 or smaller than 0. Therefore, we have

$$\lim_{i\to\infty}(c\mathbf{W})^i = \lim_{i\to\infty} c^i \lim_{i\to\infty} \mathbf{W}^i = \mathbf{0} \tag{4}$$

We next prove Lemma 1. From Equation (1), PPR scores can be obtained as follows:

$$\mathbf{s} = (1-c)(\mathbf{I} - c\mathbf{W})^{-1}\mathbf{q} \tag{5}$$

where $\mathbf{I}$ is the identity matrix and $(\mathbf{I} - c\mathbf{W})^{-1}$ is the inverse matrix of $\mathbf{I} - c\mathbf{W}$. As shown in [19], if $\lim_{i\to\infty}(c\mathbf{W})^i = \mathbf{0}$,

$$(\mathbf{I} - c\mathbf{W})^{-1} = \mathbf{I} + c\mathbf{W} + c^2\mathbf{W}^2 + \ldots = \textstyle\sum_{j=0}^{\infty}(c\mathbf{W})^j \tag{6}$$

by letting $(c\mathbf{W})^0 = \mathbf{I}$. From Equation (5) and (6), the following equation holds:

$$\mathbf{s} = (1-c)\left\{\textstyle\sum_{j=0}^{\infty}(c\mathbf{W})^j\right\}\mathbf{q} = (1-c)\textstyle\sum_{j=0}^{\infty} c^j \mathbf{p}_j \tag{7}$$

Equation (7) indicates that the u-th element of $\mathbf{s}$, $s[u]$, can be computed as follows:

$$s[u] = (1-c)\textstyle\sum_{j=0}^{\infty} c^j p_j[u] \tag{8}$$

From Equation (3), $\underline{s}_i[u]$ can be computed as follows:

$$\begin{aligned} \underline{s}_i[u] &= \underline{s}_{i-1}[u] + (1-c)c^i p_i[u] \\ &= \underline{s}_{i-2}[u] + (1-c)c^i p_i[u] + (1-c)c^{i-1}p_{i-1}[u] \\ &= (1-c)c^i p_i[u] + (1-c)c^{i-1}p_{i-1}[u] + \ldots + (1-c)p_0[u] \end{aligned} \tag{9}$$

Therefore, since $c > 0$ and $p_j[u] \geq 0$,

$$\underline{s}_i[u] = (1-c)\textstyle\sum_{j=0}^{i} c^j p_j[u] \leq (1-c)\sum_{j=0}^{\infty} c^j p_j[u] = s[u] \tag{10}$$

which completes the proof. □

We exploit the lower similarity bound introduced in Definition 1 to estimate upper similarity bound. The definition of the upper similarity bound is given as follows by utilizing random walk probabilities:

Definition 2 (Upper bound). *The upper estimation of node u in the i-th iteration, $\overline{s}_i[u]$, is defined as follows:*

$$\overline{s}_i[u] = \underline{s}_i[u] + c^{i+1}W_{max}[u]\, p_i[\mathbb{R}_i] \tag{11}$$

This definition represents that the upper bound estimate of node u can be updated in $O(1)$ time by c, $W_{max}[u]$, and $p_i[\mathbb{R}_i]$. We have the following lemma for the upper similarity bound in each iteration:

Lemma 2 (Upper bound). *For any node in the graph, $\overline{s}_i[u] \geq s[u]$ holds in all iterations.*

Proof To prove the above lemma, we first show the property that $p_{i+j}[\mathbb{R}_{i+j}] \leq p_i[\mathbb{R}_i]$ ($j = 0, 1, \ldots$) holds in all iterations by using mathematical induction.

Initial step: If $j=0$, it is obvious that $p_{i+j}[\mathbb{R}_{i+j}] = p_i[\mathbb{R}_i]$.

Inductive step: We assume that $p_{i+j-1}[\mathbb{R}_{i+j-1}] \leq p_i[\mathbb{R}_i]$. Since nodes are selected as $\mathbb{S}_{i+j} \subseteq \mathbb{S}_{i+j-1}$, it is obvious that $\mathbb{R}_{i+j} \subseteq \mathbb{R}_{i+j-1}$. Therefore, from Equation (2), we have

$$\begin{aligned} p_{i+j}[\mathbb{R}_{i+j}] &= \textstyle\sum_{v\in\mathbb{R}_{i+j}} p_{i+j}[v] \\ &= \textstyle\sum_{v\in\mathbb{R}_{i+j}} \sum_{w\in\mathbb{C}[v]} W[v,w]p_{i+j-1}[w] \\ &\leq \textstyle\sum_{w\in\mathbb{R}_{i+j-1}} \sum_{v\in\mathbb{R}_{i+j-1}} W[v,w]p_{i+j-1}[w] \\ &\leq \textstyle\sum_{w\in\mathbb{R}_{i+j-1}} p_{i+j-1}[w] = p_{i+j-1}[\mathbb{R}_{i+j-1}] \end{aligned} \tag{12}$$

This is because $\mathbb{C}[v] \subseteq \mathbb{R}_{i+j-1}$ and $\mathbf{W}$ is the column normalized adjacent matrix. As a result, $p_{i+j}[\mathbb{R}_{i+j}] \leq p_{i+j-1}[\mathbb{R}_{i+j-1}] \leq p_i[\mathbb{R}_i]$. This completes the inductive step.

By exploiting this property, we prove Lemma 2. From Equation (2), (8), and (10), PPR similarity of node u, $s[u]$, is computed as follows:

$$\begin{aligned} s[u] &= (1-c)\textstyle\sum_{j=0}^{i} c^j p_j[u] + (1-c)\sum_{j=i+1}^{\infty} c^j p_j[u] \\ &= \underline{s}_i[u] + (1-c)\textstyle\sum_{j=i+1}^{\infty}\sum_{v\in\mathbb{C}[u]} c^j W[u,v]p_{j-1}[v] \end{aligned} \tag{13}$$

Since $W[u,v] \leq W_{max}[u]$ and $\mathbb{C}[u] \subseteq \mathbb{R}_{j-1}$, we have

$$\begin{aligned} s[u] &\leq \underline{s}_i[u] + (1-c)W_{max}[u]\textstyle\sum_{j=i+1}^{\infty} c^j \sum_{v\in\mathbb{R}_{j-1}} p_{j-1}[v] \\ &= \underline{s}_i[u] + (1-c)W_{max}[u]\textstyle\sum_{j=i+1}^{\infty} c^j p_{j-1}[\mathbb{R}_{j-1}] \end{aligned} \tag{14}$$

Since $\sum_{j=i+1}^{\infty} c^j p_{j-1}[\mathbb{R}_{j-1}] = \sum_{j=0}^{\infty} c^{i+j+1}p_{i+j}[\mathbb{R}_{i+j}]$ and, as described above, $p_{i+j}[\mathbb{R}_{i+j}] \leq p_i[\mathbb{R}_i]$, we have

$$\begin{aligned} s[u] &\leq \underline{s}_i[u] + (1-c)W_{max}[u]\, p_i[\mathbb{R}_i]\textstyle\sum_{j=0}^{\infty} c^{i+j+1} \\ &= \underline{s}_i[u] + (1-c)W_{max}[u]\, p_i[\mathbb{R}_i]\frac{c^{i+1} - c^{\infty}}{1-c} \\ &\leq \underline{s}_i[u] + c^{i+1}W_{max}[u]\, p_i[\mathbb{R}_i] = \overline{s}_i[u] \end{aligned} \tag{15}$$

that completes the proof. □

We recursively compute lower and upper estimations in each iteration by using the random walk probability as shown in Definition 1 and 2. These estimations offer one significant advantage; their approximation performance improves with the number of iterations. We show this property by introducing the following lemma:

Lemma 3 (Enhancing accuracy). *$\underline{s}_i[u] \geq \underline{s}_{i-1}[u]$ and $\overline{s}_i[u] \leq \overline{s}_{i-1}[u]$ hold in the i-th iteration.*

Proof We first prove that $\underline{s}_i[u] \geq \underline{s}_{i-1}[u]$. From Equation (3), we have $\underline{s}_i[u] - \underline{s}_{i-1}[u] = (1-c)c^i p_i[u] \geq 0$. We next prove that $\overline{s}_i[u] \leq \overline{s}_{i-1}[u]$. From Equation (3) and (11), $\overline{s}_i[u] - \overline{s}_{i-1}[u]$ is computed as follows:

$$\begin{aligned} &\overline{s}_i[u] - \overline{s}_{i-1}[u] \\ &= \underline{s}_i[u] - \underline{s}_{i-1}[u] + c^i W_{max}[u](cp_i[\mathbb{R}_i] - p_{i-1}[\mathbb{R}_{i-1}]) \quad (16) \\ &= c^i\{(1-c)p_i[u] + W_{max}[u](cp_i[\mathbb{R}_i] - p_{i-1}[\mathbb{R}_{i-1}])\} \end{aligned}$$

From Equation (2),

$$\begin{aligned} p_i[u] &= \textstyle\sum_{v\in\mathbb{C}[u]} W[u,v]\, p_{i-1}[v] \\ &\leq W_{max}[u] \textstyle\sum_{v\in\mathbb{R}_{i-1}} p_{i-1}[v] = W_{max}[u]\, p_{i-1}[\mathbb{R}_{i-1}] \end{aligned} \quad (17)$$

From Equation (12), $p_i[\mathbb{R}_i] \leq p_{i-1}[\mathbb{R}_{i-1}]$. Therefore,

$$\begin{aligned} &\overline{s}_i[u] - \overline{s}_{i-1}[u] \\ &\leq c^i W_{max}[u]\{(1-c)p_{i-1}[\mathbb{R}_{i-1}] + cp_{i-1}[\mathbb{R}_{i-1}] - p_{i-1}[\mathbb{R}_{i-1}]\} = 0 \end{aligned} \quad (18)$$

This completes the proof. □

Lemma 3 ensures that our approach has the property of finding top-k nodes with exact node ranking without attempting to compute exact similarities until convergence as described in the next section. The lower/upper estimations have a property of converging to the exact values as follows:

Lemma 4 (Convergence of estimations). *After convergence, lower and upper estimations are the same as the exact similarities for all nodes, i.e., $\underline{s}_\infty[u] = \overline{s}_\infty[u] = s[u]$.*

Proof We first prove that $\underline{s}_\infty[u] = s[u]$. From Equation (10), it is clear that $\underline{s}_\infty[u] = (1-c)\sum_{j=0}^{\infty} c^j p_j[u] = s[u]$. We next prove that $\overline{s}_\infty[u] = s[u]$. From Equation (11), $\overline{s}_\infty[u] = \underline{s}_\infty[u] + c^\infty W_{max}[u]\, p_\infty[\mathbb{R}_\infty]$. Since $c^\infty = 0$, $0 \leq W_{max}[u] \leq 1$, and $0 \leq p_\infty[\mathbb{R}_\infty] \leq 1$, $c^\infty W_{max}[u]\, p_\infty[\mathbb{R}_\infty] = 0$. Therefore, $\overline{s}_\infty[u] = \underline{s}_\infty[u] = s[u]$ holds. □

Lemma 4 ensures that our approach has the property of ranking top-k nodes exactly as described in the next section.

4.3 Subgraph-based search

We dynamically construct subgraphs to efficiently compute random walk probabilities. We update the lower and upper estimations for selected nodes by using the random walk probabilities. In this section, we first detail our approach of selecting nodes in Section 4.3.1, and we formally define subgraphs in Section 4.3.2.

4.3.1 Node selection

In the proposed approach, we select a node to update the lower and upper similarity bounds of the node if (1) the node can be a top-k node (i.e., an answer-likely node) and (2) the ranking of the node in top-k nodes is *not* still determined by the estimations. Let θ_i be the k-th highest lower similarity bound among all nodes in the i-th iteration, a set of answer-likely nodes in the i-th iteration, $\mathbb{L}_i$, is defined as follows:

Definition 3 (Answer-likely nodes). *The following equation gives the set of answer-likely nodes in the i-th iteration, $\mathbb{L}_i$, whose exact similarity can be higher than θ_i:*

$$\mathbb{L}_i = \{u \in \mathbb{V} : \overline{s}_i[u] \geq \theta_i\} \quad (19)$$

The above definition indicates that a node is an answer-likely node if its upper similarity bound is not lower than θ_i. This is because, if an upper estimation of a node is lower than θ_i, the exact similarity of the node cannot be more than θ_i. As shown in later, the sets of answer-likely nodes become smaller with each iteration. A set of ranking determined nodes in top-k nodes by the i-th iteration, $\mathbb{D}_i$, is defined as follows:

Definition 4 (Ranking determined nodes). *If $|\mathbb{L}_i| = k$ (i.e., the number of answer-likely nodes is k), the following equation gives the definition of the set of ranking-determined nodes, $\mathbb{D}_i$, whose exact ranks in top-k nodes are already fixed by the i-th iteration:*

$$\mathbb{D}_i = \{u \in \mathbb{L}_i : \underline{s}_i[u] > \overline{s}_i[v] \text{ or } \overline{s}_i[u] < \underline{s}_i[v], u \neq v, \forall v \in \mathbb{L}_i\} \quad (20)$$

Otherwise (i.e., $|\mathbb{L}_i| \neq k$), $\mathbb{D}_i$ is defined as follows:

$$\mathbb{D}_i = \emptyset \quad (21)$$

Definition 4 defines node u as a ranking-determined node if (1) the number of answer-likely nodes is k, and (2) there does not exist any node $v(\neq u)$ whose lower or upper similarity bound is between $\underline{s}_i[u]$ and $\overline{s}_i[u]$. That is, if there is a node whose lower or upper similarity bound is between $\underline{s}_i[u]$ and $\overline{s}_i[u]$, the rank of node u cannot be determined.

In Definition 4, if $|\mathbb{L}_i| = k$, it requires $O(k \log k)$ time to compute $\mathbb{D}_i$ since $\mathbb{D}_i$ can be obtained by sorting the nodes of $\mathbb{L}_i$ in the lower and upper estimations. If $|\mathbb{L}_i| \neq k$, it is unnecessary to compute $\mathbb{D}_i$ from Equation (21). By exploiting the definitions of $\mathbb{L}_i$ and $\mathbb{D}_i$, the set of selected nodes, $\mathbb{S}_i$ is defined as follows:

Definition 5 (Selected nodes). *The following equation gives the definition of the set of selected nodes, $\mathbb{S}_i$, whose lower/upper estimations are updated in the i-th iteration by random walk probabilities:*

$$\mathbb{S}_i = \begin{cases} \mathbb{V} & (i = 0) \\ \mathbb{L}_{i-1} \backslash \mathbb{D}_{i-1} & (i \neq 0) \end{cases} \quad (22)$$

In the above equation, $\mathbb{L}_{i-1} \backslash \mathbb{D}_{i-1}$ represents the relative complement of $\mathbb{L}_{i-1}$ with respect to $\mathbb{D}_{i-1}$ which is given as $\{u \in \mathbb{L}_{i-1} : u \notin \mathbb{D}_{i-1}\}$.

Definition 5 indicates that we first compute the lower/upper estimations for all nodes, and we do not update the estimations of a node if (1) the node is not an answer-likely node or (2) the ranking of the node has already been fixed by prior estimations. In other words, we update the estimation of a node if the node is an answer-likely node and its ranking has not been determined by prior estimations.

To show the property of the selected nodes, we introduce several lemmas on node sets $\mathbb{L}_i$ and $\mathbb{D}_i$. We first show the two properties of the set of answer-likely nodes $\mathbb{L}_i$ as follows by letting $\mathbb{A}$ be the set of answer nodes:

Lemma 5 (Monotonic decrease of $\mathbb{L}_i$). *The elements in the node set of answer-likely nodes, $\mathbb{L}_i$, are monotonic decreasing in the iterations. That is, $\mathbb{L}_i \subseteq \mathbb{L}_{i-1}$ holds.*

Proof θ_i is the k-th highest lower estimation in the i-th iteration. Therefore, we have $\theta_i \geq \theta_{i-1}$ since lower bound estimations have monotonic increasing property as described in Lemma 3. As a result, (1) if node u is included in $\mathbb{L}_i$, the node must be included in $\mathbb{L}_{i-1}$ since $\overline{s}_{i-1}[u] \geq \overline{s}_i[u] \geq \theta_i \geq \theta_{i-1}$, and (2) otherwise, the node can be included in $\mathbb{L}_{i-1}$ since $\theta_i > \overline{s}_{i-1}[u] \geq \overline{s}_i[u] \geq \theta_{i-1}$ can be hold. □

Lemma 6 (Convergence of $\mathbb{L}_i$). *The set of answer-likely nodes equals to the set of answer nodes after convergence, i.e., $\mathbb{L}_\infty = \mathbb{A}$.*

Proof If θ is the k-th highest exact similarity among all nodes, we have $\theta_\infty = \theta$ after convergence from Lemma 4. Therefore, from Lemma 4, we have

$$\mathbb{L}_\infty = \{u \in \mathbb{V} : \overline{s}_\infty[u] \geq \theta_\infty\} = \{u \in \mathbb{V} : s[u] \geq \theta\} = \mathbb{A} \quad (23)$$

which completes the proof. □

Even though Definition 3 indicates that all the nodes in the original graph are needed to compute $\mathbb{L}_i$, we can efficiently compute $\mathbb{L}_i$ in the iterations. If $i \neq 0$ and $|\mathbb{L}_{i-1}| \neq k$, $\mathbb{L}_i$ is computed as follows:

$$\mathbb{L}_i = \{u \in \mathbb{L}_{i-1} : \overline{s}_i[u] \geq \theta_i\} \quad (24)$$

Equation (24) can be obtained by replacing $\mathbb{V}$ with $\mathbb{L}_{i-1}$ in Equation (19). That is, we can compute $\mathbb{L}_i$ from $\mathbb{L}_{i-1}$. This is because, if a node is not included in $\mathbb{L}_{i-1}$, the node cannot be included in $\mathbb{L}_i$ from Lemma 5. Moreover, if $i \neq 0$ and $|\mathbb{L}_{i-1}| = k$, $\mathbb{L}_i$ is computed as follows:

$$\mathbb{L}_i = \mathbb{L}_{i-1} \quad (25)$$

This is because $\mathbb{L}_i \subseteq \mathbb{L}_{i-1}$ and $\mathbb{L}_i$ converges to $\mathbb{A}$ from Lemma 5 and 6. We introduce the following properties of the set of ranking-determined nodes from Lemma 5 and 6:

Lemma 7 (Monotonic increase of $\mathbb{D}_i$). *The set of ranking-determined nodes, $\mathbb{D}_i$, has the property of monotonic increase; we have $\mathbb{D}_i \supseteq \mathbb{D}_{i-1}$ in the i-th iteration.*

Proof We first give proof of the case of $|\mathbb{L}_{i-1}| \neq k$. From Definition 4, in this case, we have $\mathbb{D}_{i-1} = \emptyset \subseteq \mathbb{D}_i$. We next show the proof of the case of $|\mathbb{L}_{i-1}| = k$. In this case, $\mathbb{L}_i = \mathbb{L}_{i-1} = \mathbb{A}$ holds due to Lemma 5 and 6. If node u is included in the node set $\mathbb{D}_{i-1}$, the node must be included in the node set $\mathbb{D}_i$ since (1) the estimations have the property of enhancing accuracy from Lemma 3, and (2) $\mathbb{L}_i = \mathbb{L}_{i-1}$. If node u is not included in node set $\mathbb{D}_{i-1}$, the node can be included in node set $\mathbb{D}_i$ due to the accuracy enhancement property of lower/upper estimations. □

Lemma 8 (Convergence of $\mathbb{D}_i$). *After convergence, we have $\mathbb{D}_\infty = \mathbb{A}$; the set of ranking-determined nodes equals to the set of answer nodes.*

Proof We have $\underline{s}_\infty[u] = \overline{s}_\infty[u] = s[u]$ and $\mathbb{L}_\infty = \mathbb{A}$ after convergence from Lemma 4 and 6, respectively. Therefore, from Definition 4, we have,

$$\begin{aligned}\mathbb{D}_\infty &= \{u \in \mathbb{L}_\infty : \underline{s}_\infty[u] > \overline{s}_\infty[v] \text{ or } \overline{s}_\infty[u] < \underline{s}_\infty[v], u \neq v, \forall v \in \mathbb{L}_\infty\} \\ &= \{u \in \mathbb{A} : s[u] > s[v] \text{ or } s[u] < s[v], u \neq v, \forall v \in \mathbb{A}\} = \mathbb{A}\end{aligned} \quad (26)$$

Therefore, $\mathbb{D}_\infty = \mathbb{A}$ holds after convergence. □

Lemma 5 and 7 indicate that the set of nodes $\mathbb{L}_i$ and $\mathbb{D}_i$ have monotonic decrease and increase properties, respectively. Lemma 6 and 8 ensure that node sets $\mathbb{L}_i$ and $\mathbb{D}_i$ equal to the answer node set $\mathbb{A}$ after convergence. Therefore, we can introduce the following lemma for the selected nodes:

Lemma 9 (Set of the selected nodes). *The set of selected nodes is monotonic decreasing and equals to an empty set after convergence. That is, $\mathbb{S}_i \subseteq \mathbb{S}_{i-1}$ and $\mathbb{S}_\infty = \emptyset$.*

Proof This is obvious from Lemma 5, 6, 7, and 8. □

Since, as described in Definition 5, the set of selected nodes is initialized to the set of nodes in the graph, the set of selected nodes has the property that $\mathbb{V} = \mathbb{S}_0 \supseteq \mathbb{S}_1 \supseteq \ldots \supseteq \mathbb{S}_i$. As shown in Section 4.2, this property enables our approach to compute the lower/upper estimations. Moreover, the property of $\mathbb{S}_\infty = \emptyset$ implies that our approach terminates in finite iteration number since $\mathbb{S}_\infty = \emptyset$ indicates that there is no node whose estimations are updated.

4.3.2 *Subgraphs construction*

In this section, we describe our approach that constructs subgraphs to efficiently update the estimations. A naive approach to updating the lower/upper estimations treats the whole graph in each iteration. However, this approach is too computationally expensive since it needs to obtain random walk probabilities of all nodes. Therefore, we construct subgraphs by pruning unnecessary nodes and edges from the original graph. In this section, we first define the subgraph and then show its properties. We also describe how to incrementally compute subgraphs in the iterations.

As shown in Definition 1 and 2, the lower and upper estimations are obtained by utilizing the random walk probabilities obtained by Equation (2). Therefore, the subgraphs are designed to efficiently compute random walk probabilities. To construct the subgraphs, we use the set of nodes, $\mathbb{H}_i$, whose number of hops from the query nodes is within i; $\mathbb{H}_i$ is composed of the nodes that are not more than i hops from the query nodes. We construct the subgraph in the i-th iteration, G_i, by exploiting $\mathbb{H}_i$ as follows:

Definition 6 (Subgraph). *Let $G_i = \{\mathbb{V}_i, \mathbb{E}_i\}$ be the subgraph of G in the i-th iteration where $\mathbb{V}_i \subseteq \mathbb{V}$ and $\mathbb{E}_i \subseteq \mathbb{E}$, $\mathbb{V}_i$ and $\mathbb{E}_i$ are defined as follows:*

$$\mathbb{V}_i = \mathbb{H}_i \cap \mathbb{R}_i \quad (27)$$

$$\mathbb{E}_i = \{(u, v) \in \mathbb{E} : u \in \mathbb{V}_i, v \in \mathbb{V}_i\} \quad (28)$$

where (u, v) represents an edge from node u to v.

The above definition indicates that (1) subgraph G_i has a node in the original graph if the node is within i hops from the query nodes and is reachable to the selected nodes, and (2) an edge of the original graph exists in the subgraph if the nodes linked by the edge in the original graph exist in the subgraph. We introduce the following lemma to show the property of the subgraph.

Lemma 10 (Lower/upper estimations by subgraph). *The lower/upper estimations for the selected nodes in the i-th iteration are obtained from $\mathbb{V}_i$ and $\mathbb{E}_i$.*

Proof As described in Section 4.2.1, $p_i[u]$ is the random walk probability of length i that starts at a query node and ends at node u. So, if a node is more than i-hops away from the query nodes, the random walk probability of the node must be 0 in the i-th iteration. Therefore, we can obtain the lower/upper estimations with node set $\mathbb{H}_i$ and the set of edges that are incident to $\mathbb{H}_i$. From Equation (2), (3), and (11), it is clear that, if a node is *not* reachable to $\mathbb{S}_i$, the score of the random walk probability of the node does not affect the score of the lower/upper estimations of $\mathbb{S}_i$. This indicates that node set $\mathbb{R}_i$ and the set of edges that are incident from $\mathbb{R}_i$ need to be processed to compute the lower/upper estimations for selected nodes. As a result, we need only node set $\mathbb{V}_i = \mathbb{H}_i \cap \mathbb{R}_i$ and edge set $\mathbb{E}_i = \{(u, v) \in \mathbb{E} : u \in \mathbb{V}_i, v \in \mathbb{V}_i\}$ to obtain the lower/upper estimations of the selected nodes in the i-th iteration. □

This proof implies that we can compute random walk probabilities as well as the lower/upper estimations of the selected nodes by the subgraphs. We exploit the subgraphs to efficiently compute random walk probabilities as follows:

Definition 7 (Probability computation by subgraph). *Let $\mathbb{C}_i[u]$ be the set of nodes that are incident to node u in* *the subgraph G_i, we compute the random walk probability of node u in i-th iteration, $p_i[u]$, as follows:*

$$p_i[u] = \begin{cases} q[u] & (i = 0) \\ \sum_{v \in \mathbb{C}_i[u]} W[u,v] p_{i-1}[v] & (i \neq 0) \end{cases} \quad (29)$$

The above equation can be obtained by replacing $\mathbb{C}[u]$ with $\mathbb{C}_i[u]$ in Equation (2). To find the top-k nodes, we compute the random walk probabilities of the nodes in the subgraphs by the above definition. We obtain the lower/upper estimations of selected nodes by exploiting Definition 1 and 2. Even though the naive approach processes the whole graph, we compute random walk probabilities from the subgraphs to reduce the computation cost.

However, it can require high computational cost to construct the subgraphs since this approach needs the whole graph to compute the subgraphs based on Definition 6. We, therefore, introduce an incremental approach to obtaining subgraphs that utilizes a set of nodes $\mathbb{h}_i$ and a set of edges $\mathbb{e}_i$. $\mathbb{h}_i$ is defined as the set of nodes that are i hops away from the query nodes. From this definition, it is clear that $\mathbb{h}_0 = \mathbb{H}_0 = \mathbb{Q}$ and $\mathbb{H}_i = \bigcup_{j=0}^{i} \mathbb{h}_j = \mathbb{H}_{i-1} + \mathbb{h}_i$. $\mathbb{e}_i$ is the set of edges that link node sets $\mathbb{h}_i$ and $\mathbb{H}_i$. That is, $\mathbb{e}_i = \{(u,v) \in \mathbb{E} : u \in \mathbb{h}_i, v \in \mathbb{H}_{i-1}$ or $u \in \mathbb{H}_{i-1}, v \in \mathbb{h}_i$ or $u \in \mathbb{h}_i, v \in \mathbb{h}_i\}$ since $\mathbb{H}_i = \mathbb{H}_{i-1} + \mathbb{h}_i$. Note that $\mathbb{h}_i$ and $\mathbb{e}_i$ can be obtained from a single breadth-first search rooted on the query nodes; $\mathbb{h}_i$ and $\mathbb{e}_i$ can be obtained within $O(N + M)$ time. We have the following lemma for these two sets:

Lemma 11 (Subgraph inclusion). *Let nodes and edges sets be $\mathbb{V}'_i = \mathbb{V}_{i-1} + \mathbb{h}_i$ and $\mathbb{E}'_i = \mathbb{E}_{i-1} + \mathbb{e}_i$, respectively, we have $\mathbb{V}_i \subseteq \mathbb{V}'_i$ and $\mathbb{E}_i \subseteq \mathbb{E}'_i$ if $i \neq 0$.*

Proof If $i \neq 0$, since $\mathbb{H}_i = \mathbb{H}_{i-1} + \mathbb{h}_i$ and $\mathbb{R}_i \subseteq \mathbb{R}_{i-1}$ hold, from Equation (27), we have

$$\mathbb{V}_i = (\mathbb{H}_{i-1} + \mathbb{h}_i) \cap \mathbb{R}_i \subseteq (\mathbb{H}_{i-1} + \mathbb{h}_i) \cap \mathbb{R}_{i-1} \quad (30)$$

Therefore,

$$\mathbb{V}_i \subseteq \mathbb{H}_{i-1} \cap \mathbb{R}_{i-1} + \mathbb{h}_i \cap \mathbb{R}_{i-1} \subseteq \mathbb{V}_{i-1} + \mathbb{h}_i = \mathbb{V}'_i \quad (31)$$

If $i \neq 0$, since $\mathbb{V}_i \subseteq \mathbb{V}_{i-1} + \mathbb{h}_i$, from Equation (28), we have

$$\mathbb{E}_i \subseteq \{(u,v) \in \mathbb{E} : u \in (\mathbb{V}_{i-1} + \mathbb{h}_i), v \in (\mathbb{V}_{i-1} + \mathbb{h}_i)\} \quad (32)$$

Therefore, since $\mathbb{V}_{i-1} = \mathbb{H}_{i-1} \cap \mathbb{R}_{i-1} \subseteq \mathbb{H}_{i-1}$, we have

$$\begin{aligned} \mathbb{E}_i \subseteq \{(u,v) \in \mathbb{E} : u \in \mathbb{V}_{i-1}, v \in \mathbb{V}_{i-1}\} + \\ \{(u,v) \in \mathbb{E} : u \in \mathbb{H}_{i-1}, v \in \mathbb{h}_i \\ \text{or } u \in \mathbb{h}_i, v \in \mathbb{H}_{i-1} \text{ or } u \in \mathbb{h}_i, v \in \mathbb{h}_i\} \end{aligned} \quad (33)$$

As a result, $\mathbb{E}_i \subseteq \mathbb{E}_{i-1} + \mathbb{e}_i = \mathbb{E}'_i$ holds. □

Based on Lemma 11, we incrementally construct the subgraphs in the iterations. Let a graph be $G'_i = \{\mathbb{V}'_i, \mathbb{E}'_i\}$, the graph G'_i can be incrementally obtained since $\mathbb{V}'_i = \mathbb{V}_{i-1} + \mathbb{h}_i$ and $\mathbb{E}'_i = \mathbb{E}_{i-1} + \mathbb{e}_i$. That is, graph G'_i can be obtained by adding i hop distant nodes and their corresponding edges to G_{i-1}. Since (1) $G_i \subseteq G'_i$ holds from the above lemma and (2) $\mathbb{h}_i$ and $\mathbb{e}_i$ can include nodes and edges that are not paths to $\mathbb{S}_i$, respectively, we compute subgraph G_i by determining the paths to $\mathbb{S}_i$ in graph G'_i by applying breadth-first search inversely from $\mathbb{S}_i$.

Algorithm 1 shows our incremental approach for subgraph construction. If $i = 0$, the algorithm initializes the node and edge sets to $\mathbb{V}_0 = \mathbb{Q}$ and $\mathbb{E}_0 = \{(u,v) \in \mathbb{E} : u \in \mathbb{Q}, v \in \mathbb{Q}\}$, respectively (lines 2-3). This is because $\mathbb{V}_0 = \mathbb{H}_0 \cap \mathbb{R}_0 = \mathbb{Q} \cap \mathbb{V} = \mathbb{Q}$ from Equation (22) and (27). Otherwise, it computes graph G'_i from the graph of the previous iteration, G_{i-1} (lines 5-7). It computes $\mathbb{V}_i$ and $\mathbb{E}_i$ from G'_i by utilizing breadth-first search (lines 8-9).

Algorithm 1 Subgraph construction

```
Input: $G_{i-1}$, subgraph of previous iteration; $\mathbb{h}_i$, set of nodes; $\mathbb{e}_i$, set
    of edges; $\mathbb{S}_i$, set of selected nodes
Output: $\mathbb{G}_i$, subgraph in the $i$-th iteration
 1: if $i = 0$ then
 2:     $\mathbb{V}_0 := \mathbb{Q}$;
 3:     $\mathbb{E}_0 := \{(u,v) \in \mathbb{E} : u \in \mathbb{Q}, v \in \mathbb{Q}\}$;
 4: else
 5:     $\mathbb{V}'_i := \mathbb{V}_{i-1} + \mathbb{h}_i$;
 6:     $\mathbb{E}'_i := \mathbb{E}_{i-1} + \mathbb{e}_i$;
 7:     $G'_i := \{\mathbb{V}'_i, \mathbb{E}'_i\}$;
 8:     compute paths to $\mathbb{S}_i$ in $G'_i$;
 9:     compute $\mathbb{V}_i$ and $\mathbb{E}_i$ from the paths;
10: end if
11: $G_i := \{\mathbb{V}_i, \mathbb{E}_i\}$;
12: return $\mathbb{G}_\mathbb{i}$;
```

As shown in Algorithm 1, we can incrementally compute the subgraphs without using the whole graph; we can find answer nodes efficiently. The next section describes the top-k search algorithm based on the above subgraph approach.

4.4 Search algorithm

Algorithm 2 shows our algorithm Castanet. It first initializes the set of selected nodes (line 2), and computes the subgraph (lines 7-8). If a node is included in the subgraph, it computes the random walk probability of the node (lines 9-11) since such a node is necessary to obtain the estimations for the selected nodes (Lemma 10). And it computes the estimations of the selected node (lines 12-15). If $i = 0$, it computes $\mathbb{L}_i$ by Definition 3 (lines 16-17). Otherwise, it updates $\mathbb{L}_i$ from $\mathbb{L}_{i-1}$ (lines 18-24). Since $\mathbb{D}_i = \emptyset$ if $|\mathbb{L}_i| \neq k$ from Definition 4, it computes $\mathbb{D}_i$ when $|\mathbb{L}_i| = k$ (lines 25-26). It then updates the selected nodes (line 30). We terminate the iterations if there is no selected node (line 31). It obtains the node ranking by sorting the nodes of $\mathbb{D}_i$ in descending order of their lower estimations (line 32). Finally, it returns $\mathbb{D}_i$ as the set of answer nodes (lines 33-34).

As shown in Algorithm 2, our approach does not need any precomputation unlike the previous approaches. That is, we can effectively handle ad-hoc search. Our algorithm does not require any user-defined inner-parameter. Thus, it provides the user with a simple approach to finding the top-k nodes with enhanced search speed.

4.5 Theoretical analyses

We provide theoretical analyses on the search results and the computation cost of our algorithm. We introduce the following theorem that Castanet finds answer nodes exactly:

Theorem 1 (Exactness in top-k search). *Castanet finds the top-k nodes with exact node ranking.*

Proof Algorithm 2 computes the set of selected nodes until the set converges to an empty set, and it returns $\mathbb{D}_i$ as the answer nodes. If $\mathbb{S}_{i+1} = \emptyset$, we have $\mathbb{L}_i = \mathbb{D}_i$ from Definition 5. Since $\mathbb{L}_i \supseteq \mathbb{A}$ and $\mathbb{D}_i \subseteq \mathbb{A}$ from Lemma 5, 6, 7, and 8, we have $\mathbb{L}_i = \mathbb{D}_i = \mathbb{A}$ if $\mathbb{L}_i = \mathbb{D}_i$ holds. Therefore, we have $\mathbb{D}_i = \mathbb{A}$ if $\mathbb{S}_{i+1} = \emptyset$ holds. Since $\mathbb{D}_i = \mathbb{A}$, the ranking of all nodes in $\mathbb{D}_i$ are fixed after the iterations. Therefore, we can obtain the top-k nodes with exact node ranking. □

We discuss the time complexity of our approach. Let l and t be the average number of selected nodes and the number of iterations in our approach, respectively. And let n

Algorithm 2 Castanet

```
Input: $G$, original graph; $c$, the scaling parameter; $k$, number of
    answer nodes; $\mathbb{Q}$, set of query nodes
Output: $\mathbb{A}$, set of answer nodes
 1: $i := 0$;
 2: $\mathbb{S}_0 := \mathbb{V}$;
 3: repeat
 4:     if $i \neq 0$ then
 5:         $i := i + 1$;
 6:     end if
 7:     compute $\mathbb{h}_i$ and $\mathbb{e}_i$ by breadth-first search;
 8:     compute $G_i$ by Algorithm 1;
 9:     for each node $u \in \mathbb{V}_i$ do
10:         compute $p_i[u]$ from $\mathbb{G}_i$ by Equation (29);
11:     end for
12:     for each node $u \in \mathbb{S}_i$ do
13:         compute $\underline{s}_i[u]$ by Equation (3);
14:         compute $\overline{s}_i[u]$ by Equation (11);
15:     end for
16:     if $i = 0$ then
17:         compute $\mathbb{L}_i$ by Equation (19);
18:     else
19:         if $|\mathbb{L}_{i-1}| \neq k$ then
20:             compute $\mathbb{L}_i$ by Equation (24);
21:         else
22:             $\mathbb{L}_i := \mathbb{L}_{i-1}$;
23:         end if
24:     end if
25:     if $|\mathbb{L}_i| = k$ then
26:         compute $\mathbb{D}_i$ by Equation (20);
27:     else
28:         $\mathbb{D}_i := \emptyset$;
29:     end if
30:     $\mathbb{S}_{i+1} := \mathbb{L}_i \backslash \mathbb{D}_i$;
31: until $\mathbb{S}_{i+1} = \emptyset$
32: sort nodes of $\mathbb{D}_i$;
33: $\mathbb{A} := \mathbb{D}_i$;
34: return $\mathbb{A}$;
```

and m be the average numbers of nodes and edges of the subgraphs, respectively. Note that, the original iterative approach requires $O(N + M)T$ time.

Theorem 2 (Computational cost). *Castanet requires $O((l + n + m + k \log k)t + N + M)$ time to find top-k nodes.*

Proof Our approach first constructs the subgraph which takes $O((n+m)t+N+M)$ time. This is because (1) nodes and edges in the subgraph of each iteration are computed by breadth-first search in $O((n + m)t)$ time, and (2) $\mathbb{h}_i$ and $\mathbb{e}_i$ are obtained in $O(N + M)$ time. It takes $O((n + m)t)$ and $O(l \cdot t)$ time to compute random walk probabilities and the lower/upper estimations from the subgraphs in the iterations, respectively. We can compute node sets $\mathbb{L}_i$ and $\mathbb{D}_i$ within $O(l \cdot t)$ and $O(k \log k \cdot t)$ time, respectively. After the iterations, we sort answer nodes to compute the node ranking which takes $O(k \log k)$ time. As a result, our approach requires $O((l + n + m + k \log k)t + N + M)$ time. □

As shown in the next section, $k \ll l \ll n \ll N$, $m \ll M$, and $t \ll T$ in practice. The original iterative approach needs $O((N+M)T)$ time; the proposed approach can find the top-k nodes much faster than the original iterative approach.

5. EXPERIMENTAL EVALUATION

We performed experiments to compare Castanet to the Monte Carlo-based approach proposed by Avrachenkov et al. [2], the matrix-based approach proposed by Fujiwara et al. [16], and the original iterative approach [29]. Avrachenkov et al. proposed two Monte Carlo-based approaches, MC Complete Path and MC End Point, both approximately find the top-k nodes in ad-hoc style unlike other approaches such as [4] and [12]. As described in their paper, MC Complete Path is faster than MC End Point. This is because MC Complete Path theoretically requires, approximately, c times fewer random walks than MC End Point as described in Section 2. Therefore, we selected MC Complete Path as a benchmark. The matrix-based approach by Fujiwara et al. is a state-of-the-art approach and can find top-k nodes exactly and efficiently by pruning unnecessary nodes. Their approach does not have the limitations of other matrix-based approaches such that (1) the exact top-k nodes are not guaranteed in the search results [33], or (2) only a single node is permitted as the query node [14]. Our experiments will demonstrate that:

- Efficiency: Castanet is much faster than the previous approaches for the real datasets tested (Section 5.1).
- Exactness: Unlike the previous approximate approach, the proposed approach can find the top-k nodes with exact ranking (Section 5.2).
- Applicability: Our approach can handle arbitrarily set the scaling parameters since it does not need precomputation (Section 5.3).

The experiments used the following four public datasets:

- Notredame[4]: This dataset was taken from the web pages of the University of Notre Dame[5]. In this graph, nodes represent pages from domain nd.edu and edges represent hyperlinks between them. There are 325,729 nodes and 1,497,135 edges.
- CNR[6]: CNR (Italy's National Research Council)[7] is a public research organization. This graph is the result of a crawl of the Italian CNR domain where nodes and edges correspond to pages and hyperlinks, respectively. This graph has 325,557 nodes and 3,216,152 edges.
- Email[8]: This is an email dataset from an European research institution. In this graph, each node corresponds to an email address. A directed edge between nodes u and v indicates that the owner of address u sent at least one message to address v. This dataset has 265,214 nodes and 420,045 edges.
- Social[9]: Slashdot[10] is a famous news website known for its specific user community. The graph is taken from Slashdot.org and contains interaction links among the users of Slashdot. The numbers of nodes and edges are 82,144 and 549,202, respectively.

In this section, "Castanet", "Monte", "Matrix", and "Original" represent the results of our approach, the Monte Carlo-based approach by Avrachenkov et al., the matrix-based approach by Fujiwara et al., and the original iterative approach, respectively. We randomly set three nodes as query nodes[11]. The preference scores were set to the same value. All experiments were conducted on a Linux quad 3.33 GHz Intel Xeon server with 32GB of main memory. We implemented all approaches using GCC.

[4] http://vlado.fmf.uni-lj.si/pub/networks/data/ND/NDnets.htm
[5] http://www.nd.edu/
[6] http://law.di.unimi.it/webdata/cnr-2000/
[7] http://www.cnr.it/sitocnr/home.html
[8] http://snap.stanford.edu/data/email-EuAll.html
[9] http://snap.stanford.edu/data/soc-sign-Slashdot090221.html
[10] http://slashdot.org/
[11] In word sense disambiguation, the number of query nodes is 2.2 on average as described in Section 1.

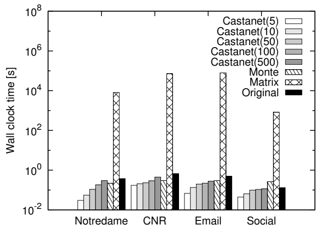


Figure 1: Search time of each approach.

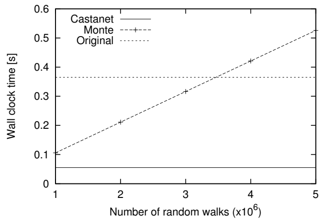


Figure 2: Efficiency versus number of random walks.

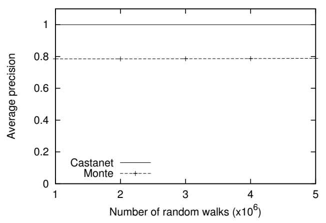


Figure 3: Accuracy versus number of random walks.

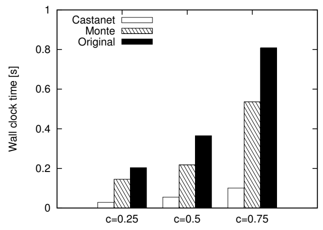


Figure 4: Efficiency versus the scaling parameters.

Table 2: Score of each parameter.

Parameter	Dataset			
	Notredame	CNR	Email	Social
N	3.25×10^5	3.25×10^5	2.65×10^5	8.21×10^4
l	2.29×10^4	2.13×10^4	2.27×10^4	7.92×10^3
n	5.12×10^4	7.01×10^4	5.98×10^4	2.88×10^4
M	1.49×10^6	3.21×10^6	4.20×10^5	5.49×10^5
m	7.30×10^4	5.70×10^5	9.76×10^4	2.17×10^5
T	30.8	32.8	56.0	28.0
t	15.5	24.3	15.7	13.1

Table 3: Breakdown of search time.

Approach	Search time [ms]		
	Precomputation	Top-k search	Overall
Castanet	–	5.52	5.52
Monte	–	21.8	21.8
Matrix	8.16×10^5	5.60×10^{-3}	8.16×10^5
Original	–	36.5	36.5

5.1 Efficiency of Castanet

We evaluated the search time of each approach. Figure 1 shows the results. In this figure, the results of the proposed approach are indicated by "Castanet(k)" where k is the number of answer nodes. We set the scaling parameter, c, to 0.5 and the number of random walks in the Monte Carlo-based approach to $2,500,000$. The impacts of the scaling parameter and the number of random walks are evaluated in later sections. Note that the number of answer nodes has, in the previous approaches, practically no impact on the search time. This is because (1) the similarities of all nodes must be computed to obtain similar nodes in the Monte Carlo-based and the original iterative approaches, and (2) precomputation time is dominant in the search time for the matrix-based approach. Table 2 details the parameters in Castanet and the original iterative approaches where $c = 0.5$ and $k = 10$. Table 3 shows the breakdown in the search time of each approach for Notredame where $c = 0.5$ and $k = 10$.

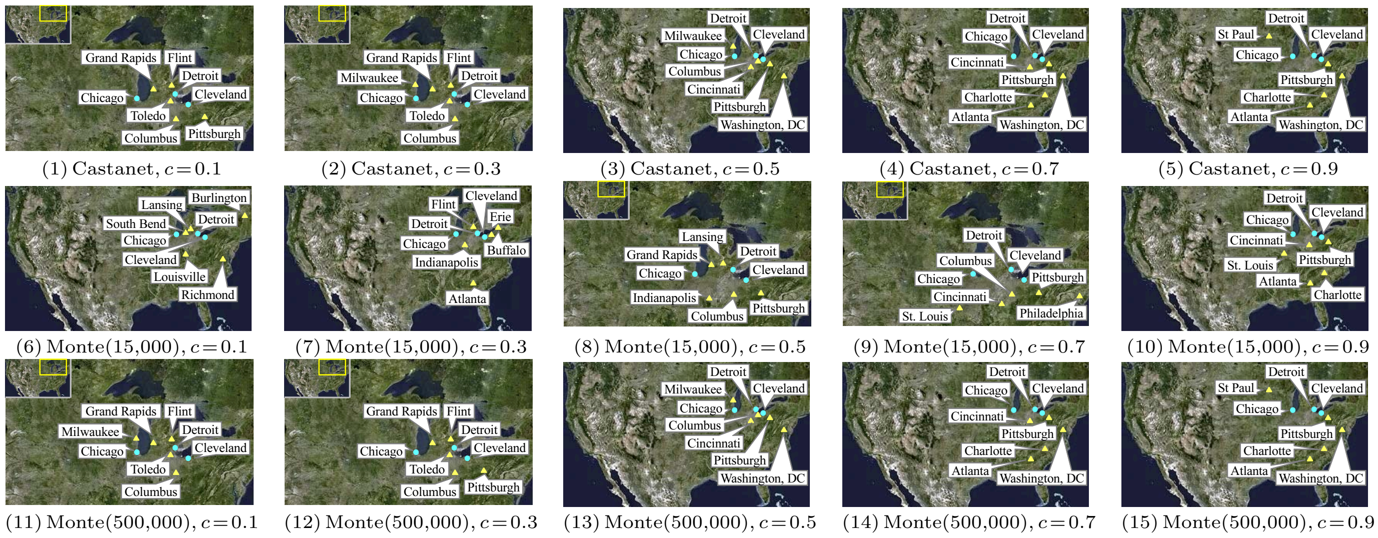
As shown in Figure 1, our approach is much faster than the original iterative approach. For example, our approach can cut the search time from the original iterative approach by up to 90% and 75% for Notredame and CNR, respectively. Note that Notredame and CNR have almost the same number of nodes. The result indicates that efficiency of our approach strengthens with the large effective diameter. The effective diameter is the 90-th percentile of shortest path length distribution [24]. Notredame and CNR have effective diameters of 9.3 and 5.6, respectively. If a graph has large effective diameter, the number of hops in the reachable node set is expected to be large. Therefore, the size of the node set $\mathbb{H}_i$ in the i-th iteration is likely to be small. As shown in Definition 6, the subgraph in the i-th iteration is obtained from the node set $\mathbb{H}_i$. Thus, the size of subgraphs is small for a large effective diameter graph as shown in Table 2, which yields the efficiency of our approach.

The result of Email and Social in Figure 1 indicates that our approach more efficiently find top-k nodes than the original iterative approach as graph size increases. These two graphs have effective diameter of 4.5 and 4.7, respectively; they are almost the same. As shown in Figure 1, our approach can cut the search time by up to 85% and 65% for Email and Social, respectively. This is because, if the given graph has large size, the sizes of subgraphs become relatively small compared to the whole graph.

Figure 1 also indicates that Castanet enhances the efficiency as the number of answer nodes k decreases. This is because the node sets $\mathbb{H}_i$ and $\mathbb{R}_i$ in Definition 6 have small size for small values of k. As shown in this figure, if $k = 500$, the Monte Carlo-based approach can be more efficient than our approach. However, this does not decrease the efficacy of our approach for the applications listed in Section 1. This is because k is typically much smaller than 500. For example, $k = 5$ in web prefetching and automatic image annotation [17, 30] as described in Section 1. Furthermore, the Monte Carlo-based approach does not find the top-k nodes exactly since it is an approximation approach. In addition, our approach is more efficient than the matrix-based approach under all conditions examined. This is because the precomputation time in the matrix-based approach is significantly high even though the top-k search time is small for other approaches as shown in Table 3. As a result, Castanet has superior search performance to the previous approaches.

5.2 Node ranking exactness

One major advantage of Castanet is that it guarantees the exactness of node ranking while the Monte Carlo-based approach does not. Since, as described in Section 2, the Monte Carlo-based approach approximates the similarity of a node by the total number of visits to the node via the random walks, the number of random walks is expected to impact the search time and approximation accuracy for the Monte Carlo-based approach. Therefore, we conducted comparative experiments using various numbers of random walks. Figure 2 and Figure 3 show the search time and the accuracy of each approach for Notredame, respectively, where $c = 0.5$ and $k = 10$. In Figure 3, we used average precision as the metric of accuracy; it is widely used to measure the accuracy of ranking models (see [7] for the definition of average precision.). Average precision outputs values between 0 and 1. Average precision is high as the ranking by Castanet or the Monte Carlo-based approach is similar to that by the original iterative approach; if average precision is 1, the ranking by Castanet or the Monte Carlo-based approach equals to that of the original iterative approach.



(1) Castanet, $c=0.1$ (2) Castanet, $c=0.3$ (3) Castanet, $c=0.5$ (4) Castanet, $c=0.7$ (5) Castanet, $c=0.9$

(6) Monte(15,000), $c=0.1$ (7) Monte(15,000), $c=0.3$ (8) Monte(15,000), $c=0.5$ (9) Monte(15,000), $c=0.7$ (10) Monte(15,000), $c=0.9$

(11) Monte(500,000), $c=0.1$ (12) Monte(500,000), $c=0.3$ (13) Monte(500,000), $c=0.5$ (14) Monte(500,000), $c=0.7$ (15) Monte(500,000), $c=0.9$

Figure 5: High similarity cities for Chicago, Detroit, and Cleveland as detected by each approach.

As shown in Figure 2, the Monte Carlo-based approach takes more computation time as the number of random walks increases. If the number of random walks is not properly set (e.g., $4,000,000$ or $5,000,000$), the Monte Carlo-based approach takes more computation time than the original iterative approach. Figure 3 shows that the average precision of Castanet is 1 since it finds top-k nodes with exact node ranking (Theorem 1). On the other hand, the Monte Carlo-based approach has lower average precision. Figure 2 and Figure 3 confirm that our approach is superior to the Monte Carlo-based approach in both speed and accuracy.

5.3 Flexibility to the scaling parameter

As mentioned in Section 1, interactive similarity search is important in real applications. Our approach can handle arbitrarily set the scaling parameters since it does not need precomputation as described in Section 4.4. To confirm this benefit, we evaluated the search time of each approach using the various scaling parameters. Figure 4 shows the results. In these experiments, we set $k = 10$ and used Notredame as the dataset. We omitted the results of the matrix-based approach since Figure 1 shows that the approach has significantly high search time than other approaches.

As shown in Figure 4, decreasing the score of the scaling parameter raises the efficiency of each approach in finding the answer nodes. This is explained below. Our approach updates the set of nodes subjected to lower/upper estimation in the each iteration. As shown in Equation (11), the upper estimation is obtained by adding $c^{i+1}W_{max}[u]\, p_i[\mathbb{R}_i]$ to the lower estimation. Therefore, if c is low, the upper estimation is expected to have a low score; this implies that we can more accurately compute the upper estimation if c is assigned a low score. As described in Section 2 , to obtain approximate similarities, the Monte Carlo-based approach computes random walks where probability c is used to select an outgoing edge from a current node at every step. This indicates that the random walk length is expected to be long if c is high. The computation time of the original iterative approach increases with the score of the scaling parameter, because the number of iterations required by the approach increases as c approaches to 1 as described in [23]. However, our approach can more efficiently find the answer nodes than the previous approaches; this implies that the proposed approach effectively supports interactive similarity search where the scaling parameter is set in ad-hoc style.

Table 4: Search time of each approach.

Approach	Search time [ms]				
	$c=0.1$	$c=0.3$	$c=0.5$	$c=0.7$	$c=0.9$
Castanet	1.18	2.02	3.27	5.76	8.78
Monte(15, 000)	1.21	2.36	4.52	8.94	29.6
Monte(500, 000)	33.8	79.0	149.5	296.6	986.5

6. CASE-STUDIES

To enhance the quality of applications, it is important to iteratively find the top-k nodes for an arbitrary graph in an ad-hoc manner. Our approach allows users to adjust the scaling parameter for a given graph on the fly. To validate our approach, we conducted case-studies on a real dataset using the various scaling parameters.

We used the city dataset[12] consisting of $71,959$ direct flights between 456 cities in North America. Flights and cities correspond to edges and nodes, respectively. Edge weights are proportional to city proximities in terms of flight times. We detected five cities with high similarity, for Chicago, Detroit, and Cleveland, these three cities surround the Great Lakes. The preference scores were set to the same value. Figure 5 shows the results of our approach and the Monte Carlo-based approach. In these plots, circles and triangles represent query cities and high similarity cities, respectively. Note that our approach output the same results as the matrix-based approach and the original iterative approach, so we omit the results of these two approaches due to space limitations. Table 4 shows the search time of each approach. For the Monte Carlo-based approach, the number of random walks was set to $15,000$ and $500,000$. The results of these settings are referred as “Monte(15,000)” and “Monte(500,000)” in Figure 5 and Table 4.

If $c = 0.1$, our approach identified Pittsburgh, Columbus, Toledo, Flint, and Grand Rapids as high similarity cities. All these cities are also from around the Great Lakes. Since PPR computes random walks from the query nodes, major

[12]http://www.psi.toronto.edu/index.php?q=affinity propagation

cities surrounding the Great Lakes are detected as the results. On the other hand, if the scaling parameter is high (e.g., $c = 0.7$ or $c = 0.9$), cities distant from the Great Lakes such as Washington, D.C. or Atlanta were detected as high similarity cities. This is because (1) the random walk length is expected to be long if c is high as described in Section 5.3, and (2) highly connected nodes from other nodes are expected to be high similarity nodes in PPR [20]. As a result, large cities that have big airline hubs are detected as answer nodes for the large scaling parameters.

These results indicate that a user should be recommended to initially set c to 0.5. If more locally-oriented results are required, (1) the scaling parameter should be reduced (e.g., $c = 0.1$), and (2) the graph should be modified by employing locally-oriented edge weights such as geographical distances instead of flight times. Using road network, not airline network, is also preferable for obtaining locally-oriented results. If the user wants to extract large cities, the large scaling parameters are recommended where edge weighs are set in proportion to the number of passengers, which increases the edge weights to large cities. Since Castanet can efficiently find top-k nodes for an arbitrarily given graph with the flexible scaling parameter setting, it can support such interactive actions of the user to enhance application efficacy.

However, as shown in Figure 5, the results of the Monte Carlo-based approach are different from those of Castanet if the number of random walks is $15,000$; the results of the Monte approach contain many improper cities even though the Monte Carlo-based approach and Castanet have almost the same search time with this setting as shown in Table 4. If the number of random walks is increased to $500,000$, the Monte Carlo-based approach has the almost same results as Castanet as shown in Figure 5. However, this leads to high search times as shown in Table 4 since the search cost of the Monte Carlo-based approach is proportional to the number of random walks as described in Section 5.2.

7. CONCLUSIONS

We addressed the problems of efficiently finding the top-k nodes for PPR. Our solution, Castanet, is designed to support interactive similarity search and based on the idea to use lower/upper estimations and prune unnecessary nodes in the iterations. The experimental results show that our approach can achieve high efficiency while ensuring that the answer results offer exact node ranking. PPR is fundamental in many applications. Our approach allows many applications to be implemented more efficiently, and helps to improve the effectiveness of future applications.

8. REFERENCES

[1] E. Agirre and A. Soroa. Personalizing PageRank for Word Sense Disambiguation. In *EACL*, pages 33–41, 2009.

[2] K. Avrachenkov, N. Litvak, D. Nemirovsky, E. Smirnova, and M. Sokol. Quick Detection of Top-k Personalized PageRank Lists. In *WAW*, pages 50–61, 2011.

[3] B. Bahmani, K. Chakrabarti, and D. Xin. Fast Personalized PageRank on MapReduce. In *SIGMOD Conference*, pages 973–984, 2011.

[4] B. Bahmani, A. Chowdhury, and A. Goel. Fast Incremental and Personalized PageRank. *PVLDB*, 4(3):173–184, 2010.

[5] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based Deyword Search in Databases. In *VLDB*, pages 564–575, 2004.

[6] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. A. Grossman, and O. Frieder. Hourly Analysis of a Very Sarge Topically Categorized Web Query Log. In *SIGIR*, pages 321–328, 2004.

[7] B. Carterette, J. Allan, and R. K. Sitaraman. Minimal Test Collections for Retrieval Evaluation. In *SIGIR*, pages 268–275, 2006.

[8] S. Chakrabarti, A. Pathak, and M. Gupta. Index Design and Query Processing for Graph Conductance Search. *VLDB J.*, 20(3):445–470, 2011.

[9] J. Cho and U. Schonfeld. RankMass Crawler: A Crawler with High PageRank Coverage Guarantee. In *VLDB*, pages 375–386, 2007.

[10] J. Domènech, J. A. Gil, J. Sahuquillo, and A. Pont. Using Current Web Page Structure to Improve Prefetching Performance. *Computer Networks*, 54(9):1404–1417, 2010.

[11] M. Eirinaki and M. Vazirgiannis. Usage-based PageRank for Web Personalization. In *ICDM*, pages 130–137, 2005.

[12] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós. Towards Scaling Fully Personalized PageRank: Algorithms, Lower Bounds, and Experiments. *Internet Mathematics*, 2(3), 2005.

[13] Y. Fujiwara, G. Irie, and T. Kitahara. Fast Algorithm for Affinity Propagation. In *IJCAI*, pages 2238–2243, 2011.

[14] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa. Fast and Exact Top-k Search for Random Walk With Restart. *PVLDB*, 5(5):442–453, 2012.

[15] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka. Efficient Search Algorithm for SimRank. In *ICDE*, 2013.

[16] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka. Efficient Personalized PageRank with Accuracy Assurance. In *KDD*, 2012.

[17] Y. Z. Guo, K. Ramamohanarao, and L. A. F. Park. Personalized PageRank for Web Page Prediction Based on Access Time-length and Frequency. In *Web Intelligence*, pages 687–690, 2007.

[18] M. S. Gupta, A. Pathak, and S. Chakrabarti. Fast Algorithms for Top-k Personalized PageRank Queries. In *WWW*, pages 1225–1226, 2008.

[19] D. A. Harville. *Matrix Algebra From a Statistician's Perspective*. Springer, 2008.

[20] G. Jeh and J. Widom. Scaling Ppersonalized Web Search. In *WWW*, pages 271–279, 2003.

[21] R. Jin, N. Ruan, S. Dey, and J. X. Yu. SCARAB: Scaling Reachability Computation on Large Graphs. In *SIGMOD Conference*, pages 169–180, 2012.

[22] U. Kang, D. H. Chau, and C. Faloutsos. Mining Large Graphs: Algorithms, Inference, and Discoveries. In *ICDE*, pages 243–254, 2011.

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

[24] J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *KDD*, pages 177–187, 2005.

[25] R. Mihalcea. Unsupervised Large-vocabulary Word Sense Disambiguation with Graph-based Algorithms for Sequence Data Labeling. In *HLT/EMNLP*, 2005.

[26] M. Nakatsuji, Y. Fujiwara, T. Uchiyama, and K. Fujimura. User Similarity from Linked Taxonomies: Subjective Assessments of Items. In *IJCAI*, pages 2305–2311, 2011.

[27] M. Nakatsuji, Y. Fujiwara, T. Uchiyama, and H. Toda. Collaborative Filtering by Analyzing Dynamic User Interests Modeled by Taxonomy. In *ISWC*, pages 361–377, 2012.

[28] R. Navigli. Word Sense Disambiguation: A Survey. *ACM Comput. Surv.*, 41(2), 2009.

[29] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[30] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic Multimedia Cross-modal Correlation Discovery. In *KDD*, pages 653–658, 2004.

[31] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. LabelMe: A Database and Web-based Tool for Image Annotation. *International Journal of Computer Vision*, 77(1-3):157–173, 2008.

[32] H. Shiokawa, Y. Fujiwara, and M. Onizuka. Fast Algorithm for Modularity-based Graph Clustering. In *AAAI*, 2013.

[33] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast Random Walk with Restart and Its Applications. In *ICDM*, pages 613–622, 2006.

[34] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with Local and Global Consistency. In *NIPS*, 2003.