INVITED PAPER *Special Section on Award-winning Papers*

Fast Ad-Hoc Search Algorithm for Personalized PageRank

Yasuhiro FUJIWARA[†a)], **Makoto NAKATSUJI**[††], ***Members***, **Hiroaki SHIOKAWA**[†††],
Takeshi MISHIMA[†], ***Nonmembers***, ***and*** **Makoto ONIZUKA**[††††], ***Member***

SUMMARY *Personalized PageRank (PPR)* is a typical similarity metric between nodes in a graph, and node searches based on PPR are widely used. In many applications, graphs change dynamically, and in such cases, it is desirable to perform ad hoc searches based on PPR. An ad hoc search involves performing searches by varying the search parameters or graphs. However, as the size of a graph increases, the computation cost of performing an ad hoc search can become excessive. In this paper, we propose a method called *Castanet* that offers fast ad hoc searches of PPR. The proposed method features (1) iterative estimation of the upper and lower bounds of PPR scores, and (2) dynamic pruning of nodes that are not needed to obtain a search result. Experiments confirm that the proposed method does offer faster ad hoc PPR searches than existing methods.

key words: *Personalized PageRank, ad-hoc, fast, algorithm*

1. Introduction

In recent years, there has been growing interest in mining the large-scale graphs that are typically found in social networks. Since the publication of PageRank [1], researchers have studied approaches to compute node importance based on graph link structures. This is because node importance has many important applications such as classifying and analyzing graphs in a graph database [2]. Many methods that are based on the idea of PageRank have already been proposed, of which *Personalized PageRank (PPR)* is the most popular measure for computing node similarity [3]. A PPR similarity score corresponds to the probability of being in a steady state after a random walk through a graph where the query node corresponds to the starting point. Although PPR can be determined by iterative computation, the probability c (called the scaling parameter) is used in this computation; it is the probability that the random walk will return to the query node. In many applications, it is preferable to perform PPR searches in an ad hoc manner on arbitrary graphs. An ad hoc search involves performing searches while varying the search parameters or graphs each time a search is performed. There are two reasons for this. The first is that the graph structure is not clear until the the query node is known due to the property that graphs change dynamically in many applications. The second is that scaling parameter c affects the PPR similarity score, so the suitable scaling parameter varies from one application to another [4]. Since the computation cost of PPR is $O((N+M)T)$ where N, M, and T are the numbers of nodes, edges, and iterations, respectively, the computation time needed to perform an ad hoc search in a large-scale graph can become excessive.

In this paper, we propose a method called *Castanet* that identifies the top-k nodes with exact node ranking. In order to reduce the search cost, our method utilizes the following two approaches: (1) Iteratively calculate upper/lower PPR scores and (2) Prune nodes that are unnecessary for the top-k search at each iteration. We have confirmed that the method proposed in this paper is more effective than the conventional method. This paper is structured as follows: First, related studies are discussed in Sect. 2. The background to our study is then presented in Sect. 3. In Sect. 4, we describe the proposed method, and we discuss the results of our tests in Sect. 5. A summary of our work is presented in Sect. 6.

2. Related Work

Previous PPR studies can be split into two categories: matrix-based and Monte Carlo-based approaches. Tong et al. studied two matrix-based approaches called B_LIN and NB_LIN [5]. These approaches calculate approximate PPR scores by eigenvalue decomposition. Fujiwara el al. proposed fast methods for searching PPR by using LU decomposition and QR decomposition [6], [7]. Unlike the method of Tong et al., these methods have the merit of being able to perform accurate searches on PPR. However, these matrix decomposition methods require matrix decomposition to be performed before searching, so they are not suitable for ad hoc searches.

Fogaras et al. proposed a method that uses a Monte Carlo-based approach to realize fast computation [8]. To precompute approximate PPR scores, they used “fingerprints” obtained by random walks starting from the query node. During a search, the PPR scores are approximated by using the distribution of end points in precomputed fingerprints. Bahmani also proposed using precomputed random walks in a similar way [9]. Avrachenkov et al. proposed a Monte Carlo-based MC Complete Path method for fast PPR search where MC is an acronym for Monte Carlo [12]. In

Manuscript received October 28, 2016.
Manuscript revised December 26, 2016.
Manuscript publicized January 23, 2017.

[†] The authors are with NTT Software Innovation Center, Musashino-shi, 180–8585 Japan.

[††] The author is with NTT Resonant, Tokyo, 108–0023 Japan.

[†††] The author is with University of Tsukuba, Tsukuba-shi, 350–8573 Japan.

[††††] The author is with Osaka University, Suita-shi, 565–0871 Japan.

a) E-mail: fujiwara.yasuhiro@lab.ntt.co.jp

DOI: 10.1587/transinf.2016AWI0002

this method, approximate PPR scores are obtained quickly by computing ad hoc random walks when a search is performed. However, in Monte Carlo-based approaches, the precision depends on the number of random walks, which induces a trade-off between speed and search precision.

We previously proposed a fast PageRank search method called F-Rank [10]. The method we are proposing here differs from F-Rank in that the proposed method searches for the top-k nodes while performing accurate node ranking. That is, although F-Rank performs a top-k search, F-Rank does not perform node ranking in the search results. The proposed method also differs from F-Rank in that the proposed method reduces the number of nodes involved in the computation by computing partial graphs based on the number of hops from the query node during the iterative computation. Although Personalized PageRank differs from PageRank in that nodes are ranked based on the query node, the proposed method uses this property to facilitate fast searching.

3. Preliminary

In this section, we define the notations used in this paper and explain the background knowledge needed to understand our approach. Table 1 lists the main symbols and their definitions. PPR computes similarities based on a “random surfer model” similar to approach taken by Google’s PageRank algorithm. PageRank exploits a graph’s link structure to compute the importance of nodes in the entire Web graph. PPR, introduced by Jeh et al., is based on the idea that global node importance scores can be tailored for each user [3]. Specifically, a set of nodes is defined for each user, and the node importance scores are computed based on priority values that are arbitrarily set for each node. The sum of these node priority values is normalized to 1. If a set of nodes is taken as the query nodes, then the importance score of each node can be considered as its similarity to the query node. Put simply, the similarity scores in PPR correspond to the steady-state probabilities of random walks. At each iteration of PPR, one of the nodes adjacent to the current node is selected with probability c, and, with restart probability $1 - c$, it jumps to a query node in accordance with its priority value. If sets V and E hold the nodes and edges of the queried graph, respectively, then the graph is represented as $G = \{V, E\}$. Also, if $\mathbf{s}$ is an $N \times 1$ PPR score vector, then the u-th element of this vector $s[u]$ denotes the PPR score of node u. $\mathbf{W}$ is an adjacency matrix of the graph where the sum of each column is normalized to 1, and each element $W[u, v]$ gives the probability of transitioning from node v to node u in a random walk. Specifically, the columns and rows of the adjacency matrix $\mathbf{W}$ correspond to the start and end of each edge respectively. The PPR scores can be obtained by calculating the following expression recursively.

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

where $\mathbf{q}$ represents a query node vector whose u-th element $q[u]$ corresponds to its preference score as a query node. In other words, if node u is not a query node, then $q[u] = 0$. The query node preference scores are normalized such that their sum is 1.

Table 1 Main symbols and their definitions.

Symbol	Definition
G	The graph being queried
N	Number of nodes in the graph
M	Number of edges in the graph
T	Number of iterations in the original method
c	Scaling parameter, $0 < c < 1$
k	Number of required answer nodes
V	Set of nodes in G
E	Set of edges in G
Q	Set of query nodes
S	Set of selected nodes
R	Set of nodes from which the selected nodes are reachable
L	Set of candidate nodes
D	Set of nodes whose ranking has been determined
A	Set of answer nodes
$C[u]$	Set of nodes that have edges incident to node u
$\mathbf{q}$	$N \times 1$ query node vector
$\mathbf{s}$	$N \times 1$ vector of similarity based on PPR
$\mathbf{W}$	$N \times N$ graph adjacency matrix with normalized columns

However, this original recursive computation method is not suitable when searching for the top-k nodes. This is because the scores of all nodes have to be updated at each iteration. Furthermore, to obtain the exact ranking of the top-k nodes, the exact scores of all nodes must be obtained by performing iterative computation until the scores converge, even though exact scores are not always essential for ranking in most applications. The original method has a computation cost of $O((N + M)T)$, making it difficult to perform fast searches on large-scale graphs. A faster search method is therefore required.

4. Proposed Method

In this section, we first present an overview of the proposed method, and then describe it in detail.

4.1 Overview

As described in Sect. 3, the original method requires that probabilities be calculated as steady state probabilities. This approach incurs high computation cost because it has to iteratively calculate scores for the whole graph. To find the top-k nodes quickly, our proposed method recursively updates the estimated similarity values of only selected nodes instead of all nodes at each step. Thus, the proposed method offers high-speed processing as it avoids iteratively processing the whole graph to find the top-k nodes.

To rapidly update the estimated similarity values, the proposed method dynamically configures a subgraph by pruning unnecessary nodes and edges from the entire graph. The random walk probabilities that are needed to compute the estimated values can be calculated from the subgraphs, so the similarity values can be updated at high speed in the proposed method. To identify unnecessary nodes and edges, the proposed method estimates the upper and lower bounds of the similarity values. The subgraphs obtained by iterative computation are smaller than the whole graph, so the answer nodes can be searched at greater speed.

This subgraph approach has the advantage of requiring fewer iterations than the original iterative approach. The original iterative approach has to perform iterative computation until the similarities converge in order to compute exact similarities. In contrast, the proposed method can determine the upper and lower bounds of the node ranking so that there is no need to perform iterative computation on nodes whose ranking has been determined. Therefore, if the ranking of all nodes in the graph is determined from the upper and lower bounds in the proposed method, then the iterative computation can be omitted. The search process is thus terminated without waiting for the similarity values to converge, and as a result our method needs fewer iterations than the original method.

Another advantage of our approach is that it can automatically determine the structures of subgraphs and the number of iterations. Since the proposed method does not require any internal parameters to be set, users can easily perform PPR searches.

4.2 Similarity Estimation

The method used to calculate estimated similarity values is discussed below. First, we introduce the notation used in computing the estimated values, and then we discuss the specific computation method and the theoretical aspects of this method.

4.2.1 Notation

To obtain the estimated values, we use probability $p_i[u]$ that a random walk of length i starting from a query node will end up at node u. In this random walk, we do not jump to a query node with probability $1 - c$. Here, $\mathbf{p}_i$ is an $N \times 1$ vector whose u-th element corresponds to $p_i[u]$. $\mathbf{p}_i$ can be calculated from the i-th power of the adjacent matrix as $\mathbf{p}_i = \mathbf{W}^i\mathbf{q}$. Since it is clear that $\mathbf{p}_i = \mathbf{W}\mathbf{p}_{i-1}$, we can incrementally compute the probability $\mathbf{p}_i$ of a random walk of length i from $\mathbf{p}_{i-1}$ as follows:

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

where $C[u]$ is the set of start nodes of edges that are incident to node u, i.e., $C[u]$ is the set of nodes directly adjacent to node u in the original graph for which u is the end point. In the i-th iteration ($i = 0, 1, 2, \ldots$), the proposed method computes a set of nodes S_i for which we update the upper and lower similarity bounds. The set of selected nodes is initialized as the set of all nodes in the original graph, i.e., $S_0 = V$, and the nodes in S_i are set so that they are always included in S_{i-1}. Therefore, $V = S_0 \supseteq S_1 \supseteq \ldots \supseteq S_i$. The specific details of the node selection method are discussed later. To compute the upper similarity bound, we use the node set R_i, which is the set of nodes from which it is possible to reach any node in set S_i. Node u is able to reach node v if there is a path from u to v in the original graph. Since node u can be reached from itself, it is clear that $S_i \subseteq R_i$, and if $u \in S_i$ then $C[u] \subseteq R_i$. Also, $p_i[R_i]$ indicates the probability that a random walk of length i that starts at a query node will reach a node in R_i, i.e., $p_i[R_i] = \sum_{u \in R_i} p_i[u]$. To calculate the upper limit, we use $W_{max}[u]$, which is the maximum weighting of edges incident to node u, i.e., $W_{max}[u] = \max\{W[u, v] : v \in V\}$.

4.2.2 Definition of Similarity

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

Definition 1 (Lower bound): The lower bound of the PPR score of node u at the i-th iteration, $\underline{s}_i[u]$, is calculated 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 will show in the next paragraph that Eq. (3) has the lower bounding property. This definition implies that (1) if $i = 0$, then the lower bound can be computed from the random walk probability and the scaling parameter, and (2) otherwise, the lower bound $\underline{s}_i[u]$ can be incrementally computed from the random walk probability and the scaling parameter. This definition also indicates that we can compute the lower bound of a node in $O(1)$ time if the random walk probability $p_i[u]$ has already been calculated.

We introduce the following lemma to demonstrate the lower bound property.

Lemma 1 (Lower bound): The relationship $\underline{s}_i[u] \leq s[u]$ holds for any node in the set S_i.

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 $(c\mathbf{W})^i$ becomes a zero matrix after convergence. Since $0 < c < 1$, it is clear that $\lim_{i\to\infty} c^i = 0$. Since matrix $\mathbf{W}$ is an adjacency matrix of the graph with normalized columns, its i-th power $\mathbf{W}^i$ corresponds to the probabilities of random walks of length i, so 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 Eq. (1), PPR similarity 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 $\mathbf{I} - c\mathbf{W}$. Since $\lim_{i\to\infty}(c\mathbf{W})^i = \mathbf{0}$, the following relation holds if $(c\mathbf{W})^0 = \mathbf{I}$ [11].

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

From Eqs. (5) and (6), we have

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

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

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

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

$$\begin{aligned}\underline{s}_i[u] &= \underline{s}_{i-1}[u]+(1-c)c^{i}p_i[u]\\ &= \underline{s}_{i-2}[u]+(1-c)c^{i}p_i[u]+(1-c)c^{i-1}p_{i-1}[u]\\ &= (1-c)c^{i}p_i[u]+\ldots+(1-c)c^{0}p_0[u]\end{aligned} \tag{9}$$

Therefore, since $c > 0$, $1-c > 0$ and $p_j[u] \geq 0$,

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

which completes the proof. □

We can use the lower similarity bound introduced in Definition 1 to calculate the upper similarity bound. The upper bound is defined below.

Definition 2 (Upper bound): The upper bound $\overline{s}_i[u]$ of the PPR score of node u at the i-th iteration is defined as follows:

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

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

Lemma 2 (Upper bound): The relation $\overline{s}_i[u] \geq s[u]$ holds for all nodes in the set S_i.

Proof To prove this lemma, we first use mathematical induction to show that the property $p_{i+j}[R_{i+j}] \leq p_i[R_i]$ ($j = 0, 1, \ldots$) holds in all iterations.

If $j=0$, then obviously $p_{i+j}[R_{i+j}]=p_i[R_i]$. Also, if $j\neq 0$ (i.e., $j \geq 1$), then assume that $p_{i+j-1}[R_{i+j-1}] \leq p_i[R_i]$ holds. Since nodes are selected so as to satisfy $S_{i+j} \subseteq S_{i+j-1}$, it is clear that $R_{i+j} \subseteq R_{i+j-1}$. Therefore, from Eq. (2), we have

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

This is because $C[v] \subseteq R_{i+j} \subseteq R_{i+j-1}$ for node v such that $v \in R_{i+j}$, and $\mathbf{W}$ is the adjacency matrix of the graph with normalized columns. It should be noted that $C[v] \subseteq R_{i+j-1}$ holds for node v such that $v \in R_{i+j}$ and $v \notin S_{i+j}$. This is because $C[v] \subseteq C[v] + S_{i+j} \subseteq R_{i+j} \subseteq R_{i+j-1}$. Therefore, $p_{i+j}[R_{i+j}] \leq p_{i+j-1}[R_{i+j-1}] \leq p_i[R_i]$. This completes the inductive step.

Next, we will demonstrate that Lemma 2 holds. From Eqs. (2), (8) and (10), the PPR similarity $s[u]$ of node u can be computed as follows:

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

Here, $W[u,v] \leq W_{max}[u]$, and for node u such that $u \in S_i$, we have $C[u] \subseteq R_{j-1}$, hence

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

Since $\sum_{j=i+1}^{\infty}c^{j}p_{j-1}[R_{j-1}] = \sum_{j=0}^{\infty}c^{i+j+1}p_{i+j}[R_{i+j}]$ and, as described above, $p_{i+j}[R_{i+j}] \leq p_i[R_i]$,

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

which completes the proof. □

As shown in Definitions 1 and 2, the lower and upper bounds are estimated from random walk probabilities at each iteration. A property of these estimated values is that their accuracy increases with each iteration. We demonstrate this property by introducing the following lemma:

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

Proof We will first show that $\underline{s}_i[u] \geq \underline{s}_{i-1}[u]$. From Eq. (3), it is clear that $\underline{s}_i[u] - \underline{s}_{i-1}[u] = (1-c)c^{i}p_i[u] \geq 0$. We will now show that $\overline{s}_i[u] \leq \overline{s}_{i-1}[u]$. From Eqs. (3) and (11), $\overline{s}_i[u] - \overline{s}_{i-1}[u]$ can be computed as follows:

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

From Eq. (2),

$$\begin{aligned}p_i[u] &= \sum_{v\in C[u]} W[u,v]\,p_{i-1}[v] \leq W_{max}[u]\sum_{v\in R_{i-1}}p_{i-1}[v]\\ &= W_{max}[u]\;p_{i-1}[R_{i-1}]\end{aligned} \tag{17}$$

From Eq. (12), $p_i[R_i] \leq p_{i-1}[R_{i-1}]$. Therefore,

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

which completes the proof. □

As described below, Lemma 3 ensures that the proposed method is able to search for the top-k nodes with an exact ranking without calculating similarities until convergence. The upper and lower bounds have the property of converging on exact similarity scores as shown below.

Lemma 4 (Convergence of estimated values): After convergence, the upper and lower estimated values are the equal to the exact similarities; i.e., $\underline{s}_\infty[u] = \overline{s}_\infty[u] = s[u]$.

Proof We will first show that $\underline{s}_\infty[u] = s[u]$. From Eq. (10), it is clear that $\underline{s}_\infty[u] = (1-c)\sum_{j=0}^{\infty} c^j p_j[u] = s[u]$. Next, we will show that $\overline{s}_\infty[u] = s[u]$. From Eq. (11), $\overline{s}_\infty[u] = \underline{s}_\infty[u] + c^\infty W_{max}[u]\, p_\infty[R_\infty]$. Since $c^\infty = 0$, $0 \leq W_{max}[u] \leq 1$ and $0 \leq p_\infty[R_\infty] \leq 1$, $c^\infty W_{max}[u]\, p_\infty[R_\infty] = 0$. Therefore $\overline{s}_\infty[u] = \underline{s}_\infty[u] = s[u]$, which completes the proof. □

As will be described later, the proposed method can achieve precise ranking according to Lemma 4.

4.3 Subgraph-Based Search

In the proposed method, we dynamically construct subgraphs during iterated computations in order to compute random walk probabilities at high speed. The upper and lower bounds of the selected nodes are computed by using random walk probabilities. In this section, we first discuss our approach to node selection, and then present a formal definition of subgraphs.

4.3.1 Node Selection

In the proposed approach, a node is selected if (1) it may be an answer node and (2) the node's ranking as an answer node cannot be determined by using estimated values. If θ_i is the k-th highest lower similarity bound among all nodes at the i-th iteration, then the set L_i of answer-likely nodes at the i-th iteration is defined as follows:

Definition 3 (Answer-likely nodes): The set L_i of answer-likely nodes whose precise similarity scores may be higher than θ_i at the i-th iteration is defined as follows:

$$L_i = \{u \in V : \overline{s}_i[u] \geq \theta_i\} \tag{19}$$

This definition indicates that a node is an answer-likely node if its upper similarity bound is no lower than θ_i. This is because, if the upper bound of a node is lower than θ_i, then its exact similarity score cannot be θ_i or greater, so it cannot be an answer node. As shown below, the set of answer-likely nodes has the property of becoming smaller with each iteration. The set D_i of nodes whose ranking is determined to be among the top-k nodes at the i-th iteration is defined as follows:

Definition 4 (Ranking determined nodes): If $|L_i| = k$ (i.e., the number of answer-likely nodes is k), the set D_i of ranking-determined nodes whose exact ranks as answer nodes have been fixed by the i-th iteration is defined as follows:

$$D_i = \{u \in L_i : \underline{s}_i[u] > \overline{s}_i[v] \text{ or } \overline{s}_i[u] < \underline{s}_i[v], u \neq v, \forall v \in L_i\} \tag{20}$$

If $|L_i| \neq k$, then D_i is defined as follows:

$$D_i = \emptyset \tag{21}$$

Definition 4 defines node u as a rank-determined node if (1) the number of answer 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 exists a node whose lower or upper similarity bound lies between $\underline{s}_i[u]$ and $\overline{s}_i[u]$, then the rank of node u cannot be determined. For example, if we have $k = 4$ and $L_i = \{u_1, u_2, u_3, u_4\}$ where $0.8 \leq s_i[u_1] \leq 0.9$, $0.6 \leq s_i[u_2] \leq 0.7$, $0.3 \leq s_i[u_3] \leq 0.5$, and $0.2 \leq s_i[u_4] \leq 0.4$, we have $D_i = \{u_1, u_2\}$. This is because node u_1 and u_2 must be the top and second similar nodes from their estimated bounds, respectively. In Definition 4, if $|L_i| = k$, it requires $O(k \log k)$ time to compute the rank-determined nodes D_i because D_i can be obtained by sorting the nodes of L_i. Also, if $|L_i| \neq k$, then there is no need to compute D_i from Eq. (21). From the definitions of node sets L_i and D_i, the set S_i of selected nodes is defined as follows:

Definition 5 (Selected nodes): The set S_i of selected nodes whose upper and lower bounds are computed in the i-th iteration is defined as follows:

$$S_i = \begin{cases} V & (i = 0) \\ L_{i-1} \backslash D_{i-1} & (i \neq 0) \end{cases} \tag{22}$$

In this equation, $L_{i-1} \backslash D_{i-1}$ is calculated as $\{u \in L_{i-1} : u \notin D_{i-1}\}$ (i.e., the set obtained by subtracting D_{i-1} from L_{i-1}).

From Eq. (5), the proposed method first calculates the estimated scores for all nodes, but does not update these scores if (1) the node is not an answer-likely node, or (2) its ranking has not been determined. In other words, we only update the estimated value of a node if it is an answer-likely node whose ranking has not been determined by prior estimations.

To show the property of the selected nodes, we introduce several lemmas on node sets L_i and D_i. First, when A is a set of answer nodes, the property of the answer-likely nodes L_i is as follows:

Lemma 5 (Monotonic decrease in L_i): During iterative computations, the answer-likely nodes L_i decrease monotonically. That is, $L_i \subseteq L_{i-1}$ holds.

Proof If θ_i is the k-th highest lower bound at the i-th iteration, then we have $\theta_i \geq \theta_{i-1}$ due to the monotonic increasing property of lower bound estimations as described in

Lemma 3. Therefore, (1) if node u is included in L_i, the node must be included in set 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 may be included in set L_{i-1} since $\theta_i > \overline{s}_{i-1}[u] \geq \overline{s}_i[u] \geq \theta_{i-1}$ may hold. □

Lemma 6 (Convergence of L_i): After convergence, the set of answer-likely nodes is equal to the set of answer nodes, i.e., $L_\infty = A$.

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

$$L_\infty = \{u \in V : \overline{s}_\infty[u] \geq \theta_\infty\} = \{u \in V : s[u] \geq \theta\} = A \tag{23}$$

□

From Lemma 3, all the nodes are needed to compute L_i, but with repeated iterations, the set L_i can be computed with increasing efficiency. If $i \neq 0$ and $|L_{i-1}| > k$, then set L_i can be computed as follows:

$$L_i = \{u \in L_{i-1} : \overline{s}_i[u] \geq \theta_i\} \tag{24}$$

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

$$L_i = L_{i-1} \tag{25}$$

This is because $L_i \subseteq L_{i-1}$ and set L_i converges to set A according to Lemmas 5 and 6.

From Lemmas 5 and 6, ranking determined nodes exhibit the following property:

Lemma 7 (Monotonic increase in D_i): The set D_i of ranking-determined nodes has the property of increasing monotonically; i.e., the relation $D_i \supseteq D_{i-1}$ holds at the i-th iteration.

Proof We will first prove the case for $|L_{i-1}| \neq k$. From Definition 4, in this case $D_{i-1} = \emptyset \subseteq D_i$. Next we will prove the case for $|L_{i-1}| = k$. In this case, $L_i = L_{i-1} = A$ holds due to Lemmas 5 and 6. If node u is included in set D_{i-1}, this node must be included in set D_i. This is because (1) the estimated values have the property of increasing accuracy according to Lemma 3, and (2) $L_i = L_{i-1}$. If node u is not included in set D_{i-1}, it may be included in set D_i due to the property of increasing accuracy of the estimated values. □

Lemma 8 (Convergence of D_i): After convergence, we have $D_\infty = A$; i.e., the set of rank-determined nodes becomes equal to the set of answer nodes.

Proof After convergence, according to Lemma 4, we have $\underline{s}_\infty[u] = \overline{s}_\infty[u] = s[u]$, and according to Lemma 6, we have $L_\infty = A$. Therefore, from Definition 4:

$$\begin{aligned} D_\infty &= \{u \in L_\infty : \underline{s}_\infty[u] > \overline{s}_\infty[v] \text{ or } \overline{s}_\infty[u] < \underline{s}_\infty[v], u \neq v, \forall v \in L_\infty\} \\ &= \{u \in A : s[u] > s[v] \text{ or } s[u] < s[v], u \neq v, \forall v \in A\} = A \end{aligned} \tag{26}$$

and so $D_\infty = A$ holds after convergence. □

Lemmas 5 and 7 show that the sets L_i and D_i have monotonically decreasing and increasing properties, respectively. Also from Lemmas 6 and 8 sets L_i and D_i are equal to the set A of answer nodes after convergence. Therefore, the following lemma holds for the selected nodes:

Lemma 9 (Selected nodes): The set of selected nodes is monotonically decreasing and becomes an empty set after convergence; i.e., $S_i \subseteq S_{i-1}$ and $S_\infty = \emptyset$.

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

Since, as described in Definition 5, the set of selected nodes is initialized to the set of all nodes in the graph, the set of selected nodes has the property that $V = S_0 \supseteq S_1 \supseteq \ldots \supseteq S_i$. Moreover, the property of $S_\infty = \emptyset$ implies that our approach terminates in a finite number of iterations since this indicates that there are no nodes whose estimated values have to be updated.

4.3.2 Subgraph Construction

In this section, we discuss our method for constructing subgraphs in order to efficiently update the estimated values at high speed. A naive approach is to update the upper/lower estimated values for the whole graph at each iteration. However, this approach is 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 what we mean by subgraph, and then discuss its properties. We also describe how to incrementally update subgraphs at each iteration.

As shown in Definitions 1 and 2, the upper and lower estimated values are computed by using random walk probabilities obtained by Eq. (2). Therefore, the subgraphs are constructed to facilitate the fast computation of random walk probabilities. To construct the subgraphs, we use the set, H_i, of nodes whose number of hops from the query nodes is no greater than i (i.e., H_i does not contain any nodes more than i hops from a query node). We construct subgraph G_i at the i-th iteration based on set H_i as follows:

Definition 6 (Subgraph): If $G_i = \{V_i, E_i\}$ is the subgraph of graph G at the i-th iteration, then V_i and E_i are defined as follows:

$$V_i = H_i \cap R_i \tag{27}$$

$$E_i = \{(u, v) \in E : u \in V_i, v \in V_i\} \tag{28}$$

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

This definition indicates that (1) if a node in the original graph G is within i hops of a query node and the query node is reachable from this node then the subgraph includes this node, and (2) if two nodes are joined by an edge in a partial graph and these nodes are both in the partial graph, then this edge is a subgraph.

This definition also shows that as the number of query nodes decreases, the size of the subgraphs becomes smaller and searches can be performed at higher speed. This is because if there are fewer query nodes, the set of nodes in the subgraph given by Eq. (27) becomes smaller. The relationship between the number of query nodes and the search time is shown in Sect. 5.

We introduce the following lemma to illustrate the properties of subgraphs.

Lemma 10 (Upper/lower bounds of subgraphs): The upper/lower estimated values of selected nodes at the i-th iteration can be computed from V_i and E_i.

Proof As described above, $p_i[u]$ is the probability that a random walk of length i starting from a query node will end up at node u. So if a node is situated more than i hops away from a query node, its random walk probability at the i-th iteration must be 0. As a result, we can obtain the upper/lower estimated values from node set H_i and the set of edges that are incident to H_i. From Eqs. (2), (3) and (11), it is clear that, if the set of nodes S_i is unreachable from a node, then the node's random walk probability will not affect the estimated upper or lower bounds of any node in S_i. This indicates that it is possible to compute the upper and lower bounds of nodes included in set S_i from node set R_i and the set of edges between nodes in R_i. As a result, we need only node set $V_i = H_i \cap R_i$ and edge set $E_i = \{(u, v) \in E : u \in V_i, v \in V_i\}$ to obtain the upper/lower estimated values of the selected nodes in the i-th iteration. □

This proof implies that we can use subgraphs to compute random walk probabilities and estimate upper/lower bounds for selected nodes. Subgraphs can be used to efficiently compute upper and lower bounds as follows:

Definition 7 (Computing probabilities by subgraphs): If $C_i[u]$ is the set of nodes that are incident to node u in the subgraph G_i, then we can compute the random walk probability $p_i[u]$ of node u at the i-th iteration as follows:

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

The above equation can be obtained by replacing $C[u]$ with $C_i[u]$ in Eq. (2). To find the top-k nodes at high speed, we use the above formula to calculate the probabilities of random walks from the subgraph. We use Definitions 2 and 1 to obtain the upper and lower bounds of nodes respectively. The naive approach processes the whole graph to calculate these bounds, but by using subgraphs we are able to compute these estimated values more efficiently.

However, it can be computationally expensive to construct these subgraphs if the approach of Definition 6 is applied directly, because all the nodes in the graph are used, and this can make it impossible to construct subgraphs efficiently. We therefore introduce an incremental approach to obtaining subgraphs that utilizes a set of nodes, h_i, and a set of edges, e_i, in iterative computations. Here, h_i is defined as the set of nodes that are i hops away from a query node. Therefore, $h_0 = H_0 = Q$, and $H_i = \bigcup_{j=0}^{i} h_j = H_{i-1} + h_i$. Also, e_i is the set of edges that link node sets h_i and H_{i-1}. That is, since $H_i = H_{i-1} + h_i$, we have $e_i = \{(u, v) \in E : u \in h_i, v \in H_{i-1}$ or $u \in H_{i-1}, v \in h_i$ or $u \in h_i, v \in h_i\}$. Note that h_i and e_i can be obtained from a single breadth-first search rooted on the query nodes; whereby h_i and e_i can be extracted with low computation cost in time $O(N + M)$. For sets h_i and e_i, we have the following lemma:

Lemma 11 (Subgraph inclusion): If the sets of nodes and edges are denoted by $V'_i = V_{i-1} + h_i$ and $E'_i = E_{i-1} + e_i$ respectively, then if $i \neq 0$, we have $V_i \subseteq V'_i$ and $E_i \subseteq E'_i$.

Proof If $i \neq 0$, since $H_i = H_{i-1} + h_i$ and $R_i \subseteq R_{i-1}$, we have the following equation from Eq. (27):

$$V_i = (H_{i-1} + h_i) \cap R_i \subseteq (H_{i-1} + h_i) \cap R_{i-1} \tag{30}$$

Therefore, we have

$$V_i \subseteq H_{i-1} \cap R_{i-1} + h_i \cap R_{i-1} \subseteq V_{i-1} + h_i = V'_i \tag{31}$$

If $i \neq 0$, then from Eq. (28), $V_i \subseteq V_{i-1} + h_i$, and so

$$E_i \subseteq \{(u, v) \in E : u \in (V_{i-1} + h_i), v \in (V_{i-1} + h_i)\} \tag{32}$$

Therefore, since $V_{i-1} = H_{i-1} \cap R_{i-1} \subseteq H_{i-1}$:

$$\begin{aligned} E_i \subseteq & \{(u, v) \in E : u \in V_{i-1}, v \in V_{i-1}\} + \\ & \{(u, v) \in E : u \in H_{i-1}, v \in h_i \\ & \text{or } u \in h_i, v \in H_{i-1} \text{ or } u \in h_i, v \in h_i\} \end{aligned} \tag{33}$$

As a result, $E_i \subseteq E_{i-1} + e_i = E'_i$ holds. □

Based on Lemma 11, we can incrementally construct subgraphs while iterating. For graph G'_i given by $G'_i = \{V'_i, E'_i\}$, we have $V'_i = V_{i-1} + h_i$ and $E'_i = E_{i-1} + e_i$, whereby graph G'_i can be constructed incrementally. That is, G'_i can be obtained by adding to graph G_{i-1} all the nodes that are a hop distance of i from the query nodes, and their corresponding edges. Since (1) $G_i \subseteq G'_i$ holds according to the above lemma, and (2) sets h_i and e_i include nodes and edges that are not included in paths to the set S_i of selected nodes, we can compute subgraph G_i by performing a breadth-first search to determine the paths to S_i in graph G'_i.

Algorithm 1 shows our incremental approach for subgraph construction. If $i = 0$, the algorithm initializes the node and edge sets to $V_0 = Q$ and $E_0 = \{(u, v) \in E :$

Algorithm 1 Subgraph construction

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

Algorithm 2 Castanet

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

$u \in Q, v \in Q\}$, respectively (lines 2 – 3). This is because $V_0 = H_0 \cap R_0 = Q \cap V = Q$ from Eq. (27). Otherwise, it computes graph G'_i from the graph of the previous iteration, G_{i-1} (lines 5 – 7). V_i and E_i are computed from G'_i by performing a breadth-first search (lines 8 – 9). As shown in Algorithm 1, we can incrementally compute subgraphs without using the whole graph.

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). For each node included in the subgraph, it computes the node's random walk probability (lines 9 – 11), since these random walk probabilities are needed to estimate the upper and lower bounds (Lemma 10). It estimates the bounds of the selected nodes (lines 12 – 15). If $i = 0$, it computes L_i for the answer-likely nodes according to Definition 3 (lines 16 – 17). Otherwise, it incrementally updates the set L_i from L_{i-1} (lines 18 – 24). From Eq. (4), if $|L_i| \neq k$ then $D_i = \emptyset$, so D_i is only calculated when $|L_i| = k$ (lines 25 – 26). It then updates the selected nodes (line 30). If the set of selected nodes is empty, then the iterations are terminated (line 31). The node ranking is obtained by sorting the nodes of set D_i according to their upper or lower estimated bounds (line 32). Finally, D_i is returned as the set of answer nodes (lines 33 – 34).

In practice, if several nodes have same edge weights from adjacent nodes, the nodes can have the same similarity for the given query. As a result, the number of answer nodes could be more than k. In that case, upper and lower bounds also have the same scores for such the nodes from Definition 1 and 2. Therefore, if several nodes have the same k-th highest lower bound, our approach checks whether the nodes have the same edge weights from adjacent nodes. If so, we determine D_i although D_i has more than k nodes to rank them; otherwise, we compute L_i by iteratively updating upper and lower bounds.

As shown in Algorithm 2, our search method does not need any precomputation. In other words, it can perform ad-hoc searches. Also, our algorithm does not need any user-defined internal parameters. Therefore, it offers the user a simple means of performing PPR searches.

4.5 Theoretical Analysis

This section presents a theoretical analysis of the search results and the computation cost of our algorithm. First, we will present a theoretical discussion on the search results.

Theorem 1 (Search accuracy): Our proposed method computes the top-k nodes with exact ranking.

Proof Algorithm 2 performs computations until the set of selected nodes converges to an empty set, whereupon it returns D_i as the set of answer nodes. If $S_{i+1} = \emptyset$, then $L_i = D_i$ from Definition 5. Since $L_i \supseteq A$ and $D_i \subseteq A$ from Lemmas 5, 6, 7 and 8, we have $L_i = D_i = A$ if $L_i = D_i$ holds. Therefore we have $D_i = A$ if $S_{i+1} = \emptyset$ holds. Since $D_i = A$, the ranking of all nodes in set D_i is fixed after achieving convergence. Therefore, we can compute the top-k nodes with exact node ranking. □

Next, 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 the proposed method, respectively. Also, let n and m be the average numbers of nodes and edges in the subgraphs, respectively. Note that the original method requires $O(N + M)T$ time.

Theorem 2 (Computation cost): The proposed method requires, on average, $O((l + n + m + k \log k)t + N + M)$ time to find the 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 are computed by breadth-first search in $O((n+m)t)$ time in the subgraph at each iteration, and (2) h_i and e_i are obtained, on average, in $O(N + M)$ time. It takes, on average, $O((n+m)t)$ and $O(l \cdot t)$ time to compute the random walk probabilities and estimate the upper/lower bounds from the subgraphs at each iteration, respectively. We compute the sets of answer-likely nodes L_i in $O(l \cdot t)$, and we obtain rank-determined nodes D_i in $O(k \log k \cdot t)$ time. This is because, in each iteration, we can compute rank-determined nodes in $O(k \log k)$ time by exploiting quicksort. After reaching convergence, the answer nodes are sorted to obtain the node ranking, which takes $O(k \log k)$ time. As a result, our approach requires, on average, $O((l+n+m+k \log k)t+N+M)$ time. □

5. Experimental Evaluation

We perform an experimental evaluation of the proposed Castanet algorithm below. In this experiment, we compare our approach with three other methods — the Monte Carlo-based approach proposed by Avrachenkov et al. [12], the matrix-based approach proposed by Fujiwara et al. [7], and the original iterative approach [3]. The results obtained with these four methods are labelled "Castanet", "Monte", "Matrix" and "Original", respectively.

We used the following data sets in the experiment:

- Notredame[†]: A set of Web data taken from the University of Notre Dame. In this graph, nodes represent Web pages and edges represent hyperlinks between them. There are 325,729 nodes and 1,497,135 edges.
- CNR[††]: CNR is a public research organization in Italy. This data set is a Web graph of the CNR domain, with 325,557 nodes and 3,216,152 edges.
- Email[†††]: A set of email data from a European research institute. In this graph, each node corresponds to an email address, and an edge between nodes corresponds to one or more email messages sent between the corresponding email addresses. This graph has 265,214 nodes and 420,045 edges.
- Social[††††]: This graph is obtained from Slashdot.org. The nodes correspond to Slashdot users, and the edges represent interactions among these users. This graph has 82,144 nodes and 549,202 edges.

In the experiments, three query nodes were chosen at random. The experiments were performed on an Intel Xeon server with a 3.33 GHz CPU.

5.1 Search Time

We evaluated the search time of each approach. Figure 1 shows the results. In this figure, the results of the proposed method 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 number of answer nodes had no effect on the search times of the existing methods. This is because (1) similarity scores are computed for all nodes in the Monte Carlo-based method and the original iterative method, and (2) the time required for matrix precomputation dominates the search time of the matrix-based method. Table 2 details the parameters of the proposed method and the original iterative approaches for $c = 0.5$ and $k = 10$. Table 3 shows a breakdown of the search time of each approach for the Notredame data set where $c = 0.5$ and $k = 10$.

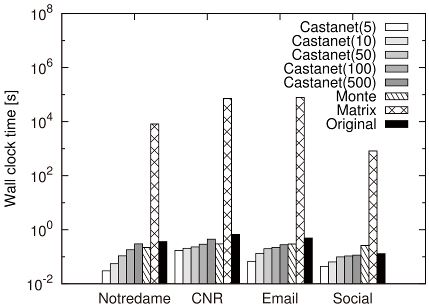

Fig. 1 Search times of each method.

Table 2 Parameter scores.

Parameter	Data set			
	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 times.

Method	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

As shown in Fig. 1, our approach achieves a search time reduction of up to 90% compared with the original iterative methods if $k = 5$. This figure also shows that the search time of the proposed method decreases as the number of answer nodes (k) is reduced. This is because a smaller value of k leads to smaller node sets H_i and R_i in Definition 6. If $k = 500$, then the Monte-Carlo method is sometimes faster than the proposed method, but this value of k is unlikely to be used in practical applications [4], [13]. Also, as shown in Table 3, the actual search time of methods based on matrix decomposition may be faster than the proposed method, but if the precomputation time is factored in, then the proposed method is faster overall.

5.2 Search Accuracy

One characteristic of the proposed method is that it achieves exact node ranking while the Monte Carlo-based approach does not. Since the search accuracy of the Monte Carlo-based approach depends on the number of random walks, we varied the number of random walks in this experiment and compared the performance of the Monte Carlo method with the proposed method. Figures 2 and 3 show the search time and accuracy of each approach for the Notredame data set, respectively, with $c = 0.5$ and $k = 10$. In Fig. 3, we used average precision as the metric of search accuracy [14]. The

[†] http://vlado.fmf.uni-lj.si/pub/networks/data/ND/NDnets.htm

[††] http://law.di.unimi.it/webdata/cnr-2000/

[†††] http://snap.stanford.edu/data/email-EuAll.html

[††††] http://snap.stanford.edu/data/soc-sign-Slashdot090221.html

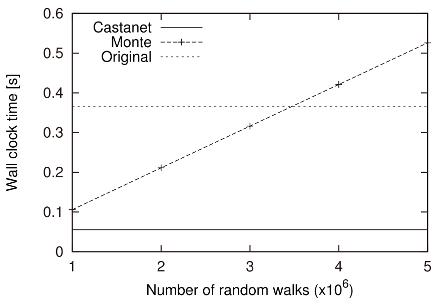

Fig. 2 Search time vs. number of random walks.

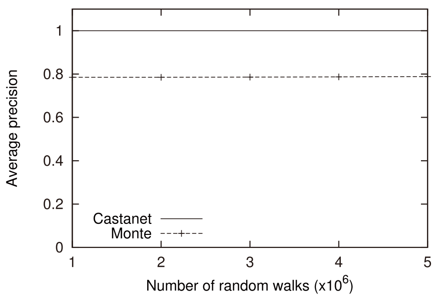

Fig. 3 Search accuracy vs. number of random walks.

average precision has a value ranging from 0 to 1, where results that are the same as the original search results are deemed to have an average precision of 1.

As shown in Fig. 2, the Monte Carlo-based method requires more computation time as the number of random walks increases. If the number of random walks is not properly set, the Monte Carlo-based approach can require more computation time than the original iterative approach. As Fig. 3 shows, the proposed method has an average precision of 1 because it is guaranteed to provide accurate rankings, but the Monte Carlo method has lower search precision. This figure also shows that the accuracy of the Monte Carlo-based method does not improve as the number of random walks increases. This is because the accuracy of the Monte Carlo method had already converged under these experimental conditions. From Fig. 2 and Fig. 3, it can be seen that our approach is superior to the Monte Carlo-based approach in both speed and accuracy.

5.3 Evaluation with Multiple Scaling Parameters

As discussed in Sect. 1, it is essential to be able to handle ad hoc scaling parameters in real applications. Moreover, as discussed in Sect. 4.4, the proposed method requires no pre-computation, so it is able to handle ad hoc scaling parameters. To evaluate this feature of the proposed method, we evaluated its performance with multiple scaling parameter settings. The results are shown in Fig. 4. In this experiment, we set $k = 10$ and used the Notredame data set. As demonstrated by the above experiment, the matrix decomposition method has no scaling parameters that can be applied quickly in an ad hoc fashion, so we only compared our method with the Monte Carlo method.

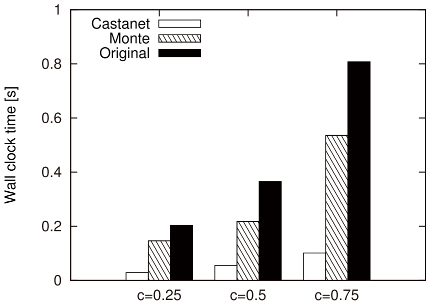

Fig. 4 Search time vs. scaling parameters.

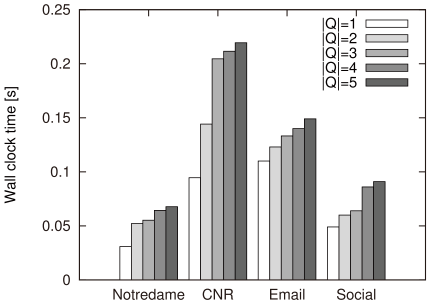

Fig. 5 Search time vs. number of query nodes.

As shown in Fig. 4, decreasing the score of the scaling parameter results in greater search speed. This is because in our method the upper bound scores are obtained by adding $c^{i+1} W_{max}[u]\ p_i[R_i]$ to the lower bound scores as shown in Eq. (11), so if c is small then the upper bound is expected to have small score. Also, random walk length in the Monte Carlo-based method increases stochastically with the value of c. Therefore, the search times increase in the Monte Carlo-based method as c becomes larger.

5.4 Evaluating Different Numbers of Query Nodes

As discussed in Sect. 4.3.2, the proposed method calculates subgraphs by using set H_i of nodes for which the number of hops from the query nodes is no more than i, so faster searches can be performed by reducing the number of nodes that have to be considered. Here, we studied how the search time varies with the number of query nodes. The results are shown in Fig. 5, where $|Q|$ represents the number of query nodes.

As this figure shows, the proposed method can process queries faster as the number of query nodes decreases. This is because, as can be seen from Definition 6, the number of query nodes has a large effect on the size of node set H_i, and node set H_i has a large effect on the size of the subgraph. We confirmed that the subgraphs become smaller as the number of query nodes decreases, resulting in shorted search times in the proposed method.

6. Conclusion

In this article, we have addressed the problem of efficiently finding the top-k nodes for PPR. We proposed an algorithm called Castanet that can perform high-speed searches by using upper and lower bounds to prune unnecessary nodes. Our experiments confirmed that this approach is more effective than conventional methods in dealing with ad hoc changes to graphs and scaling parameters.

References

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," Technical Report 1999-66, Stanford InfoLab, Nov. 1999.

[2] A. Balmin, V. Hristidis, and Y. Papakonstantinou, "Objectrank: Authority-based keyword search in databases," VLDB, pp.564–575, 2004.

[3] G. Jeh and J. Widom, "Scaling personalized web search," WWW, pp.271–279, 2003.

[4] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu, "Automatic multimedia cross-modal correlation discovery," KDD, pp.653–658, 2004.

[5] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," ICDM, pp.613–622, 2006.

[6] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa, "Fast and exact top-k search for random walk with restart," PVLDB, vol.5, no.5, pp.442–453, 2012.

[7] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka, "Efficient personalized pagerank with accuracy assurance," KDD, 2012.

[8] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós, "Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments," Internet Mathematics, vol.2, no.3, pp.333–358, 2005.

[9] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized pagerank," PVLDB, vol.4, no.3, pp.173–184, 2010.

[10] H. Shiokawa, Y. Fujiwara, and M. Onizuka, "Fast algorithm for modularity-based graph clustering," AAAI, pp.1170–1176, 2013.

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

[12] K. Avrachenkov, N. Litvak, D. Nemirovsky, E. Smirnova, and M. Sokol, "Quick detection of top-k personalized pagerank lists," WAW, vol.6732, pp.50–61, 2011.

[13] Y.Z. Guo, K. Ramamohanarao, and L.A.F. Park, "Personalized pagerank for web page prediction based on access time-length and frequency," Web Intelligence, pp.687–690, 2007.

[14] B. Carterette, J. Allan, and R.K. Sitaraman, "Minimal test collections for retrieval evaluation," SIGIR, pp.268–275, 2006.

Yasuhiro Fujiwara is a distinguished technical member of NTT Software Innovation Center. He received the B.E. and M.E. degrees From Waseda University in 2001 and 2003, respectively, and he received the Ph.D. degree from The University of Tokyo in 2012. He joined NTT Cyber Solutions Laboratories in 2003. He currently a research engineer in NTT Software Innovation Center. His research interest include database, data mining, artificial intelligence, and machine learning.

Makoto Nakatsuji is a Manager in NTT Resonant Inc. He completed his Ph.D. in Social Informatics at Kyoto University Graduate School of Informatics in 2010. He was a visiting scholar in The Tetherless World Constellation at Rensselaer Polytechnic Institute in 2013. His research interests include semantic data mining, recommendation, deep learning, Question Answering systems, and graph analysis.

Hiroaki Shiokawa is an Assistant Professor at University of Tsukuba. He received B.S. in information science, M.E. and Ph.D. in engineering from University of Tsukuba in 2009, 2011 and 2015 respectively. From 2011 to 2015, he was a researcher at NTT labs, and he joined Center for Computational Sciences at University of Tsukuba in Nov. 2015. His current main research interests include database systems, data engineering, data mining, and graph data management.

Takeshi Mishima received the B.E. and M.E. degrees in information science from the University of Tsukuba in 1994 and 1996, respectively. He joined NTT in 1996 and has been engaged in the research and development of control system architecture for switching system. He is currently working for a database replication middleware using Off-The-Shelf resources at NTT Software Innovation Center. He is a member of the Information Processing Society of Japan (IPSJ).

Makoto Onizuka is a professor in the Graduate School of Information Science and Technology at the Osaka University. He received his Ph.D. in the Graduate School of Information Science and Technology at the Tokyo Institute of Technology. His main research is in the area of database systems, with special interest in graph mining algorithms, distributed query optimization, and exploratory search. He is a recipient of the IPSJ Yamashita SIG Research Award (2004) and the DBSJ Kambayashi Award for Young Researcher (2008). He serves is a vice-chair of Technical Committee on Data Engineering, and he is a guest editor-in-chief of the special section on Data Engineering and Information Management (2016).