Efficient and Exact Local Search for Random Walk Based Top-K Proximity Query in Large Graphs

Yubao Wu, Ruoming Jin, and Xiang Zhang

Abstract—Top-k proximity query in large graphs is a fundamental problem with a wide range of applications. Various random walk based measures have been proposed to measure the proximity between different nodes. Although these measures are effective, efficiently computing them on large graphs is a challenging task. In this paper, we develop an efficient and exact local search method, FLoS (Fast Local Search), for top-k proximity query in large graphs. FLoS guarantees the exactness of the solution. Moreover, it can be applied to a variety of commonly used proximity measures. FLoS is based on the *no local optimum* property of proximity measures. We show that many measures have no local optimum. Utilizing this property, we introduce several operations to manipulate transition probabilities and develop tight lower and upper bounds on the proximity values. The lower and upper bounds monotonically converge to the exact proximity value when more nodes are visited. We further extend FLoS to measures having local optimum by utilizing relationship among different measures. We perform comprehensive experiments on real and synthetic large graphs to evaluate the efficiency and effectiveness of the proposed method.

Index Terms—Local search, proximity search, random walk, top-k search, nearest neighbors

1 Introduction

Given a large graph and a query node, finding its k-nearest-neighbor (kNN) is a primitive operation that has recently attracted intensive research interests [1], [2], [3], [4]. In general, there are two challenges in top-k proximity query. One is to design proximity measures that can effectively capture the similarity between nodes. Another challenge is to develop efficient algorithms to compute the top-k nodes for a given measure.

Designing effective proximity (similarity) measures is a nontrivial task. Random walk based measures have been shown to be effective in many applications. Some examples include discounted/truncated hitting time [1], [5], penalized hitting probability (PHP) [6], [7], random walk with restart [8], [9], RoundTripRank [10], and absorption probability [11].

Although various proximity measures have been developed, how to efficiently compute them remains a challenging problem. For most random walk based measures, a naive method requires matrix inversion, which is prohibitive for large graphs. Two *global* approaches have been developed. One applies the power iteration method over the entire graph [4], [12], [13]. Another approach precomputes and stores the inversion of a matrix [8], [14], [15]. The precomputing step is usually expensive and needs to be repeated whenever the graph changes.

To improve the efficiency, local methods have been developed [1], [5], [7], [9]. The idea is to visit the nodes near the query node and dynamically expand the search range. Node proximities are estimated based on local information only. Without using the global information, however, most existing local search methods cannot guarantee to find the exact solution. Moreover, they are usually designed for specific measures and cannot be generalized to other measures.

In this paper, we propose Fast Local Search (FLoS), a simple and unified local search method for efficient and exact top-k proximity query in large graphs. FLoS has the following properties.

- *Exact*: It guarantees to find the exact top-k nodes.
- *Unified*: It is a general method that can be applied to a variety of random walk based proximity measures. Most existing methods are designed for specific measures.
- *Efficient*: It uses a simple local search strategy that needs neither preprocessing nor iterating over the entire graph. Experimental results show that it is orders of magnitude faster than alternatives.

The key idea behind FLoS is that we can develop upper and lower bounds on the proximity of the nodes near the query node. These bounds can be dynamically updated when a larger portion of the graph is explored and will finally converge to the exact proximity value. The top-k nodes can be identified once the differences between their upper and lower bounds are small enough to distinguish them from the remaining nodes.

The theoretical basis of FLoS relies on the no local optimum property of proximity measures. That is, given

- *Y. Wu and X. Zhang are with the Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106. E-mail: {yubao.wu, xiang.zhang}@case.edu.*
- *R. Jin is with the Department of Computer Science, Kent State University, Kent, OH 44240. E-mail: jin@cs.kent.edu.*

Manuscript received 25 Sept. 2015; revised 18 Dec. 2015; accepted 2 Jan. 2016. Date of publication 7 Jan. 2016; date of current version 30 Mar. 2016. Recommended for acceptance by L. B. Holder.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2016.2515579

a query node q, for any node i ($i \neq q$) in the graph, i always has a neighbor that is closer to q than i is. We show that many measures have no local optimum. This property ensures that the proximity of unvisited nodes is bounded by the maximum proximity (or minimum proximity for some measures) in the boundary of the visited nodes. It can be utilized to find the top-k nodes without exploring the entire graph under the assumption that the exact proximity can be computed based on local information. However, for most measures, the exact proximity cannot be computed without searching the entire graph. To tackle this challenge, we introduce several simple operations to modify transition probabilities, which enable developing upper and lower bounds on the proximity of visited nodes. The developed upper (lower) bounds monotonically decrease (increase) when more nodes are visited. We further study the relationship among different measures and show that FLoS can also be applied to measures having local optimum. Extensive experimental results show that, for a variety of measures, FLoS can dramatically improve the efficiency compared to the state-of-the-art methods.

2 Related Work

Various random walk based proximity measures have been proposed recently [16]. Examples include truncated or discounted hitting time [1], [5], [17], penalized hitting probability [6], [7], random walk with restart [1], [8], effective importance (degree normalized random walk with restart) [9], RoundTripRank [10], and absorption probability [11]. The Katz score [18] captures multiple paths between two nodes and is closely related to the random walk based proximity measures [13].

The basic approach for proximity query is to use the power iteration method [12]. An improved iteration method designed for random walk with restart decomposes the proximity into random walk probabilities of different length [4]. It tries to reduce the number of iterations by estimating the proximity based on the information collected so far. The iteration method can also be improved by the prioritized execution of the iterative computation, where the node with the largest residual proximity value is updated first [13]. Another approach precomputes the information needed for proximity estimation during the query process [8], [14], [15]. However, this step is time consuming and becomes infeasible when the graph is large or constantly changing. Graph embedding (GE) method embeds nodes into geometric space so that node proximities can be preserved as much as possible [19]. The embedding step is also time-consuming. Moreover, the proximities in the new space are not exactly the same as the ones in the original graph.

Based on the intuition that nodes near the query node tend to have high proximity, local search methods try to visit a small number of nodes to approximate the proximities. Best-first [7] and depth-first [20] search strategies simply extract a fixed number of nodes near the query node. An approximate local search algorithm is proposed for truncated hitting time [5]. The key idea is to develop upper and lower bounds that can be used to approximate the proximities of local nodes. A similar local search algorithm is developed for personalized PageRank and degree normalized personalized PageRank [1]. The push style method is first developed in [21] for random walk with restart, and later improved by [2], [22] for the top-k query problem. Starting from the query node, the push style method propagates the proximity value to the nodes in the neighborhood of the query node, and obtains approximate proximity values for them. This basic idea has been adapted to compute the top-k nodes for effective importance [9], RoundTripRank [10], and Katz score [23]. Most of the existing local search methods cannot guarantee the exactness. Moreover, all of them are designed for specific proximity measures. It is unclear whether the local search methods can be generalized to other measures.

TABLE 1
Main Symbols

Symbols	Definitions
$G(V, E)$	undirected graph G with node set V and edge set E
N_i	neighbors of node i
$w_{i,j}$	weight of edge (i, j)
w_i	degree of node $i, w_i = \sum_{j \in N_i} w_{i,j}$
q	query node
$\mathbf{e}$	$n \times 1$ vector with $\mathbf{e}_q = 1$ and $\mathbf{e}_i = 0$ if $i \neq q$, where $n = \lvert V \rvert$
k	number of returned nodes
S	a set of nodes
$\bar{S}$	complement of $S : \bar{S} = V \setminus S$
δS	boundary of $S : \{i \in S \mid \exists j \in N_i \cap \bar{S}\}$
$\delta \bar{S}$	boundary of $\bar{S} : \{i \in \bar{S} \mid \exists j \in N_i \cap S\}$
$\mathbf{r}$	$n \times 1$ vector, $\mathbf{r}_i$: proximity of node i w.r.t. the query node q
$\bar{\mathbf{r}}$	upper bound of $\mathbf{r} : \bar{\mathbf{r}}_i \geq \mathbf{r}_i, \forall i \in S$
$\underline{\mathbf{r}}$	lower bound of $\mathbf{r} : \underline{\mathbf{r}}_i \leq \mathbf{r}_i, \forall i \in S$
$p_{i,j}$	transition probability from node i to j
$\mathbf{P}$	transition probability matrix : $\mathbf{P}_{q,j} = 0; \mathbf{P}_{i,j} = p_{i,j}$ if $i \neq q$
d, $\mathbf{r}_\mathrm{d}$	a dummy node d with constant proximity value $\mathbf{r}_\mathrm{d}$
c	decay factor in PHP, DHT, RWR, EI, or RT
$u \rightsquigarrow v$	node u can reach node v in the transition graph
$u \not\rightsquigarrow v$	node u cannot reach node v in the transition graph
$u \rightsquigarrow S$	node u can reach at least one node in S
$u \not\rightsquigarrow S$	node u cannot reach any node in S

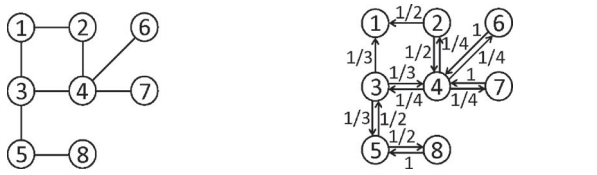
3 No Local Optimum Property

In this section, we first introduce the basic concept of no local optimum property of proximity measures and discuss how it can be used to bound the proximity of the unvisited nodes. Then we study whether commonly used measures have no local optimum and discuss the relationship between them. Table 1 lists the main symbols and their definitions. The top-k query problem is defined as

Definition 1 (Top-k Query Problem). *Suppose that we have an undirected graph $G(V, E)$, a query node q and a number k. Let $\mathbf{r}_i$ represent the proximity of node i with regard to the query node q. The top-k query problem aims at finding a node set $K \subseteq V \setminus \{q\}$ such that $\lvert K \rvert = k$ and $\mathbf{r}_i \geq \mathbf{r}_j$, for any node $i \in K$ and $j \in V \setminus \big(K \cup \{q\}\big)$.*

3.1 Theoretical Basis

Note that for some measures, such as penalized hitting probability, random walk with restart, effective importance, RoundTripRank, Katz score and absorption probability, the larger the proximity the closer the nodes. In this case, no local optimum means no local maximum. For other



(a) original graph (b) transition graph (PHP, $q=1$)

Fig. 1. An example graph and its transition graph.

measures, such as discounted or truncated hitting time, the smaller the proximity the closer the nodes. In this case, no local optimum means no local minimum.

Given an undirected and edge weighted graph $G = (V, E)$ and a query node $q \in V$, let $\mathbf{r}$ be the proximity vector with $\mathbf{r}_i$ representing the proximity of node $i \in V$ with respect to the query node q.

Definition 2 (No Local Maximum). *A proximity measure has no local maximum if for any node $i \neq q$, there exists a neighbor node j of i (i.e., $j \in N_i$), such that $\mathbf{r}_j > \mathbf{r}_i$.*

Definition 3 (No Local Minimum). *A proximity measure has no local minimum if for any node $i \neq q$, there exists a neighbor node j of i (i.e., $j \in N_i$), such that $\mathbf{r}_j < \mathbf{r}_i$.*

We say that a proximity measure has no local optimum if it has no local maximum or minimum. In Section 3.2, we will examine whether the commonly used proximity measures have no local optimum. Unless otherwise mentioned, in the next, we assume that the larger the proximity the closer the nodes, and focus on the no local maximum property. All conclusions can also be applied to the proximity measures with no local minimum.

Let S be a set of nodes, and $\bar{S} = V \setminus S$ be the remaining nodes. We use $\delta S = \{i \in S \mid \exists j \in N_i \cap \bar{S}\}$ to denote the *boundary* of S, and $\delta\bar{S} = \{i \in \bar{S} \mid \exists j \in N_i \cap S\}$ to denote the boundary of $\bar{S}$.

Fig. 1a shows an undirected graph with 8 nodes. Suppose that the node set $S = \{1, 2, 3, 4\}$, then we have $\bar{S} = \{5, 6, 7, 8\}$, $\delta S = \{3, 4\}$, and $\delta\bar{S} = \{5, 6, 7\}$.

Theorem 1. *Let S be a node set containing the query node, and u be the node with the largest proximity in δS. If a proximity has no local maximum, we have that $\mathbf{r}_u > \mathbf{r}_j$ ($\forall j \in \bar{S}$).*

Proof. Suppose otherwise. We have that $\exists j \in \bar{S}$, such that $\mathbf{r}_u \leq \mathbf{r}_j$. Now suppose that node v is the node with the largest proximity in $\bar{S}$. We have that $\forall i \in \bar{S} \cup \delta S$, $\mathbf{r}_v \geq \mathbf{r}_i$. The neighbors of node v must exist in $\bar{S} \cup \delta S$, i.e., $N_v \subseteq \bar{S} \cup \delta S$. Therefore, we have $\mathbf{r}_v \geq \mathbf{r}_i$ ($\forall i \in N_v$), which means node v is a local maximum. This contradicts the assumption. □

Based on Theorem 1, assuming that we already have the exact proximity vector $\mathbf{r}$, we can design a simple local search strategy as shown in Algorithm 1 to find the top-k nodes. It starts from the query node q and uses K and S to store the top-k nodes and visited nodes respectively. In each iteration, the algorithm finds the node u that has the largest proximity in $S \setminus K$ and expands S to the neighbors of node u. Since $\delta S \subseteq S \setminus K$, the maximum proximity value in $S \setminus K$ must be no less than the maximum proximity value in δS, and greater than the maximum proximity value in the unvisited nodes $\bar{S}$ based on Theorem 1. Thus, K contains the top-k nodes. The algorithm continues until $|K| = k + 1$.

Let h be the average number of neighbors of a node. In each iteration t, on average h nodes are added to S. This takes $O(h \log ht)$ time for a sorted list. The overall complexity of Algorithm 1 is thus $O(\sum_{t=1}^{k} h \log ht) = O(hk \log\ hk)$.

TABLE 2
No Local Optimum Property of Some Measures

Proximity measures	Abbr.	Ref.	Property
Penalized hitting probability	PHP	[6]	No local maximum
Effective importance	EI	[9]	No local maximum
Discounted hitting time	DHT	[1]	No local minimum
Truncated hitting time	THT	[5]	No local minimum (within L hops)
Random walk with restart	RWR	[8]	Local maximum
RoundTripRank	RT	[10]	Local maximum
Katz score	KZ	[18]	Local maximum
Absorption probability	AP	[11]	Local maximum

Algorithm 1. The Basic Top-k Local Search Algorithm

Input: $G(V, E)$, query node q, proximity vector $\mathbf{r}$, number k
Output: Top-k node set K
1: $S \leftarrow \{q\}$; $K \leftarrow \{\}$;
2: **while** $|K| < k + 1$ **do**
3: $u \leftarrow \text{argmax}_{i \in S \setminus K} \mathbf{r}_i$;
4: $K \leftarrow K \cup \{u\}$; $S \leftarrow S \cup N_u$;
5: **return** $K \setminus \{q\}$;

3.2 Measures with and without Local Optimum

Table 2 summarizes whether the commonly used proximity measures have no local optimum property. Next, we use penalized hitting probability [6], [7] as an example to illustrate that it has no local maximum. We use w_i to denote the degree of node i, and $w_{i,j}$ to denote the edge weight between i and j. The transition probability from i to j is thus $p_{i,j} = w_{i,j}/w_i$.

Suppose the undirected graph in Fig. 1a has unit edge weight. Node 3 has degree 3, thus its transition probability to node 4 is $p_{3,4} = 1/3$. Based on these transition probabilities, we can construct the corresponding transition graph as shown in Fig. 1b. In the transition graph, each directed edge and the number on the edge represent the transition probability from one node to the other.

PHP penalizes the random walk for each additional step. Given a query node q, let $\mathbf{r}$ denote the PHP proximity vector, with $\mathbf{r}_i$ representing the proximity value of node i. PHP can be defined recursively as

$$\mathbf{r}_i = \begin{cases} 1, & \text{if } i = q, \\ c \sum_{j \in N_i} p_{i,j} \mathbf{r}_j, & \text{if } i \neq q, \end{cases}$$

where c ($0 < c < 1$) is the decay factor in the random walk process. In [6], $c = e^{-1}$ is used as the decay factor. The query node q has constant proximity value 1, and there is no transition probability going out of the query node. For example, there is no outgoing edges from the query node 1 in the transition graph in Fig. 1b.

Let $\mathbf{P}$ be the transition probability matrix with

$$\mathbf{P}_{i,j} = \begin{cases} 0, & \text{if } i = q, \\ p_{i,j}, & \text{if } i \neq q. \end{cases}$$

Then the above recursive definition can be expressed as the following matrix form

$$\mathbf{r} = c\mathbf{P}\mathbf{r} + \mathbf{e},$$

where $\mathbf{e}_i = 1$ if $i = q$, and $\mathbf{e}_i = 0$ if $i \neq q$.

Lemma 1. *PHP has no local maximum.*

Proof. Suppose that node i is a local maximum. We have $\mathbf{r}_i = c\sum_{j\in N_i} p_{i,j}\mathbf{r}_j \leq c\sum_{j\in N_i} p_{i,j}\mathbf{r}_i = c\mathbf{r}_i < \mathbf{r}_i$. We get a contradiction that $\mathbf{r}_i < \mathbf{r}_i$. □

The proofs for whether other proximity measures in Table 2 have local optimum can be found in the Appendices. In particular, in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2016.2515579, we show that EI, DHT, and THT have no local optimum. In Appendix G, available in the online supplemental material, we show that RWR, RT, KZ, and AP have local maximum.

Some proximity measures have inherent relationship. The following theorem says that PHP, EI, and DHT are equivalent in terms of ranking.

Theorem 2. *PHP, EI, and DHT give the same ranking results.*

Please see Appendix A, available in the online supplemental material, for the proof.

From the discussion in this section, we know that if a proximity measure has no local optimum, we can apply local search described in Algorithm 1 to find the top-k nodes under the assumption that the proximity values of all the nodes are given. However, without exploring the entire graph, the exact values of all the nodes are unknown. To tackle this challenge, we can develop lower and upper bounds on the proximity values of visited nodes. When more nodes are visited, the lower and upper bounds become tighter and eventually converge to the exact proximity value.

Next, we will use PHP, which has no local optimum, as a concrete example to explain how to derive the lower and upper bounds. The strategy can be applied to other proximity measures with no local optimum. How to apply the local search strategy to the measures having local optimum will be discussed in Section 6.

4 Bounding the Proximity

To develop the lower and upper bounds, we introduce three basic operations, i.e., deletion, restoration, and destination change of transition probability. We show how proximities change if we modify transition probabilities according to these operations. Then we discuss how to derive lower and upper bounds based on them.

4.1 Modifying Transition Probability

We first introduce the operation of deleting a transition probability. Fig. 2a shows an example, in which the original graph is on the top, with node 1 being the query node and

(a) deletion and restoration (b) destination change

Fig. 2. Basic operations on transition probability.

transition probabilities $p_{2,1} = p_{2,3} = 0.5$ and $p_{3,2} = 1$. After deleting $p_{2,3}$, the resulting graph is shown at the bottom. Note that deleting a transition probability is different from deleting an edge. Deleting an edge will change the transition probabilities on the remaining graph, while deleting a transition probability will not.

Theorem 3. *Deleting a transition probability will not increase the proximity of any node.*

Please see Appendix B, available in the online supplemental material, for the proof.

Continue with the example in Fig. 2a. Suppose that the decay factor $c = 0.5$. The original PHP proximity vector is $\mathbf{r} = [1, 2/7, 1/7]$. After deleting $p_{2,3}$, the new proximity vector is $\mathbf{r}' = [1, 1/4, 1/8]$.

It can be shown in a similar way that if we restore the transition probability as shown in Fig. 2a, the proximities will not decrease.

Theorem 4. *Restoring a deleted transition probability will not decrease the proximity of any node.*

Proof. Omitted. □

Fig. 2b shows an example in which we change the destination of the original transition probability $p_{3,2}$ from node 2 to 1. Thus, $p_{3,1}$ is set to $p_{3,2}$, and $p_{3,2}$ is set to 0.

Theorem 5. *Changing the destination of transition probability $p_{i,j}$ from node j to node u with $\mathbf{r}_u \geq \mathbf{r}_j$ ($\mathbf{r}_u \leq \mathbf{r}_j$) will not decrease (increase) the proximity of any node.*

Please see Appendix B, available in the online supplemental material, for the proof.

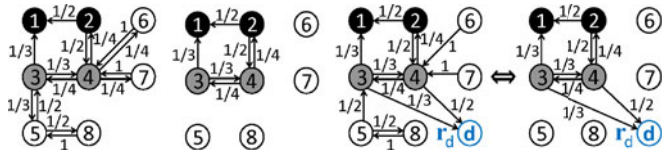
Let us continue with the example in Fig. 2b, where node 1 is the query node. After we change the destination of $p_{3,2}$ from node 2 to 1, the proximity values should be non-decreasing. With a decay factor $c = 0.5$, the proximity vectors before and after the destination change are $\mathbf{r} = [1, 2/7, 1/7]$ and $\mathbf{r}' = [1, 3/8, 1/2]$ respectively.

The proofs for other measures having no local optimum are similar and omitted here.

4.2 Lower Bound

Let S, $\bar{S}$, δS and $\delta\bar{S}$ represent the set of visited nodes, the set of unvisited nodes, the boundary of S, and the boundary of $\bar{S}$, respectively. From Theorem 3, if we delete all transition probabilities $\{p_{i,j} : i \text{ or } j \in \bar{S}\}$ in the original graph, the proximity value of any node u computed using the resulting graph, $\underline{\mathbf{r}}_u$, will be less than or equal to its original value $\mathbf{r}_u$, i.e., $\underline{\mathbf{r}}_u \leq \mathbf{r}_u$. Thus, $\underline{\mathbf{r}}$ can be used as the lower bound of $\mathbf{r}$.

Let us take the undirected graph in Fig. 1a for example. Its transition graph is shown in Fig. 3a, with node 1 being the query node and transition probabilities shown on the



(a) original $\mathbf{r}$ (b) LB $\underline{\mathbf{r}}$ (c) upper bound $\overline{\mathbf{r}}$

Fig. 3. Lower and upper bounds based on basic operations.

edges. Suppose that the current set of visited nodes $S = \{1, 2, 3, 4\}$. Thus $\bar{S} = \{5, 6, 7, 8\}$, $\delta S = \{3, 4\}$, and $\delta\bar{S} = \{5, 6, 7\}$. The nodes in S but not in δS are black. The nodes in δS are gray. The nodes in $\bar{S}$ are white.

Fig. 3b shows the resulting transition graph after deleting all transition probabilities $\{p_{i,j} : i \text{ or } j \in \bar{S}\}$. The proximity values $\underline{\mathbf{r}}$ computed based on Fig. 3b will lower bound the original proximity values $\mathbf{r}$ for the nodes in S.

4.3 Upper Bound

From Theorem 5, if we change the destination of the transition probabilities $\{p_{i,j} : i \in \delta S, j \in \delta\bar{S}\}$ to a newly added dummy node d with a constant proximity value $\mathbf{r}_\mathrm{d}$ and $\mathbf{r}_\mathrm{d} > \mathbf{r}_v$ $(\forall v \in \bar{S})$, the proximity value of any node u computed using the resulting graph, $\overline{\mathbf{r}}_u$, will be greater than or equal to its original value $\mathbf{r}_u$, i.e., $\overline{\mathbf{r}}_u \geq \mathbf{r}_u$. Thus, $\overline{\mathbf{r}}$ can be used as the upper bound of $\mathbf{r}$.

Continue with the example in Fig. 3a. In the original graph, $\delta S = \{3, 4\}$, $\delta\bar{S} = \{5, 6, 7\}$, $p_{3,5} = 1/3$, $p_{4,6} = p_{4,7} = 1/4$. The left figure in Fig. 3c shows the resulting graph after we change all the transition probabilities going from δS to $\delta\bar{S}$ to the newly added dummy node d. Specifically, $p_{3,\mathrm{d}} = 1/3$ and $p_{4,\mathrm{d}} = p_{4,6} + p_{4,7} = 1/2$. Note that after changing the destination to d, there will be no transition probability from any node in S to any node in $\bar{S}$. Therefore, for the nodes in S, the upper bounds can be computed by the subgraph induced by the nodes in S and the dummy node d, as shown on the right in Fig. 3c.

Note that to get the upper bound, we need to add a dummy node d with constant proximity value $\mathbf{r}_\mathrm{d} \geq \mathbf{r}_v$ $(\forall v \in \bar{S})$. In the next section, we will present the fast local search algorithm, FLoS, and discuss how to choose the value $\mathbf{r}_\mathrm{d}$.

5 Fast Local Search

In this section, we present the FLoS algorithm, which utilizes the bounds developed in Section 4 to enable the local search. We show that the bounds can only change monotonically when more nodes are visited.

5.1 The FLoS Algorithm

Algorithm 2 describes the FLoS algorithm. It has four main steps. In the first step, the algorithm expands locally to the neighbors of a selected visited node. In the second and third steps, it updates the lower and upper bounds of the visited nodes. Finally, it checks whether the top-k nodes are identified.

The local expansion step is shown in Algorithm 3. It picks the node in δS having the largest average lower and upper bound values, and expands to the neighbors of this node. S and δS are then updated accordingly. We take the average of the lower and upper bound value as an approximation of the exact proximity value. Expanding the node with the largest value is the best-first search strategy.

Algorithm 2. Fast Top-k Local Search (FLoS)

Input: $G(V, E)$, query q, number k, decay factor c, value τ
Output: Top-k node set K

$S^0 \leftarrow \{q\}$; $\delta S^0 \leftarrow \{q\}$; $\underline{\mathbf{r}}_q^0 \leftarrow 1$; $\overline{\mathbf{r}}_q^0 \leftarrow 1$; $\mathbf{P}_{q,q}^0 \leftarrow 0$;
bStop $\leftarrow$ false; $t \leftarrow 0$;
while bStop = false **do**
  $t \leftarrow t + 1$;
  LocalExpansion();
  UpdateLowerBound();
  UpdateUpperBound();
  CheckTerminationCriterion();
return K;

Algorithm 3. LocalExpansion()

$u \leftarrow \mathrm{argmax}_{i \in \delta S^{t-1}} (\underline{\mathbf{r}}_i^{t-1} + \overline{\mathbf{r}}_i^{t-1})$;
$S^t \leftarrow S^{t-1} \cup N_u$;
Update δS^t;

Algorithm 4 shows how to update the lower bound by using PHP as an example. It can also be applied to other measures with no local optimum. To update the bounds, we first construct the transition matrix $\mathbf{P}$. Note that the size of $\mathbf{P}$ is $|S| \times |S|$ instead of $|V| \times |V|$. We do not allocate memory for the full matrix $\mathbf{P}$, but only use adjacency list to represent it. The lower bound vector $\underline{\mathbf{r}}$ is initiated the same as in the previous iteration or 0 for the newly added nodes. We then use the standard iterative method, which is shown in Algorithm 7, to solve the linear equation $\underline{\mathbf{r}} = c\mathbf{P}\underline{\mathbf{r}} + \mathbf{e}$ and update $\underline{\mathbf{r}}$.

Algorithm 4. UpdateLowerBound()

$\mathbf{P}_{i,j}^t \leftarrow w_{i,j} / \sum_{v \in N_i} w_{i,v}$, if node i or j are newly added;
$\mathbf{P}_{q,j}^t \leftarrow 0$, if node j is newly added;
$\mathbf{P}_{i,j}^t \leftarrow \mathbf{P}_{i,j}^{t-1}$, if nodes i and j exist in the last iteration;
$\underline{\mathbf{r}}_i^t \leftarrow 0$, if node i is newly added;
$\underline{\mathbf{r}}_i^t \leftarrow \underline{\mathbf{r}}_i^{t-1}$, if node i exists in the last iteration;
$\mathbf{e}_i \leftarrow 1$, if $i = q$; $\mathbf{e}_i \leftarrow 0$, otherwise;
$\underline{\mathbf{r}}^t \leftarrow$ IterativeMethod($\mathbf{P}^t$, $\underline{\mathbf{r}}^t$, $\mathbf{e}$, c, τ);

Algorithm 5. UpdateUpperBound()

Extend $\mathbf{P}^t$ with 1 column and 1 row for the dummy node d;
$\mathbf{P}_{i,\mathrm{d}}^t \leftarrow 1 - \sum_{j \in N_i \cap S^t} \mathbf{P}_{i,j}^t$, if node $i \in \delta S^t$;
$\mathbf{P}_{i,\mathrm{d}}^t \leftarrow 0$, if $i \in S^t \setminus \delta S^t$;
$\mathbf{P}_{\mathrm{d},i}^t \leftarrow 0$, for any node i;
$\overline{\mathbf{r}}_i^t \leftarrow 1$, if node i is newly added;
$\overline{\mathbf{r}}_i^t \leftarrow \overline{\mathbf{r}}_i^{t-1}$, if node i exists in the last iteration;
$\overline{\mathbf{r}}_\mathrm{d}^t \leftarrow \mathbf{r}_\mathrm{d}^t \leftarrow \max_{i \in \delta S^{t-1}} \overline{\mathbf{r}}_i^{t-1}$ `// dummy node value`
Extend $\mathbf{e}$ with 1 new element $\mathbf{e}_\mathrm{d} = \mathbf{r}_\mathrm{d}^t$ for the node d;
$\overline{\mathbf{r}}^t \leftarrow$ IterativeMethod($\mathbf{P}^t$, $\overline{\mathbf{r}}^t$, $\mathbf{e}$, c, τ);

Algorithm 5 shows how to update the upper bound. The transition matrix $\mathbf{P}$ has one additional dummy node d and its related transition probabilities $\{p_{i,\mathrm{d}} : i \in \delta S\}$. The values in $\overline{\mathbf{r}}$ are initiated the same as the values in the previous iteration or 1 for the newly added nodes. The smaller the value

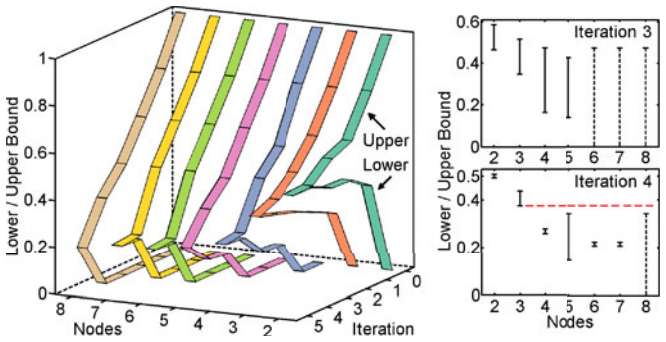


Fig. 4. Lower and upper bounds in different iterations (PHP: $q = 1$, $c = 0.8$).

of $\mathbf{r}_{\mathrm{d}}$, the tighter the upper bounds. On the other hand, we also need to make sure that the value $\mathbf{r}_{\mathrm{d}}$ is larger than the exact proximity value of any unvisited node. Therefore in line 7, we use the largest upper bound value in the boundary of the last iteration as $\mathbf{r}_{\mathrm{d}}^t$. We have $\mathbf{r}_{\mathrm{d}}^t = \max_{i \in \delta S^{t-1}} \bar{\mathbf{r}}_i^{t-1} \geq \max_{i \in \delta S^{t-1}} \mathbf{r}_i > \mathbf{r}_j (\forall j \in \bar{S}^{t-1})$, where the last inequality is based on Theorem 1. Thus $\mathbf{r}_{\mathrm{d}}^t > \mathbf{r}_j (\forall j \in \bar{S}^t)$, since $\bar{S}^t \subseteq \bar{S}^{t-1}$. This guarantees the correctness of the upper bound according to Theorem 5. Finally, we solve the linear equation $\bar{\mathbf{r}} = c\mathbf{P}\bar{\mathbf{r}} + \mathbf{e}$ to update $\bar{\mathbf{r}}$ by using Algorithm 7. Algorithm 7 is very efficient in practice. This is because when the initial values of the iterative method is close to the exact solution, the algorithm will converge very fast. In our method, between two adjacent iterations, the proximity values of the visited nodes are very close. Therefore, updating the proximity is very efficient.

Algorithm 6 shows the termination criterion. We select the k nodes in $S \setminus (\delta S \cup \{q\})$ with largest $\underline{\mathbf{r}}$ values. If the minimum lower bound of the selected nodes is greater than or equal to the maximum upper bound of the remaining visited nodes, the selected nodes will be the top-k nodes in the entire graph. This is because the maximum proximity of unvisited nodes is bounded by the maximum proximity in δS, which is in turn bounded by the maximum upper bound in δS.

Algorithm 6. CheckTerminationCriterion()

1: **if** $|S^t \setminus (\delta S^t \cup \{q\})| \geq k$ **then**
2: $\quad K \leftarrow k$ nodes in $S^t \setminus (\delta S^t \cup \{q\})$ with largest $\underline{\mathbf{r}}_i^t$;
3: $\quad$ **if** $\min_{i \in K} \underline{\mathbf{r}}_i^t \geq \max_{i \in S^t \setminus (K \cup \{q\})} \bar{\mathbf{r}}_i^t$ **then** bStop $\leftarrow$ true

Algorithm 7. IterativeMethod()

Input: matrix $\mathbf{P}$, vector $\mathbf{r}_{\mathrm{in}}$, vector $\mathbf{e}$, decay factor c, value τ
Output: proximity vector $\mathbf{r}_{\mathrm{out}}$
1: $\mathbf{r}^0 \leftarrow \mathbf{r}_{\mathrm{in}}$; $l \leftarrow 0$;
2: **repeat** $l \leftarrow l + 1$; $\mathbf{r}^l \leftarrow c\mathbf{P}\mathbf{r}^{l-1} + \mathbf{e}$; **until** $||\mathbf{r}^l - \mathbf{r}^{l-1}|| < \tau$;
3: **return** $\mathbf{r}^l$;

Fig. 4 shows the lower and upper bounds at different iterations using the example graph in Fig. 1a. One iteration represents one local expansion process. The newly visited nodes in each iteration are listed in Table 3.

The left figure in Fig. 4 shows how the lower and upper bounds change through local expansions. Query node 1 has constant proximity value 1.0 thus is not shown. It can be seen that the bounds monotonically change and eventually converge to the exact proximity value when all the nodes are visited. The monotonicity of the bounds is proved theoretically in next two sections.

TABLE 3
Newly Visited Nodes in Each Iteration

Iteration	1	2	3	4	5
Newly visited nodes	{2, 3}	{4}	{5}	{6, 7}	{8}

The right figure in Fig. 4 shows the lower and upper bounds in iteration 3 (at the top) and 4 (at the bottom). The interval from the lower to upper bounds is indicated by the vertical line segment. The interval of the bounds for the unvisited node is indicated by the dashed vertical line. In iteration 3, nodes $\{6, 7, 8\}$ are unvisited, and their upper bound is the upper bound for node 4, which is the maximum upper bound for the boundary nodes $\{4, 5\}$. In iteration 4, the bounds become tighter, and the minimum lower bound of nodes $\{2, 3\}$ is larger than the maximum upper bound of the remaining nodes $\{4, 5, 6, 7, 8\}$, which is indicated by the horizontal red dashed line. Therefore, nodes $\{2, 3\}$ are guaranteed to be the top-2 nodes after iteration 4, even though node 8 is still unvisited.

5.2 Monotonicity of the Lower Bound

We first consider the monotonicity of the lower bound. Let S^{t-1} and S^t represent the set of visited nodes in iterations $(t-1)$ and t respectively. In the next, we prove that the lower bound is monotonically non-decreasing when more nodes are visited, i.e., $\underline{\mathbf{r}}_i^t \geq \underline{\mathbf{r}}_i^{t-1} (i \in S^{t-1})$.

Given a directed transition graph, we say that a node u can *reach* a node v if there exists a sequence of adjacent nodes (i.e., a path) which starts from u and ends at v. For example, in the transition graph in Fig. 5a, node 1 can reach node 6, but node 5 cannot reach node 6. We use $u \rightsquigarrow v$ to denote that node u can reach v, and $u \not\rightsquigarrow v$ to denote that node u cannot reach v. We also use $u \rightsquigarrow S$ to denote that node u can reach at least one node in S, and $u \not\rightsquigarrow S$ to denote that node u cannot reach any node in S.

From iteration $(t-1)$ to t, we only restore some transition probabilities in $\{p_{i,j} : i \text{ or } j \in S^t \setminus S^{t-1}\}$. The following Theorem 6 says that if node i can reach at least one of the newly added nodes, the lower bound of node i is strictly increasing. If node i cannot reach any of the newly added nodes, the lower bound value of node i will not change during the iteration.

Fig. 5 shows an example. Fig. 5a shows the full transition graph when the query is node 3. Figs. 5b and 5c show the transition graphs constructed in the first and second iteration of Algorithm 2 respectively. $S^1 = \{3, 1, 4, 5\}$, $S^2 = \{3, 1, 4, 5, 2\}$, and node 2 is the newly visited node in the

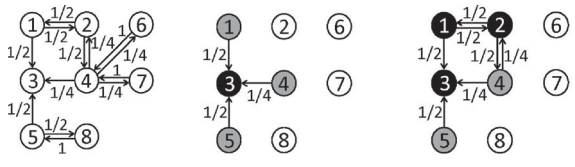
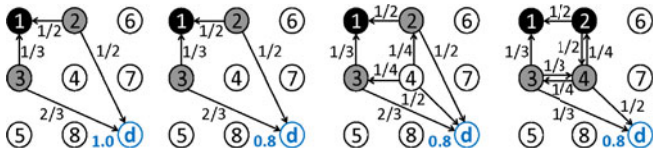


Fig. 5. Example transition graphs between two adjacent iterations for analyzing lower bound monotonicity with query node 3.



(a) 1st iteration (b) after step 1 (c) after step 2 (d) 2nd iteration

Fig. 6. Transition graphs between two adjacent iterations (upper bound).

second iteration. We can see that node 5 cannot reach node 2. Thus, the lower bound value $\underline{\mathbf{r}}_5$ is unchanged. The lower bound values of nodes $\{1, 4\}$ are strictly increasing.

Theorem 6 (Monotonicity of the lower bound). *For any node $i \in S^{t-1}$, we have that*

$$\begin{cases} \underline{\mathbf{r}}_i^t = \underline{\mathbf{r}}_i^{t-1}, & \text{if } i \nrightarrow S^t \setminus S^{t-1}, \\ \underline{\mathbf{r}}_i^t > \underline{\mathbf{r}}_i^{t-1}, & \text{if } i \rightsquigarrow S^t \setminus S^{t-1}, \end{cases}$$

where $\rightsquigarrow$ and $\nrightarrow$ represent the reachability in the transition graph at the tth iteration.

Please see Appendix C, available in the online supplemental material, for the proof.

5.3 Monotonicity of the Upper Bound

In this section, we analyze the monotonicity of the upper bound. Specifically, we prove that the upper bound values are strictly increasing until they converge to the exact proximity values. That is, for any node $i \in S^{t-1}$, $\overline{\mathbf{r}}_i^t < \overline{\mathbf{r}}_i^{t-1}$ until $\overline{\mathbf{r}}_i^{t-1} = \mathbf{r}_i$.

From iteration $(t-1)$ to t, we decrease the proximity value of the dummy node and add new nodes in $S^t \setminus S^{t-1}$. After adding the new nodes, the transition probabilities need to be updated accordingly. Specifically, we need to

1) Decrease the proximity value of the dummy node from $\overline{\mathbf{r}}_{\mathrm{d}}^{t-1}$ to $\overline{\mathbf{r}}_{\mathrm{d}}^t$;
2) Add the transition probabilities $\{p_{i,j}\}$ from the newly added nodes $i(\in S^t \setminus S^{t-1})$ to nodes $j(\in S^t)$, and $\{p_{i,\mathrm{d}}\}$ from i to the dummy node d;
3) Add the transition probabilities $\{p_{j,i}\}$ from nodes $j(\in \delta S^{t-1})$ to the newly added nodes $i(\in S^t \setminus S^{t-1})$, and remove their correspondences in $\{p_{j,\mathrm{d}}\}$.

An example is shown in Fig. 6. Fig. 6a shows the transition graph for the first iteration. Figs. 6b, 6c and 6d show the resulting graphs after applying steps 1, 2, and 3 respectively. The graph in Fig. 6d is the final transition graph for the next iteration.

The upper bound values will monotonically change at each step. In step 1, reducing the value of $\mathbf{r}_{\mathrm{d}}$ will not increase the upper bound values. Applying step 2 will not change the upper bound values for the nodes in S^{t-1}, since all the newly added transition probabilities begin from nodes $i(\in S^t \setminus S^{t-1})$. In step 3, we resets the transition probabilities from nodes $j(\in \delta S^{t-1})$ to the newly added nodes $i(\in S^t \setminus S^{t-1})$. This is equivalent to destination change, i.e., changing $\{p_{j,\mathrm{d}}\}$ to $\{p_{j,i}\}$. Moreover, we have that $\overline{\mathbf{r}}_{\mathrm{d}}^t \geq \overline{\mathbf{r}}_i^t$. Thus, in step 3, the upper bound values will not increase. We provide rigorous analysis for the three steps in Appendix D, available in the online supplemental material.

Theorem 7 (Monotonicity of the upper bound). *For any node $i \in S^{t-1}$, we have that*

$$\begin{cases} \overline{\mathbf{r}}_i^{t-1} = \overline{\mathbf{r}}_i^t, & \text{if } i \nrightarrow \mathrm{d}, \\ \overline{\mathbf{r}}_i^{t-1} > \overline{\mathbf{r}}_i^t, & \text{if } i \rightsquigarrow \mathrm{d}, \end{cases}$$

where $\rightsquigarrow$ and $\nrightarrow$ represent the reachability in the transition graph at the tth iteration.

Please see Appendix D, available in the online supplemental material, for the proof.

If we have that $i \nrightarrow \mathrm{d}$ in the transition graph at the tth iteration, we have that $i \nrightarrow \mathrm{d}$ in the transition graph at any future iteration. Thus, $\overline{\mathbf{r}}_i^t$ will not change during the future iterations. Since $\overline{\mathbf{r}}_i^t$ converges to the exact proximity value $\mathbf{r}_i$ when the entire graph is visited. We must have that $\overline{\mathbf{r}}_i^t = \mathbf{r}_i$ when $i \nrightarrow \mathrm{d}$. In conclusion, the upper bound strictly decreases until it converges to the exact proximity value.

The lower and upper bounds can be further tightened by adding self-loop transition probabilities to the nodes in δS. Please see Appendix E, available in the online supplemental material, for more details.

5.4 Complexity

Assume Algorithm 2 executes in β iterations. Let h be the average number of neighbors of a node. The LocalExpansion step takes $O(ht)$ time to find the node to expand at the tth iteration. To update the lower bound, updating $\mathbf{P}$ needs $O(h^2)$ operations, and updating $\underline{\mathbf{r}}$ and $\mathbf{e}$ needs $O(h)$ operations. Subgraph induced by S has $O(h^2t)$ edges, so matrix $\mathbf{P}$ has $O(h^2t)$ non-zero entries. Therefore using the iterative method to solve linear equations takes $O(\alpha h^2 t)$ time, where α is the number of iterations used in IterativeMethod. Thus the overall complexity of UpdateLowerBound in the tth iteration is $O(\alpha h^2 t)$. The complexity of UpdateUpperBound function is the same as that of UpdateLowerBound. In the CheckTerminationCriterion step, finding the nodes with largest lower bounds takes $O(ht)$ time. Therefore, the overall complexity of FLoS is $O(\sum_{t=1}^{\beta} (\alpha h^2 t + ht)) = O(\alpha h^2 \beta^2)$.

At each iteration, FLoS visits h new nodes on average. In the worst case, where the whole graph is visited, FLoS needs to run $\beta = n/h$ iterations. Thus, the worst case complexity of FLoS is $O(\alpha h^2 \beta^2) = O(\alpha n^2)$.

In the above complexity analysis, the number of iterations β is proportional to the number of visited nodes. Appendix F, available in the online supplemental material, provides theoretical analysis of the number of visited nodes. In Section 8, we show experimental results on the number of visited nodes using real graphs.

Note that so far, we have used PHP to illustrate the key principles underlying the fast local search method. EI and DHT are equivalent with PHP thus there is no need to develop algorithm for them. For THT, deleting a transition probability will not increase the proximity of any node. Therefore, when we delete all the transition probabilities $\{p_{i,j} : i \text{ or } j \in \bar{S}\}$ in the original transition graph, the proximity value of any node computed based on the modified transition graph will be the lower bound. For the upper bound, we add a dummy node with value 1, which is the largest

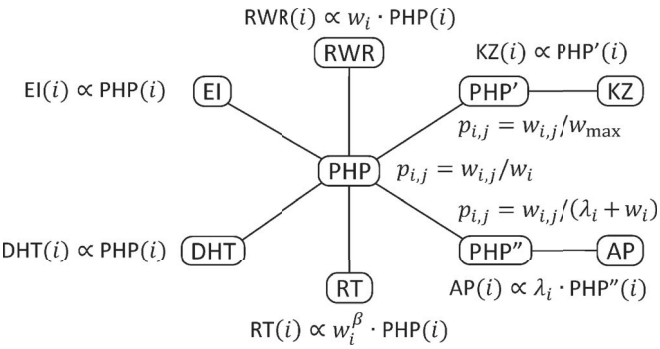


Fig. 7. Relationships between PHP and other proximity measures.

possible proximity value of THT. All other processes are similar to those of PHP and omitted here.

6 Extensions of FLoS to the Proximity Measures Having Local Maximum

In this section, we study how to extend the FLoS method to random walk with restart, RoundTripRank, Katz score, and absorption probability.

The idea is to use the relationships between PHP and these proximity measures. Fig. 7 summarizes the relationships between PHP and other proximity measures. For RWR, its proximity is proportional to the PHP proximity multiplied by the node degree. For RT, its proximity is proportional to the PHP proximity multiplied by the node degree to the power of β, where β is a constant in RT. For KZ, we define a new proximity measure PHP′, which is a variant of PHP. There is a simple relationship between KZ and PHP′. For AP, we define another new proximity measure PHP″, which is also a variant of PHP. There is a simple relationship between AP and PHP″. Compared with PHP, the transition probabilities in PHP′ and PHP″ are changed. In PHP, the transition probability is normalized by the node degree w_i. In PHP′, the transition probability is normalized by the maximum degree $w_{\max}$. In PHP″, the transition probability is normalized by the value $(\lambda_i + w_i)$, where λ_i is a constant in AP. Thus, the FLoS algorithm for PHP can be readily modified for PHP′ and PHP″. Appendix G, available in the online supplemental material, provides proofs for these relationships.

Next, we use RWR as an example to show how to extend FLoS to these proximity measures by using the relationship. Suppose that node $v \in \delta S^t$ has the largest PHP proximity value. Based on Theorem 1, for any node $i \in \bar{S}^t$, we have that $\text{PHP}(i) \leq \text{PHP}(v)$. Let $w(\bar{S}^t)$ denote the maximum degree of unvisited nodes in $\bar{S}^t$. We have that $w_i \cdot \text{PHP}(i) \leq w(\bar{S}^t) \cdot \text{PHP}(i) \leq w(\bar{S}^t) \cdot \text{PHP}(v)$. Therefore, if we maintain the maximum degree of unvisited nodes, we can develop the upper bound for the proximity values of unvisited nodes.

Specifically, we can apply FLoS to RWR as follows. In Algorithm 3, we can change line 1 to the following line.

1: $u \leftarrow \text{argmax}_{i \in \delta S^{t-1}} w_i \cdot (\underline{\mathbf{r}}_i^{t-1} + \overline{\mathbf{r}}_i^{t-1})$;

In Algorithm 6, we can change line 2 and 3 to the following two lines.

2: $K \leftarrow k$ nodes in $S^t \setminus (\delta S^t \cup \{q\})$ with largest $w_i \cdot \underline{\mathbf{r}}_i^t$ values;

3: **if** $\min_{i \in K} w_i \cdot \underline{\mathbf{r}}_i^t \geq \max_{i \in S^t \setminus (K \cup \{q\})} w_i \cdot \overline{\mathbf{r}}_i^t$ **and** $\min_{i \in K} w_i \cdot \underline{\mathbf{r}}_i^t \geq w(\bar{S}^t) \cdot \max_{i \in \delta S^t} \overline{\mathbf{r}}_i^t$ **then** bStop $\leftarrow$ true

All other processes remain the same.

For other proximity measures, we extend the FLoS algorithm in a similar way. Please see Appendix G, available in the online supplemental material, for further details.

7 Top-K Reverse-Proximity Query Problem

In this section, we study the top-k reverse-proximity query problem [24] and discuss how FLoS can be applied to solve it efficiently.

Given a query node q, we can compute the proximity values of all other nodes. We can also use each node i as the query, and compute the proximity value of q. We refer to this proximity of node q as the reverse proximity of node i. The top-k reverse-proximity query problem aims at finding the top-k nodes that are ranked by the reverse proximity. In Table 2, only EI and KZ are symmetric and all other proximity measures are not symmetric. The top-k reverse-proximity query problem is different from the top-k proximity query problem when the proximity measure is not symmetric.

Note that the top-k reverse-proximity query problem is different from the reverse top-k problem studied in [25]. Given a query node q, the reverse top-k problem aims at finding all the nodes that have q in their top-k proximity sets. In this paper, we study the top-k reverse-proximity query problem, which aims at finding the top-k nodes ranked by the reverse proximity [24].

The top-k reverse-proximity query problem has been studied when RWR is used as the proximity measure [24]. In a recent paper [10], the original and reverse proximity values in RWR are interpreted as importance and specificity respectively. If node i has large RWR proximity value when the query node is q, node i is important for node q. On the other hand, if node q has large RWR proximity value when the query node is i, node i is specific for node q. The authors show that ranking by the combination of two directions performs better than ranking by one direction.

The naive method to solve the top-k reverse-proximity query problem is as follows. First, each node is used as the query node, and the proximity value of node q is computed by the iterative method. Then the top-k nodes with largest reverse proximity values are selected. Suppose that the iterative method takes $O(\alpha m)$ for each query node. The naive method takes time $O(\alpha mn)$, where α is the number of iterations in the iterative method, m is the number of edges, and n is the number of nodes. This is expensive and prohibitive for large graphs.

For the RWR proximity measure, it is shown that the reverse proximity vector can be computed using the iterative method, which has the same complexity $O(\alpha m)$ as computing the original proximity vector [25]. However, the iterative method is still expensive since it needs to iterate over the entire graph. Moreover, it is unclear how to compute the reverse proximity vectors for other measures in a similar way.

To extend FLoS to the reverse proximity measures, we use the relationships between PHP and the reverse proximity measures. Fig. 8 summarizes these relationships. rPHP, rRWR, rEI, rDHT, rRT, rKZ, and rAP represent the reversed version of their corresponding measures. Appendix H, available in the online supplemental material, provides the

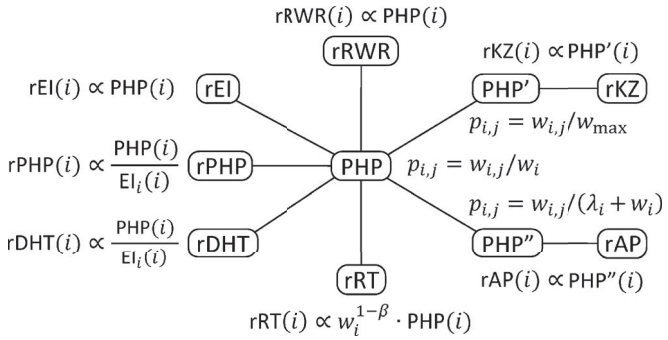


Fig. 8. Relationships between PHP and the reverse proximity measures.

proofs for these relationships. Based on these relationships, we can develop the bounds for the reverse proximity values based on the bounds for the PHP or its variant proximity values. Appendix H, available in the online supplemental material, shows more details about how to extend the FLoS algorithm to the reverse proximity measures.

8 Experimental Results

In this section, we present extensive experimental results on evaluating the performance of the FLoS algorithm. The datasets are shown in Table 4. The real datasets are publicly available from the website `http://snap.stanford.edu/data/`. The synthetic datasets are generated using the Erdös-Rényi random graph (RAND) model [26] and R-MAT model [27] with different parameters. All programs are written in C++. All experiments are performed on a server with 32G memory, Intel Xeon 3.2 GHz CPU, and Redhat 4.1.2 OS.

8.1 State-of-the-Art Methods

The measures we use include PHP, EI, RWR, RT, KZ, THT, and AP. We compare FLoS with the state-of-the-art methods for each measure as summarized in Table 5. These methods are categorized into global and local methods.

The global iteration (GI) method directly applies the iterative method on the entire graph [12]. It guarantees to find the exact top-k nodes. The graph embedding method can answer the query in constant time after embedding [19]. It can only be applied to RWR. However, the embedding process is very time consuming. Moreover, it only returns approximate results. The Castanet algorithm is specifically designed for RWR. It improves the GI method and guarantees the exactness of the results [4]. AA_KZ improves the global iteration method by prioritized execution of the iterative computation and also guarantees the exactness of the results [13]. K-dash is the state-of-the-art matrix-based method for RWR which guarantees result exactness [14]. Note that K-dash and GE can only be applied on two medium-sized real graphs because of the expensive preprocessing step.

TABLE 4
Datasets Used in the Experiments

Datasets		Abbr.	#Nodes	#Edges
Real	Amazon	AZ	334,863	925,872
	DBLP	DP	317,080	1,049,866
	Youtube	YT	1,134,890	2,987,624
	LiveJournal	LJ	3,997,962	34,681,189
Synthetic	In-memory	–	Varying size	
		–	Varying density	
	Disk-resident	–	Varying size	

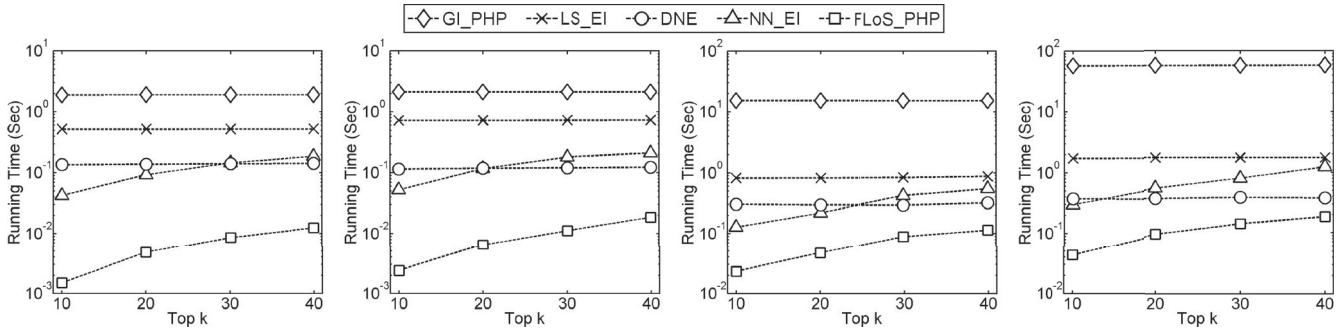
TABLE 5
State-of-the-Art Methods Used for Comparison

Our methods (Exact)	State-of-the-art methods			
	Abbr.	Key idea	Ref.	Exactness
FLoS_PHP	GI_PHP	Global iteration	[12]	Exact
	DNE	Local search	[7]	Approx.
	NN_EI	Local search	[9]	Exact
	LS_EI	Local search	[1]	Approx.
FLoS_RWR	GI_RWR	Global iteration	[12]	Exact
	GE_RWR	Graph embedding	[19]	Approx.
	Castanet	Improved GI	[4]	Exact
	K-dash	Matrix inversion	[14]	Exact
	LS_RWR	Local search	[1]	Approx.
FLoS_RT	GI_RT	Global iteration	[12]	Exact
	LS_RT	Local search	[10]	Approx.
FLoS_KZ	GI_KZ	Global iteration	[12]	Exact
	LS_KZ	Local search	[23]	Approx.
	AA_KZ	Improved GI	[13]	Exact
FLoS_THT	GI_THT	Global iteration	[12]	Exact
	LS_THT	Local search	[5]	Approx.
FLoS_AP	GI_AP	Global iteration	[12]	Exact
FLoS_rPHP	GI_rPHP	Global iteration	[12]	Exact

Dynamic neighborhood expansion (DNE) method applies a best-first expansion strategy to find the top-k nodes using PHP [7]. This strategy is heuristic and does not guarantee to find the exact solution. The number of visited nodes is fixed to $4,000$ in the experiments. NN_EI applies the push style method [2], [21] in local search, and guarantees the exactness of the top-k results [9]. Since PHP and EI are equivalent in terms of ranking, we can compare the methods for PHP and EI directly. LS_RWR applies the dynamic programming technique [28] to develop bounds in local search [1]. It returns approximate results. LS_EI is based on LS_RWR and has similar performance [1]. LS_RT leverages the push style method [21] developed for RWR to estimate the bounds and find the approximate top-k nodes with largest RoundTripRank proximity values. LS_KZ locally searches a small portion of the graph and adapts the push style method [21] to find the approximate top-k results for the Katz score. LS_THT is a local search method for THT [5].

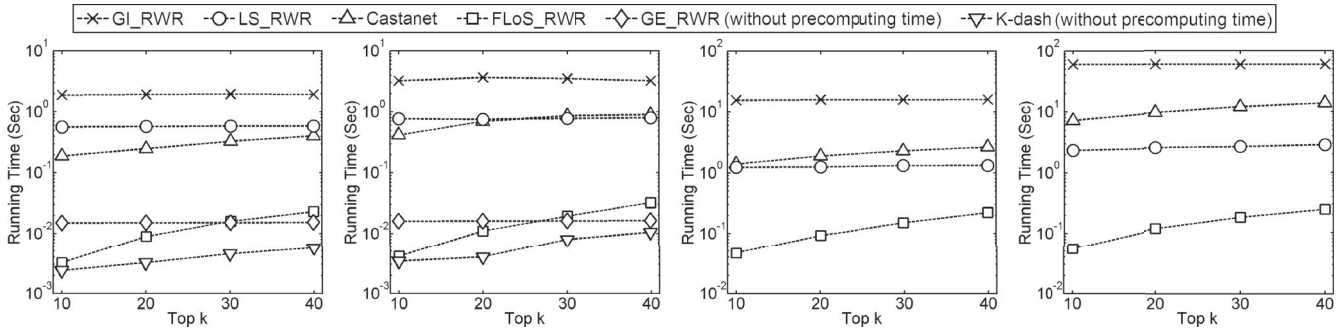
The decay factors in PHP, RWR, EI, and RT are all set to 0.5. The decay factor in KZ is set to $0.99/w_{\max}$. In RT, we set the parameter $\beta = 0.4$. In AP, we set the parameter $\lambda_i = 10$ for any node i. The truncated length in THT is set to 10.

We use FLoS_rPHP to denote the FLoS method for reverse PHP. Reverse RWR gives the same ranking as PHP, so we only evaluate FLoS_PHP. The FLoS method for reverse RT is quite similar to that for RT, thus we only evaluate FLoS_RT. When we set the parameter $\lambda_i = 10$ for any node i, AP becomes symmetric. Thus AP and reverse AP give the same ranking results, and we only evaluate FLoS_AP.



(a) AZ (b) DP (c) YT (d) LJ

Fig. 9. Running time of different methods for PHP on real graphs.



(a) AZ (b) DP (c) YT (d) LJ

Fig. 10. Running time of different methods for RWR on real graphs.

8.2 Evaluation on Real Graphs

We study the efficiency of the selected methods on real graphs when varying the number of returned nodes k. For each k, we repeat the experiments 10^3 times, each with a randomly picked query node. The average running time is reported. For methods using the iteration procedure in Algorithm 7, the termination threshold is set to $\tau = 10^{-5}$. We also perform experiments using a fixed number of 10 iterations. The results are similar and omitted here.

8.2.1 Evaluation of FLoS_PHP

Fig. 9 shows the running time of different methods for PHP. The running time of DNE is almost a constant for different k, because it visits a fixed number of nodes. The running time of NN_EI increases when k increases. FLoS_PHP is more efficient than NN_EI, which demonstrates that the bounds of FLoS are tighter. LS_EI has a constant running time. This is because it extracts the cluster containing the query node. Note that LS_EI takes tens of hours in the preprocessing step to cluster the graphs.

Fig. 11a shows the ratio between the number of visited nodes using FLoS_PHP and total number of nodes in the graph. The value indicated by the bar is the average ratio of 10^3 queries. The minimum and maximum ratios are also shown in the figure. As can be seen from the figure, only a very small part of the graph is needed for FLoS to find the exact solution. Moreover, the ratio decreases when the graph size increases. This indicates that FLoS is more effective for larger graphs.

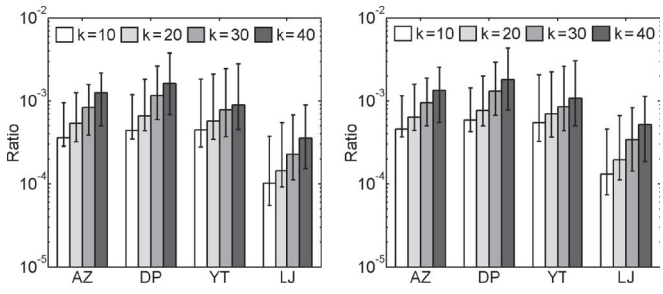
8.2.2 Evaluation of FLoS_RWR

Fig. 10 shows the running time for RWR. K-dash has the best performance after precomputing the matrix inversion as shown in Figs. 10a and 10b. The precomputing step of K-dash takes tens of hours for the medium-sized AZ and DP graphs and cannot be applied to the other two larger graphs. GE_RWR also has fast response time. However, as discussed before, its embedding step is time consuming and not applicable to larger graphs. Moreover, it does not find the exact solution. Castanet method cuts the running time from the GI method by 72 to 91 percent. LS_RWR method has constant running time, and it needs tens of hours in the precomputing step to cluster the graphs.

Fig. 11b shows the ratio of the number of visited nodes of the FLoS_RWR method. The results are similar to that of Fig. 11a.

8.2.3 Evaluation of FLoS_RT

Fig. 12a shows the running time of different methods for RT. The number on the right side of the rectangle legend indicates the value of k. Since the GI_RT method has almost constant running time for different k, we only show the result



(a) FLoS_PHP (b) FLoS_RWR

Fig. 11. Ratio between the number of visited nodes and the total number of nodes on real graphs.

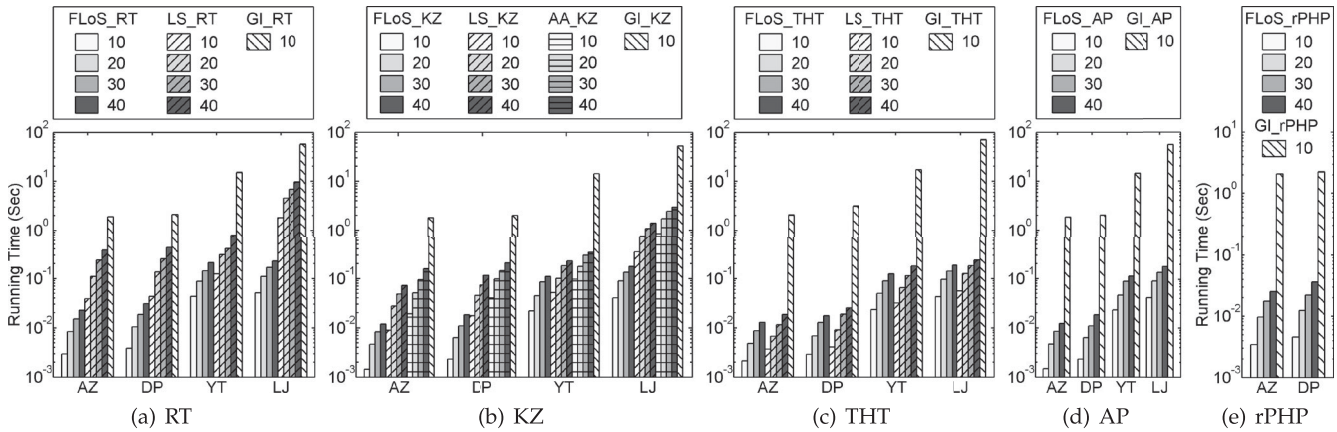


Fig. 12. Running time of different methods for RT, KZ, THT, AP, and rPHP on real graphs.

when $k = 10$. The running time of FLoS_RT and LS_RT increases when k increases. FLoS_RT is the most efficient method. FLoS_RT is about 1 order of magnitude faster than LS_RT, and 2 orders of magnitude faster than GI_RT. LS_RT uses the push style method to develop the bounds, which are looser than those of FLoS_RT.

8.2.4 Evaluation of FLoS_KZ

Fig. 12b shows the running time of different methods for KZ. We also only show the running time when $k = 10$ for the GI_KZ method since it has almost constant running time for different k. FLoS_KZ, LS_KZ, and AA_KZ methods all have increasing running time when increasing k. FLoS_KZ is about 1-2 orders of magnitude faster than LS_KZ and AA_KZ. LS_KZ uses the push style method to develop the bounds, which are not as tight as those of FLoS_KZ. The results also demonstrate that the bounds in AA_KZ are looser than those of FLoS_KZ.

8.2.5 Evaluation of FLoS_THT

Fig. 12c shows the running time for THT. As we can see, FLoS_THT runs faster than LS_THT, which is specifically designed to speed up the computation for THT. This is because the lower and upper bounds of FLoS_THT are tighter than those of LS_THT. Both of the two local search methods are 2 to 3 orders of magnitude faster than GI_THT.

8.2.6 Evaluation of FLoS_AP

Fig. 12d shows the running time for AP. Similar to the results of other proximity measures, FLoS_AP runs 2-3 orders of magnitude faster than the GI_AP method.

8.2.7 Evaluation of FLoS_rPHP

In FLoS_rPHP, we pre-compute the exact values $\mathrm{EI}_i(i)$ for each node i by the K-dash method [14]. The precomputation step takes 28.5 and 34.6 hours for two medium-sized graphs, AZ and DP. Thus we did not apply FLoS_rPHP on the large graphs.

Fig. 12e shows the running time of our local search method and the global iteration method for reverse PHP. Similar to the results for other proximity measures, FLoS_rPHP runs 2-3 orders of magnitude faster than the GI_rPHP method.

8.2.8 Number of Visited Nodes in Local Search Methods

In this section, we study the number of visited nodes of different local search methods on real graphs. The number of visited nodes in the DNE method is fixed, thus it is not included. Fig. 13a shows the ratio between the number of visited nodes using different local search methods and total number of nodes in the YT graph. Fig. 13b shows that in the LJ graph. The value indicated by the bar is the average ratio of 10^3 queries. The minimum and maximum ratios are also shown in the figure. As can be seen from the figure, other local search methods need to visit larger number of nodes than the FLoS methods do. This demonstrates the tightness of the bounds in the FLoS methods. We also can observe that the LS_EI and LS_RWR methods visit relatively large number of nodes and the ratio is stable when the number k changes. This is because in each expansion of the LS_EI and LS_RWR methods, all the nodes in one cluster will be visited. Thus, they need to visit larger number of nodes.

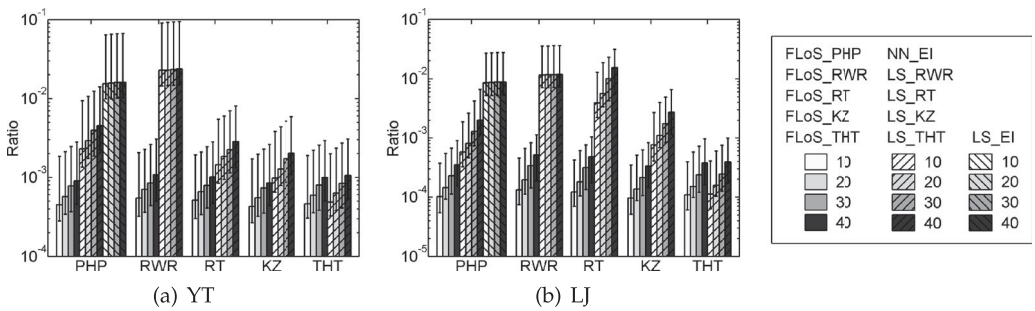


Fig. 13. Ratio between the number of visited nodes and the total number of nodes on real graphs for the local search methods.

TABLE 6
Statistics of In-Memory Synthetic Graphs

Varying size	#Nodes	1×2^{20}	2×2^{20}	4×2^{20}	8×2^{20}
	#Edges	1×10^7	2×10^7	4×10^7	8×10^7
	Density	9.5	9.5	9.5	9.5
Varying density	#Nodes	1×2^{20}	1×2^{20}	1×2^{20}	1×2^{20}
	#Edges	5×10^6	10×10^6	15×10^6	20×10^6
	Density	4.8	9.5	14.3	19.1

8.3 Evaluation on In-Memory Synthetic Graphs

We generate synthetic graphs with different parameters to evaluate the selected methods. More specifically, we study two types of graphs: Erdös-Rényi random graph (RAND) [26] and scale-free graph based on the R-MAT model [27]. There are two parameters, the size and density of the graphs. We study how these two parameters affect the running time of different methods for PHP, RWR, RT, and KZ.

We download the graph generator available from the website `https://github.com/dhruvbird/GTgraph` and use the default parameters to generate two series of graphs with varying size and varying density, using RAND and R-MAT respectively. The graphs with varying size have the same density but different number of nodes. The graphs with varying density have the same number of nodes but different densities. The statistics are shown in Table 6.

We apply the selected methods for PHP, RWR, RT and KZ on these graphs with $k = 20$. For each graph, we repeat the query 10^3 times with randomly picked query nodes, and report the average running time.

8.3.1 Evaluation of FLoS_PHP

Fig. 14a shows the running time of the selected methods for PHP on the series of RAND graphs with varying size. The running time of GI_PHP increases as the number of nodes increases. FLoS_PHP, DNE, NN_EI and LS_EI all have almost constant running time when the number of nodes increases. This is because these methods only search locally. When the density of the graph is fixed, adding more nodes to the graph will not change the size of the search space of these methods. Fig. 14b shows the running time on the series of R-MAT graphs with varying size. Similar trends are observed. Comparing Figs. 14a and 14b, GI_PHP has less running time on R-MAT than on RAND graphs, while other methods have more. The reason is that R-MAT graphs have the power-law distribution, thus it is easier for FLoS_PHP, DNE, NN_EI and LS_EI to encounter hub nodes with larger degree when expanding subgraph. The faster performance of GI_PHP on R-MAT may be because of the greater data locality due to the hub node.

Fig. 14c shows the running time of the selected methods for PHP on the series of RAND graphs with varying density. The running time of all the methods increases as the density increases. FLoS_PHP and NN_EI have increasing running time because the number of visited nodes in these two methods increases when the density becomes larger. LS_EI has increasing running time because the number of nodes and edges increases in local clusters. Fig. 14d shows the running time on the series of R-MAT graphs with varying density. Similar trends are observed.

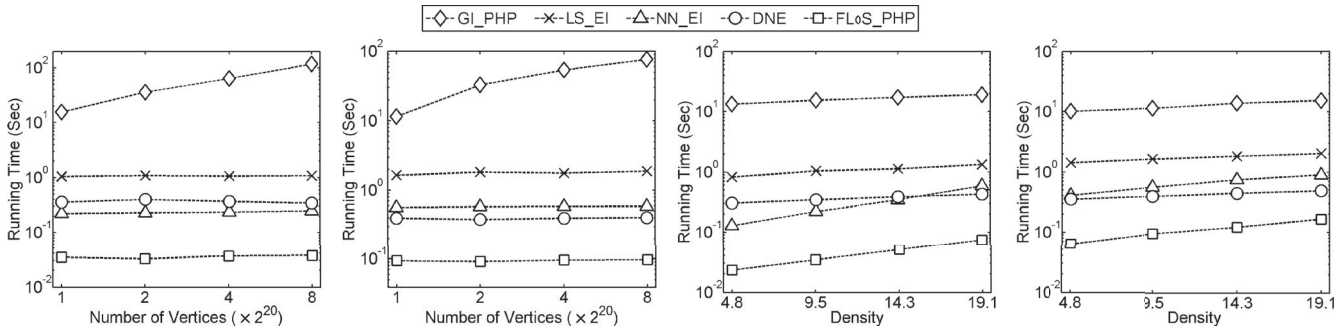
8.3.2 Evaluation of FLoS_RWR

Fig. 15a shows the running time of the selected methods for RWR on the series of RAND graphs with varying size. The running time of GI_RWR and Castanet increases as the number of nodes increases. Castanet method cuts the running time from the GI method by 69 to 88 percent. FLoS_RWR and LS_RWR both have almost constant running time when the number of nodes increases. This is because FLoS_RWR and LS_RWR only search locally. Fig. 15b shows the running time on the series of R-MAT graphs with varying size. Similar trends are observed. Comparing Figs. 15a and 15b, GI_RWR has less running time on the R-MAT graphs than on the RAND graphs, while other methods have more. The reason is similar as what discussed previously.

Fig. 15c shows the running time on the series of RAND graphs with varying density. The running time of all the methods increases as the density increases. Fig. 15d shows the running time on the series of R-MAT graphs with varying density. Similar trends are observed.

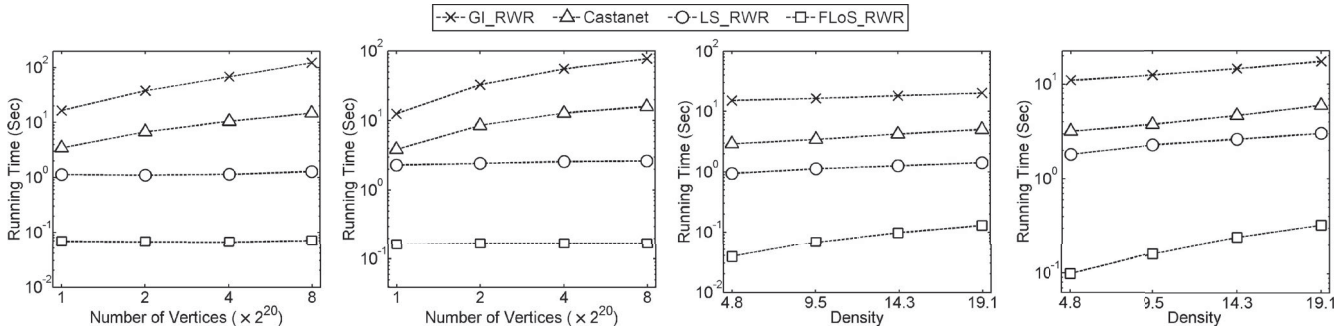
8.3.3 Evaluation of FLoS_RT

Fig. 16a shows the running time of the selected methods for RT on the series of R-MAT graph with varying size. The running time of GI_RT increases as the number of nodes increases. FLoS_RT has almost constant running time when the number of nodes increases. Because it only searches locally. LS_RT has increasing running time. LS_RT needs to find the node with the largest residual proximity value in each iteration. When the number of nodes in the graph increases, the search space may also increase. This may be the reason why it has a slightly increasing running time.



(a) varying size, RAND (b) varying size, R-MAT (c) varying density, RAND (d) varying density, R-MAT

Fig. 14. Running time of different methods for PHP on in-memory synthetic graphs ($k = 20$).



(a) varying size, RAND (b) varying size, R-MAT (c) varying density, RAND (d) varying density, R-MAT

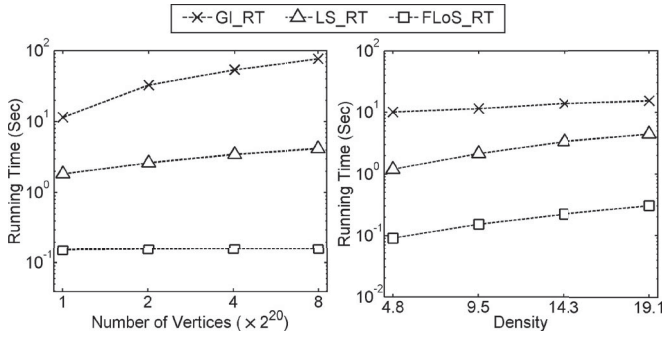
Fig. 15. Running time of different methods for RWR on in-memory synthetic graphs ($k = 20$).

Fig. 16b shows the running time of the selected methods for RT on the series of R-MAT graph with varying density. The running time of all the methods increases as the density increases. Both FLoS_RT and LS_RT have increasing running time because they will visit more nodes in a graph with larger density.

8.3.4 Evaluation of FLoS_KZ

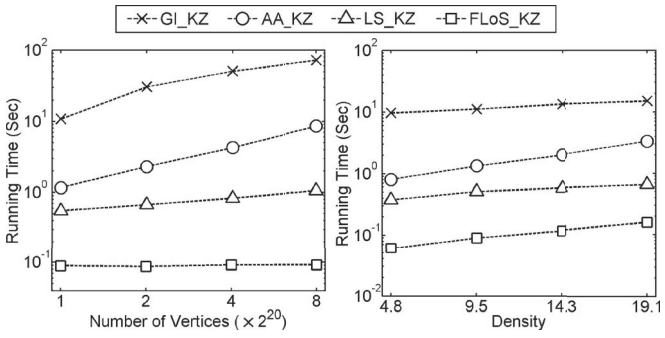
Fig. 17a shows the running time of the selected methods for KZ on the series of R-MAT graph with varying size. The running time of GI_KZ increases as the number of nodes increases. FLoS_KZ has almost constant running time when the number of nodes increases. LS_KZ and AA_KZ both have increasing running time when increasing graph size. LS_KZ needs to update the node with the largest residual proximity value in each iteration, thus it has a slightly increasing running time when the graph size increases. In AA_KZ, computing the upper bound of each node requires linear time $O(m)$. Thus it has increasing running time.

Fig. 17b shows the running time of the selected methods for KZ on the series of R-MAT graph with varying density. The running time of all the methods increases as the density increases. The reason why FLoS_KZ, LS_KZ and AA_KZ have increasing running time is that they need to visit more nodes when increasing the graph density.



(a) varying size (b) varying density

Fig. 16. Running time of different methods for RT on in-memory synthetic graphs (R-MAT, $k = 20$).



(a) varying size (b) varying density

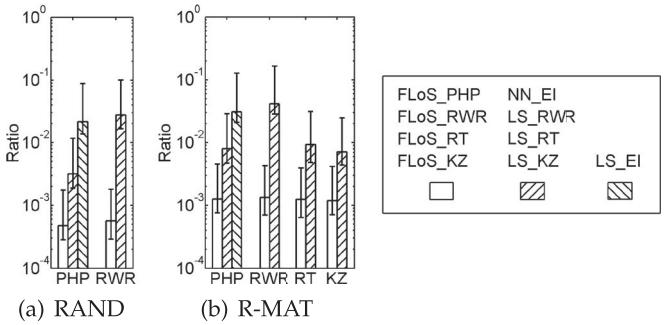
Fig. 17. Running time of different methods for KZ on in-memory synthetic graphs (R-MAT, $k = 20$).

8.3.5 Number of Visited Nodes in Local Search Methods

In this section, we study the number of visited nodes using different local search methods on synthetic graphs. We use the synthetic graphs with 2^{20} nodes and 10^7 edges. The number of query nodes is fixed to $k = 20$. Fig. 18a shows the ratio between the number of visited nodes using different local search methods for PHP and RWR and total number of nodes in the RAND graph. Fig. 18b shows the ratio between the number of visited nodes using different local search methods for PHP, RWR, RT and KZ and total number of nodes in the R-MAT graph. The value indicated by the bar is the average ratio of 10^3 queries. The minimum and maximum ratios are also shown in the figure. As can be seen from the figure, other local search methods need to visit larger number of nodes than the FLoS methods do. This demonstrates the tightness of the bounds in the FLoS methods.

8.4 Evaluation on Disk-Resident Synthetic Graphs

What if the graphs are too large to fit into memory? To test the performance of FLoS on disk-resident graphs, we

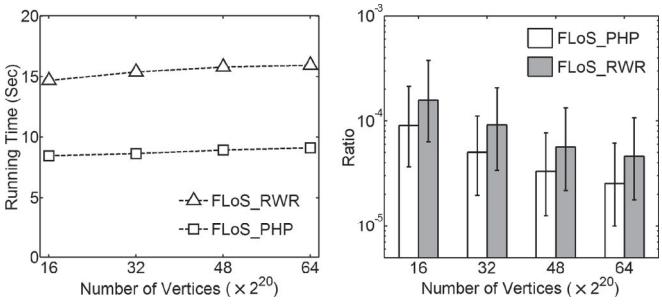


(a) RAND (b) R-MAT

Fig. 18. Ratio between the number of visited nodes and the total number of nodes on synthetic graphs for the local search methods (2^{20} nodes and 10^7 edges, $k = 20$).

TABLE 7
Statistics of Disk-Resident Synthetic Graphs

#Nodes	16×2^{20}	32×2^{20}	48×2^{20}	64×2^{20}
#Edges	16×10^7	32×10^7	48×10^7	64×10^7
Disk size	3.1 G	6.5 G	9.9 G	13.2 G



(a) running time (b) visited nodes

Fig. 19. Results of FLoS_PHP and FLoS_RWR on disk-resident synthetic graphs ($k = 20$).

generate disk-resident R-MAT graphs, whose statistics are in Table 7. We use the open source Neo4j (available from `http://www.neo4j.org`) version 2.0 graph database. The FLoS method for disk-resident graphs only calls some basic query functions provided by Neo4j, such as, querying the neighbors of one node. And the remaining work is the same as that for in-memory graphs. We apply the FLoS_PHP and FLoS_RWR methods on the disk-resident graphs with $k = 20$. We repeat the query 10^3 times with randomly picked query nodes and report the average running time. In the experiments, we restrict the memory usage to 2 GB.

Fig. 19a shows the running time of FLoS_PHP and FLoS_RWR. From the figure, we can see that FLoS can process disk-resident graphs in tens of seconds. The reason is that FLoS only needs to find the neighbors of visited nodes and the transition probabilities on the edges. These results also verify that FLoS has almost constant running time when the number of nodes increases. Fig. 19b shows the ratio of the number of visited nodes to the total number of nodes in the graph. FLoS only needs to explore a small portion of the whole graph to return the top-k nodes. When the graph size becomes larger, the portion of visited nodes becomes smaller.

9 Conclusion

Top-k nodes query in large graphs is a fundamental problem that has attracted intensive research interests. Existing methods need expensive preprocessing steps or are designed for specific proximity measures. In this paper, we propose a unified method, FLoS, which adopts a local search strategy to find the exact top-k nodes efficiently. FLoS is based on the no local optimum property of proximity measures. By exploiting the relationship among different proximity measures, we can also extend FLoS to the proximity measures having local optimum. FLoS can be further extended to solve the top-k reverse-proximity query problem. Extensive experimental results demonstrate that FLoS enables efficient and exact query for a variety of random walk based proximity measures.

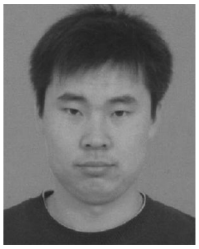
Acknowledgments

This work was partially supported by the US National Science Foundation grants IIS-1162374, IIS-1218036, IIS-0953950, the NIH/NIGMS grant R01GM103309, and the OSC (Ohio Supercomputer Center) grant PGS0218.

References

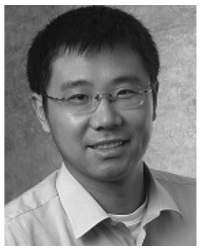
[1] P. Sarkar and A. W. Moore, "Fast nearest-neighbor search in disk-resident graphs," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 513–522.
[2] S. Chakrabarti, A. Pathak, and M. Gupta, "Index design and query processing for graph conductance search," *VLDB J.*, vol. 20, no. 3, pp. 445–470, 2011.
[3] P. Lee, L. V. Lakshmanan, and J. X. Yu, "On top-k structural similarity search," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 774–785.
[4] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka, "Efficient ad-hoc search for personalized PageRank," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 445–456.
[5] P. Sarkar and A. W. Moore, "A tractable approach to finding closest truncated-commute-time neighbors in large graphs," in *Proc. Conf. Uncertainty Artif. Intell.*, 2007, pp. 335–343.
[6] Z. Guan, J. Wu, Q. Zhang, A. Singh, and X. Yan, "Assessing and ranking structural correlations in graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 937–948.
[7] C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei, "Evaluating geo-social influence in location-based social networks," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 1442–1451.
[8] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *Proc. IEEE 11th Int. Conf. Data Mining*, 2006, pp. 613–622.
[9] P. Bogdanov and A. Singh, "Accurate and scalable nearest neighbors in large networks based on effective importance," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 523–528.
[10] Y. Fang, K.-C. Chang, and H. W. Lauw, "RoundTripRank: Graph-based proximity with importance and specificity," in *Proc. IEEE Int. Conf. Data Eng.*, 2013, pp. 613–624.
[11] X.-M. Wu, Z. Li, A. M. So, J. Wright, and S.-F. Chang, "Learning with partially absorbing random walks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 3077–3085.
[12] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: SIAM, 2003.
[13] S. Khemmarat and L. Gao, "Fast top-k path-based relevance query on massive graphs," in *Proc. Int. Conf. Data Eng.*, 2014, pp. 316–327.
[14] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa, "Fast and exact top-k search for random walk with restart," *Proc. VLDB Endowment*, vol. 5, no. 5, pp. 442–453, 2012.
[15] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka, "Efficient personalized PageRank with accuracy assurance," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 15–23.
[16] S. Cohen, B. Kimelfeld, and G. Koutrika, "A survey on proximity measures for social networks," in *Search Computing*. New York, NY, USA: Springer, 2012, pp. 191–206.
[17] P. Sarkar, A. W. Moore, and A. Prakash, "Fast incremental proximity search in large graphs," in *Proc. Int. Conf. Mach. Learn.*, 2008, pp. 896–903.
[18] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.
[19] X. Zhao, A. Chang, A. D. Sarma, H. Zheng, and B. Y. Zhao, "On the embeddability of random walk distances," *Proc. VLDB Endowment*, vol. 6, no. 14, pp. 1690–1701, 2013.
[20] Q. Mei, D. Zhou, and K. Church, "Query suggestion using hitting time," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2008, pp. 469–478.
[21] P. Berkhin, "Bookmark-coloring algorithm for personalized PageRank computing," *Internet Math.*, vol. 3, no. 1, pp. 41–62, 2006.
[22] M. Gupta, A. Pathak, and S. Chakrabarti, "Fast algorithms for top-k personalized PageRank queries," in *Proc. World Wide Web*, 2008, pp. 1225–1226.
[23] P. Esfandiar, F. Bonchi, D. F. Gleich et al., "Fast Katz and commuters: Efficient estimation of social relatedness in large networks," in *Proc. Algorithms Models Web-Graph*, 2010, pp. 132–145.

[24] A. A. Benczur, K. Csalogany, T. Sarlos, and M. Uher, "SpamRank - Fully automatic link spam detection work in progress," in *Proc. AIRWeb*, 2005, pp. 25–38.
[25] A. W. Yu, N. Mamoulis, and H. Su, "Reverse top-k search using random walk with restart," *Proc. VLDB Endowment*, vol. 7, no. 5, pp. 401–412, 2014.
[26] P. Erdős and A. Rényi, "On the evolution of random graphs," *Magyar Tud. Akad. Mat. Kutató Int. Közl*, vol. 5, pp. 17–61, 1960.
[27] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *Proc. SIAM Int. Conf. Data Mining*, 2004, pp. 442–446.
[28] G. Jeh and J. Widom, "Scaling personalized web search," in *Proc. World Wide Web*, 2003, pp. 271–279.
[29] C. Meyer, *Matrix Analysis and Applied Linear Algebra*. Philadelphia, PA, USA: SIAM, 2000.
[30] E. A. Guillemin, *Introductory Circuit Theory*. New York, NY, USA: Wiley, 1953.
[31] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2002, pp. 538–543.



Yubao Wu received the bachelor's and master's degrees both from Dalian University of Technology, China. He is a fourth year PhD student in the Department of Electrical Engineering and Computer Science, Case Western Reserve University. His research interests include big data analytics, data mining, and bioinformatics.

Ruoming Jin received the doctor's degree in computer science from the Ohio State University in 2005. He is an associate professor in the Department of Computer Science at Kent State University. His research interests are on data mining, database, biomedical informatics, and cloud computing.



Xiang Zhang received the doctor's degree in computer science from the University of North Carolina at Chapel Hill in 2011. He is the T&D Schroeder assistant professor in the Department of Electrical Engineering and Computer Science at Case Western Reserve University. His research bridges the areas of data mining, database, and bioinformatics.