Fast and Unified Local Search for Random Walk Based K-Nearest-Neighbor Query in Large Graphs

Yubao Wu [†], Ruoming Jin [‡], Xiang Zhang [†]
[†]Department of Electrical Engineering and Computer Science, Case Western Reserve University
[‡]Department of Computer Science, Kent State University
yubao.wu@case.edu, jin@cs.kent.edu, xiang.zhang@case.edu

ABSTRACT

Given a large graph and a query node, finding its k-nearest-neighbor (kNN) is a fundamental problem. Various random walk based measures have been developed to measure the proximity (similarity) between nodes. Existing algorithms for the random walk based top-k proximity search can be categorized as *global* and *local* methods based on their search strategies. Global methods usually require an expensive precomputing step. By only searching the nodes near the query node, local methods have the potential to support more efficient query. However, most existing local search methods cannot guarantee the exactness of the solution. Moreover, they are usually designed for specific proximity measures.

Can we devise an efficient local search method that applies to different measures and also guarantees result exactness? In this paper, we present FLoS (Fast Local Search), a unified local search method for efficient and exact top-k proximity query in large graphs. 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 simple operations on transition probabilities, which allow developing lower and upper bounds on the proximity. The bounds monotonically converge to the exact proximity when more nodes are visited. We further show that FLoS can also be applied to measures having local optimum by utilizing relationship among different measures. We perform comprehensive experiments to evaluate the efficiency and applicability of the proposed method.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

Keywords

Local search; nearest neighbors; top-k search; random walk; proximity search

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2610500.

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 in the database research community [18, 5, 13, 9]. In general, there are two challenges in this problem. One is to design proximity measures that can effectively capture the similarity between nodes. Another is to devise efficient algorithms to compute the top-k nodes for a given measure.

Designing effective proximity (similarity) measures is a difficult task. Random walk based measures have been shown to be effective in many applications. Some examples include discounted/truncated hitting time [17, 18], penalized hitting probability [11, 21], and random walk with restart [1, 20, 3].

Although various proximity measures have been developed, how to efficiently compute them remains a challenging problem. For most random walk based measures, the 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 [16, 9]. Another approach precomputes and stores the inversion of a matrix [20, 8, 10]. The precomputing step is usually expensive and needs to be repeated whenever the graph changes.

To improve efficiency, local methods dynamically expand the search range to visit nodes near the query node [17, 19, 18, 21]. Node proximities are estimated based on local information only. Without using the global information, however, most of the existing local search methods cannot guarantee to find the exact solution. Moreover, they are designed for specific measures and cannot be generalized to other measures.

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

- It guarantees to find the exact top-k nodes.
- It is a general method that can be applied to a variety of random walk based proximity measures.
- It uses a simple local search strategy that needs neither preprocessing nor iterating over the entire graph.

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 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 [6]. Examples include truncated or discounted hitting time [17, 19, 18], penalized hitting probability [11, 21], and random walk with restart [1, 20, 18] and its variants such as effective importance (degree normalized random walk with restart) [3].

The basic approach for proximity query is to use the power iteration method [16]. An improved iteration method designed for random walk with restart decomposes the proximity into random walk probabilities of different length [9]. It tries to reduce the number of iterations by estimating the proximity based on the information collected so far. Another approach precomputes the information needed for proximity estimation during the query process [20, 8, 10]. However, this step is time consuming and becomes infeasible when the graph is large or constantly changing. Graph embedding method embeds nodes into geometric space so that node proximities can be preserved as much as possible [22]. 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 [21] and depth-first [14] search strategies simply extract a fixed number of nodes near the query node. An approximate local search algorithm is proposed for truncated hitting time [17]. The key idea is to develop upper and lower bounds that can be used to approximate the proximities of local nodes. Methods in [18, 3] apply a similar idea on the effective importance proximity measure. 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

Symbol	Definition
$G(V, E)$	undirected graph G with node set V and edge set E
N_i	neighbors of node i
w_{ij}	weight of edge (i, j)
w_i	weighted node degree $w_i = \sum_{j \in N_i} w_{ij}$
q	query node
k	number of returned nodes
S	a set of nodes
$\bar{S}$	complement of S: $\bar{S} = V \setminus S$
δS	boundary of S: $\{i \in S \vert \exists j \in N_i \cap \bar{S}\}$
$\delta \bar{S}$	boundary of $\bar{S}$: $\{i \in \bar{S} \vert \exists j \in N_i \cap S\}$
$\mathbf{r}$	$n \times 1$ vector, $\mathbf{r}_i$ is the proximity of node i w.r.t. query 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; $p_{i,j} = w_{ij}/w_i$
$\mathbf{P}$	transition probability matrix with $\mathbf{P}_{i,j} = p_{i,j}$
$\mathbf{T}$	modified $\mathbf{P}$: $\mathbf{T}_{i,j} = 0$, if $i = q$; $\mathbf{T}_{i,j} = \mathbf{P}_{i,j}$, if $i \neq q$
d, $\mathbf{r}_\mathrm{d}$	a dummy node d with constant proximity value $\mathbf{r}_\mathrm{d}$
c	decay factor (PHP or DHT); restart probability (RWR)

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.

3.1 Theoretical Basis

Note that for some measures, such as penalized hitting probability, random walk with restart and effective importance, the larger the proximity the closer the nodes. In this case, no local optimum means no local maximum. For other measures, such as discounted/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 1. [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 2. [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 $\delta S = \{i \in S \vert \exists j \in N_i \cap \bar{S}\}$ to denote the *boundary* of S, and $\delta \bar{S} = \{i \in \bar{S} \vert \exists j \in N_i \cap S\}$ to denote the boundary of $\bar{S}$.

Algorithm 1: Basic top-k local search

Input: $G(V, E)$, q, $\mathbf{r}$, k
Output: Top-k nodes set K

```
S ← {q}; δS̄ ← N_q;
while |S| < k + 1 do
    u = arg max_{i∈δS̄} r_i;
    S = S ∪ {u}; δS̄ = δS̄ \ {u} ∪ (N_u \ S);
return S \ {q};
```

Figure 1(a) 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. □

Theorem 1 can also be generalized to $\delta\bar{S}$.

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

PROOF. Let $S' = S \cup \delta\bar{S}$. We have $\delta S' \subseteq \delta\bar{S}$. Suppose that node $b \in \delta S'$ has the largest value in $\delta S'$. We have $\mathbf{r}_u \geq \mathbf{r}_b$. From Theorem 1, we have $\mathbf{r}_b > \mathbf{r}_i$ $(\forall i \in \bar{S}')$. Then, we have $\mathbf{r}_u \geq \mathbf{r}_j$ $(\forall j \in \bar{S} = \bar{S}' \cup \delta\bar{S})$. □

Based on Corollary 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 begins from the query node q and uses S to store the top-k nodes. In each iteration, the algorithm finds the node u that has the largest proximity in $\delta\bar{S}$ and put it in S. $\delta\bar{S}$ is then updated accordingly. The algorithm continues until $|S| = k + 1$.

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

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 (PHP) [11, 21] as an example to illustrate that it has no local maximum. We use w_i to denote the weighted degree of node i, and w_{ij} to denote the edge weight between i and j. The transition probability from i to j is thus $p_{i,j} = w_{ij}/w_i$.

Suppose the undirected graph in Figure 1(a) has unit edge weight. Node 3 has weighted 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 Figure 1(b). In the transition graph,

(a) Original graph (b) Transition graph (PHP, $q = 1$)

Figure 1: An example graph and its transition graph

Table 2: No local optimum property of some measures

Proximity measures	Abbr.	Ref.	Property
Penalized hitting probability	PHP	[11, 21]	No local maximum
Effective importance	EI	[3]	No local maximum
Discounted hitting time	DHT	[18]	No local minimum
Truncated hitting time	THT	[17]	No local minimum (within L hops)
Random walk with restart	RWR	[20]	Local maximum

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. It can be defined as the following recursive function:

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

where c $(0 < c < 1)$ is the decay factor in the random walk process. In [11], $c = e^{-1}$ is used as the decay factor. The query node q in PHP 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 Figure 1(b).

Let $\mathbf{P}$ be the transition probability matrix with $\mathbf{P}_{i,j} = p_{i,j}$, and $\mathbf{T}$ be the modified transition probability matrix with

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

Then the proximity $\mathbf{r}$ based on PHP can be written in the following matrix form

$$\mathbf{r} = c\mathbf{T}\mathbf{r} + \mathbf{e}_q,$$

where $\mathbf{e}_q(i) = 1$ if $i = q$, and $\mathbf{e}_q(i) = 0$ 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 of other proximity measures in Table 2 can be derived in a similar way. The detailed proofs can be found in the Appendix.

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.*

PROOF. The proof is in the Appendix. □

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

(a) Deletion and restoration (b) Destination change

Figure 2: Basic operations on transition probability

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 the 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, and their bounds are shown in the Appendix. How to apply the local search strategy to the measure having local optimum will be discussed in Section 5.6.

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. Figure 2(a) shows an example, in which the original graph is on the top, with node 1 being the query node and 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.

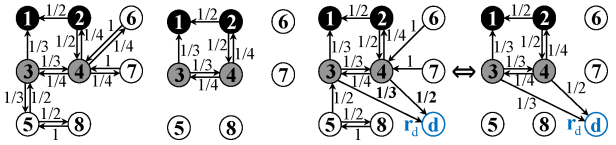
THEOREM 3. *Deleting a transition probability will not increase the proximity of any node.*

PROOF. Let $\mathbf{T}$ be the original transition probability matrix of PHP. Deleting $\mathbf{T}_{i,j}$ from $\mathbf{T}$ is the same as setting $\mathbf{T}_{i,j}$ to 0. Let $\mathbf{T}'$ represent the resulting matrix. PHP proximity $\mathbf{r}$ is computed based on $\mathbf{T}$, and $\mathbf{r}'$ is based on $\mathbf{T}'$.

Let $\Delta\mathbf{r} = \mathbf{r} - \mathbf{r}'$, and $\Delta\mathbf{T} = \mathbf{T} - \mathbf{T}'$. Note that $\Delta\mathbf{T}$ has only one non-zero element $\Delta\mathbf{T}_{i,j} = \mathbf{T}_{i,j}$. We have that $\Delta\mathbf{r} = c(\mathbf{T}' + \Delta\mathbf{T})\mathbf{r} - c\mathbf{T}'\mathbf{r}' = c\mathbf{T}'\Delta\mathbf{r} + c\Delta\mathbf{T}\mathbf{r} = c\mathbf{T}'\Delta\mathbf{r} + \mathbf{e}_i$, where $\mathbf{e}_i$ is a vector with the only non-zero element $\mathbf{e}_i(i) = c\mathbf{T}_{i,j}\mathbf{r}_j$. The solution of the previous equation is $\Delta\mathbf{r} = (\mathbf{I} - c\mathbf{T}')^{-1}\mathbf{e}_i$. The elements of $c\mathbf{T}'$ are non-negative and $||c\mathbf{T}'||_\infty < 1$. It can be expanded by Neumann series [15] as $(\mathbf{I} - c\mathbf{T}')^{-1} = \sum_{l=0}^{\infty}(c\mathbf{T}')^l > 0$. Thus the solution $\Delta\mathbf{r}$ must be non-negative. This completes the proof. □

Continue with the example in Figure 2(a). 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 Figure 2(a), the proximities will not decrease.



(a) Original $\mathbf{r}$ (b) LB $\underline{\mathbf{r}}$ (c) UB $\bar{\mathbf{r}}$

Figure 3: Lower and upper bounds based on basic operations

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

PROOF. Omitted. □

Figure 2(b) 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 l with $\mathbf{r}_l \geq \mathbf{r}_j$ ($\mathbf{r}_l \leq \mathbf{r}_j$) will not decrease (increase) the proximity of any node.*

PROOF. The proof is in the Appendix. □

Let us continue with the example in Figure 2(b), 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

Recall that in Algorithm 1, 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., $\underline{\mathbf{r}}_u \leq \mathbf{r}_u$. Therefore, $\underline{\mathbf{r}}$ can be used as the lower bound of $\mathbf{r}$.

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

Figure 3(b) shows the resulting 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 Figure 3(b) 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, $\bar{\mathbf{r}}_u$, will be greater than or equal to its original value $\mathbf{r}_u$, i.e., $\bar{\mathbf{r}}_u \geq \mathbf{r}_u$. Therefore, $\bar{\mathbf{r}}$ can be used as the upper bound of $\mathbf{r}$.

Continue with the example in Figure 3(a). 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 Figure 3(c) shows the resulting graph

Algorithm 2: Fast top-k local search (FLoS)

Input: $G(V, E)$, q, k, c, τ
Output: Top-k nodes set K

1: $S^0 \leftarrow \{q\}$; $\delta S^0 \leftarrow \{q\}$; $\underline{\mathbf{r}}_q^0 = 1$; $\overline{\mathbf{r}}_q^0 = 1$; $\mathbf{T}_{q,q}^0 = 0$;
2: bStop = false; $t = 0$;
3: **while** bStop == false **do**
4: $\quad t$++;
5: $\quad$ LocalExpansion();
6: $\quad$ UpdateLowerBound();
7: $\quad$ UpdateUpperBound();
8: $\quad$ CheckTerminationCriteria();
9: **return** K;

Algorithm 3: LocalExpansion()

1: $u = \arg\max_{i \in \delta S^{t-1}} \frac{1}{2}(\underline{\mathbf{r}}_i^{t-1} + \overline{\mathbf{r}}_i^{t-1})$;
2: $S^t = S^{t-1} \cup N_u$;
3: Update δS^t;

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 Figure 3(c).

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 $\mathbf{r}_\mathrm{d}$.

5. FAST LOCAL SEARCH

In this section, we present the FLoS algorithm, which utilizes the bounds developed in Section 4 to enable local search. We show that the bounds can only change monotonically when more nodes are visited. A theoretical analysis on the number of visited nodes is also provided.

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 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 modified transition matrix $\mathbf{T}$. Note that the size of $\mathbf{T}$ is $|S| \times |S|$ instead of $|V| \times |V|$. We do not allocate memory for the full matrix $\mathbf{T}$, and only use adjacent 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 (as shown

Algorithm 4: UpdateLowerBound()

1: $\mathbf{T}_{i,j}^t = w_{ij} / \sum_{l \in N_i} w_{il}$, if node i or j are newly added;
2: $\mathbf{T}_{i,j}^t = \mathbf{T}_{i,j}^{t-1}$, if nodes i and j exist in the last iteration;
3: $\underline{\mathbf{r}}_i^t = 0$, if node i is newly added;
4: $\underline{\mathbf{r}}_i^t = \underline{\mathbf{r}}_i^{t-1}$, if node i exists in the last iteration;
5: $\mathbf{e}(i) = 1$, if $i = q$; $\mathbf{e}(i) = 0$, otherwise;
6: $\underline{\mathbf{r}}^t$ = IterativeMethod($\mathbf{T}^t$, $\underline{\mathbf{r}}^t$, $\mathbf{e}$, c, τ);

Algorithm 5: UpdateUpperBound()

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

Algorithm 6: CheckTerminationCriteria()

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

Algorithm 7: IterativeMethod()

Input: $\mathbf{T}$, $\mathbf{r}_{in}$, $\mathbf{e}$, c, τ
Output: $\mathbf{r}_{out}$

1: $\mathbf{r}^0 = \mathbf{r}_{in}$; $n = 0$;
2: **repeat** n++; $\mathbf{r}^n = c\mathbf{T}\mathbf{r}^{n-1} + \mathbf{e}$; **until** $||\mathbf{r}^n - \mathbf{r}^{n-1}|| < \tau$;
3: **return** $\mathbf{r}^n$;

in Algorithm 7) to solve the linear equation $\underline{\mathbf{r}} = c\mathbf{T}\underline{\mathbf{r}} + \mathbf{e}$ to update $\underline{\mathbf{r}}$.

Algorithm 5 shows how to update the upper bound. The transition matrix $\mathbf{T}$ 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 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 $\mathbf{r}_\mathrm{d}^t = \max_{i \in \delta S^{t-1}} \overline{\mathbf{r}}_i^{t-1} \geq \max_{i \in \delta S^{t-1}} \mathbf{r}_i > \mathbf{r}_j (\forall j \in \bar{S}^{t-1})$, 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 $\overline{\mathbf{r}} = c\mathbf{T}\overline{\mathbf{r}} + \mathbf{e}$ to update $\overline{\mathbf{r}}$ 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 criteria. We select the k nodes in $S \setminus \delta S \setminus \{q\}$ with largest $\underline{\mathbf{r}}$ values. If the minimum lower bound of the selected nodes is greater than the maximum upper bound of the remaining visited nodes, the selected nodes will be the top-k nodes in the entire graph.

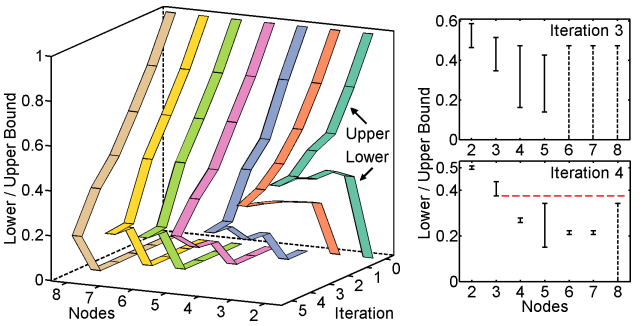


Figure 4: Lower and upper bounds in different iterations (PHP: $q = 1$, $c = 0.8$)

Table 3: Newly visited nodes in each iteration

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

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.

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

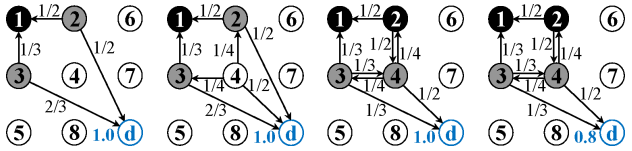
The left figure in Figure 4 shows how the lower and upper bounds change through local expansions. Query node 1 has constant proximity value 1.0 thus is not shown. It can be seen that the bounds monotonically change and eventually converge to the exact proximity value when all the nodes are visited. The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

The right figure in Figure 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 unvisited node is indicated by the dashed line segment. 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 Bounds

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. From iteration $(t-1)$ to t, we only restore some transition probabilities in $\{p_{i,j} : i \text{ or } j \in S^t \setminus S^{t-1}, i \neq q\}$. From Theorem 4, we have $\underline{\mathbf{r}}_i^t \geq \underline{\mathbf{r}}_i^{t-1} (\forall i \in S^{t-1})$. Therefore, the lower bound will monotonically non-decrease when more nodes are visited.

Next we examine the monotonicity of the upper bound. From iteration $(t-1)$ to t, we add new nodes in $S^t \setminus S^{t-1}$ and decrease $\mathbf{r}_\mathrm{d}$. After adding new nodes, the transition probabilities need to be updated accordingly. Specifically, we need to (1) add transition probabilities $\{p_{i,j}\}$ from the newly added nodes $i (\in S^t \setminus S^{t-1})$ to nodes $j (\in \delta S^{t-1})$, and $\{p_{i,\mathrm{d}}\}$ from i to the dummy node d; (2) 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 $\{p_{j,\mathrm{d}}\}$.



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

Figure 5: Transition graphs between two adjacent iterations

An example is shown in Figure 5. Figure 5(a) shows the transition graph for the first iteration. Figure 5(b) shows the resulting graph after applying step 1. Figure 5(c) shows the resulting graph after applying step 2. Reducing $\mathbf{r}_\mathrm{d}$ in Figure 5(c) will result in the final transition graph for the next iteration as shown in Figure 5(d).

Note that applying step 1 will not change the upper bound values for the nodes in S^{t-1}, since all the newly added transition probabilities begin from nodes $i (\in S^t \setminus S^{t-1})$. Step 2 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 $\mathbf{r}_\mathrm{d} \geq \mathbf{r}_\mathrm{i}$. Thus, step 2 will not increase the upper bound values according to Theorem 5. From Theorem 5, reducing the value of $\mathbf{r}_\mathrm{d}$ will not increase the upper bound values. Therefore, the upper bound will be monotonically non-increase from iteration $(t-1)$ to t.

5.3 Tightening the Bounds

The bounds used in FLoS can be further tightened by adding self-loop transition probabilities to the nodes in δS. We will still use PHP as an example to illustrate the process. We first define the star-to-mesh transformation on the transition graph, which is inspired by the star-mesh transformation in circuit theory [12].

DEFINITION 3. [Star-to-mesh transformation] *(1) Delete a node $u \in V \setminus \{q\}$ and its incident transition probabilities; (2) For any pair of nodes $i, j \in N_u$, add transition probabilities $p'_{i,j} = c p_{i,u} p_{u,j}$.*

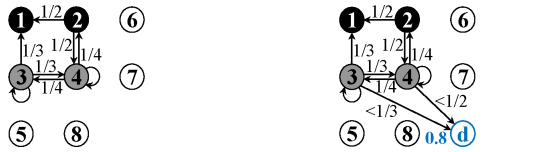
Note that if $i = j$, it becomes the self-loop transition probability $p'_{i,i} = c p_{i,u} p_{u,i}$. Applying the star-to-mesh transformation for a node will not change the PHP proximity values of the remaining nodes.

LEMMA 2. *Applying the star-to-mesh transformation of node u will not change the PHP proximity values of nodes $V \setminus \{q, u\}$.*

PROOF. This can be proved by plugging the equation $\mathbf{r}_u = c \sum_{i \in N_u} p_{u,i} \mathbf{r}_i$ into the recursive equations of neighbor nodes $i \in N_u$ in the original transition graph. □

The self-loop transition probabilities generated in the star-to-mesh transformation can be used to further tighten the lower and upper bounds.

LEMMA 3. *Adding self-loop transition probability $p_{i,i} = c \sum_{j \in N_i \cap \delta \bar{S}} p_{i,j} p_{j,i}$ $(\forall i \in \delta S)$ will tighten the lower bound.*



(a) Adding self-loop for LB $\underline{\mathbf{r}}$ (b) Adding self-loop for UB $\overline{\mathbf{r}}$

Figure 6: Transition graphs with self-loops

PROOF. Adding a self-loop transition probability is equivalent to changing the destination of the transition probability $p_{i,i}$ from a dummy node with proximity value 0 to node i. Therefore, the lower bound values of all the nodes will be non-decreasing.

Next we show the new bound values are still lower bounds. For the nodes in $\delta\bar{S}$, we apply star-to-mesh transformation sequentially in any order. After the star-to-mesh transformation of one node $j \in \delta\bar{S}$, we delete all the newly added transition probabilities except the self-loop transition probabilities of nodes in δS. After all the nodes in $\delta\bar{S}$ have been deleted, the self-loop transition probability of a node $i \in \delta S$ is $p_{i,i} = c\sum_{j\in N_i\cap\delta\bar{S}} p_{i,j}p_{j,i}$. During this process, we only apply star-to-mesh transformation and transition probability deletion. Therefore, the new bound values are still lower bounds. □

LEMMA 4. *Adding self-loop transition probability* $p_{i,i} = c\sum_{j\in N_i\cap\delta\bar{S}} p_{i,j}p_{j,i}$ *and set* $p_{i,\mathrm{d}} = c\sum_{j\in N_i\cap\delta\bar{S}} p_{i,j}(1-p_{j,i})$ *($\forall i \in \delta S$) will tighten the upper bound.*

PROOF. Note that the sum is $p_{i,i}+p_{i,\mathrm{d}} = c\sum_{j\in N_i\cap\delta\bar{S}} p_{i,j} < \sum_{j\in N_i\cap\delta\bar{S}} p_{i,j}$, which is the transition probability to dummy node d in the original transition graph. This means that we change the destination of transition probability $p_{i,i}$ from node d to node i. Since the proximity of d is no less than that of i, the upper bound of all nodes will be non-increasing. We can prove that the proximity values are still upper bounds through the star-to-mesh transformation. □

Figure 6 shows the transition graphs with self-loop probabilities for computing the lower and upper bounds. The original ones are shown in Figure 3.

5.4 Analysis on the Number of Visited Nodes

In this subsection, we analyze the number of visited nodes. Let h be the average number of neighbors of a node. Suppose that the nodes in the boundary of visited nodes are ρ hops away from the query node q. We assume that the number of visited nodes equals h^ρ. Note that this is an upper bound of the number of visited nodes. In Section 6, we show the actual number of visited nodes in real graphs.

The proximity of one node will decrease by a factor of c when it is one hop farther away from the query node. The nodes in the boundary of the visited nodes have proximity less than c^ρ because they are ρ hops away from the query.

What is the distribution of the gaps between the upper and lower bounds? The upper bound $\overline{\mathbf{r}}$ is computed based on the recursive equation $\overline{\mathbf{r}} = c\mathbf{T}\overline{\mathbf{r}} + \mathbf{e}_{q,\mathrm{d}}$, where vector $\mathbf{e}_{q,\mathrm{d}}$ has only two non-zero elements $\mathbf{e}_{q,\mathrm{d}}(q) = 1$ and $\mathbf{e}_{q,\mathrm{d}}(\mathrm{d}) = \mathbf{r}_\mathrm{d}$. When we set the proximity value of dummy node to 0, i.e., $\mathbf{e}_{q,\mathrm{d}}(\mathrm{d}) = 0$, we will get the recursive equation $\underline{\mathbf{r}} = c\mathbf{T}\underline{\mathbf{r}} + \mathbf{e}_q$ for the lower bound $\underline{\mathbf{r}}$. Let $\mathbf{r}' = \overline{\mathbf{r}} - \underline{\mathbf{r}}$ be the gaps between the upper and lower bounds. We have that $\mathbf{r}' = c\mathbf{T}\mathbf{r}' + \mathbf{e}_\mathrm{d}$, where vector $\mathbf{e}_\mathrm{d}$ has only one non-zero element $\mathbf{e}_\mathrm{d}(\mathrm{d}) = \mathbf{r}_\mathrm{d}$. Thus the gaps $\mathbf{r}'$ can be interpreted as the PHP proximity values when the query node is the dummy node d with constant proximity value $\mathbf{r}_\mathrm{d}$.

Based on this observation, the gap $\mathbf{r}'_u$ of one node u will decrease by a factor of c when it is one hop farther away from the boundary. Let ρ_u denote the number of hops that u is away from the query node. Node u is $\rho - \rho_u$ hops away from the boundary. So, the gap $\mathbf{r}'_u$ is less than $c^{\rho-\rho_u}\cdot c^\rho$, i.e., $\mathbf{r}'_u < c^{2\rho-\rho_u}$.

For any two nodes u and v, we can distinguish their rankings if $\mathbf{r}'_u + \mathbf{r}'_v < \epsilon$, where ϵ is the difference of the exact proximity values of nodes u and v. We already derive that $\mathbf{r}'_u < c^{2\rho-\rho_u}$ and $\mathbf{r}'_v < c^{2\rho-\rho_v}$. Thus if we have that $\rho > \frac{1}{2}\log_c\epsilon - \frac{1}{2}\log_c(c^{-\rho_u} + c^{-\rho_v})$, then the pair of nodes u and v can be distinguished.

Now we consider the case when u and v are the kth and $(k+1)$th nodes in the exact ranking list respectively. We have $h^{\rho_u} \geq k$ and $h^{\rho_v} \geq k+1$. Thus, the number of visited hops should satisfy $\rho > \frac{1}{2}\log_c\frac{\epsilon}{2} + \frac{1}{2}\log_h k$. Therefore, we need to visit $h^\rho = O((kh^{\log_c\frac{\epsilon}{2}})^{\frac{1}{2}})$ nodes to distinguish the kth and $(k+1)$th nodes.

5.5 Complexity

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

Note that so far, we have used PHP to illustrate the key principles underlying the fast local search method. All results are applicable to other measures with no local optimum such as DHT, THT, and EI. The derivations are similar to those of PHP and omitted here.

5.6 Extension to RWR

In this section, we discuss how to apply local search to RWR which has local optimum. The key idea is to utilize the relationship between RWR and PHP. Utilizing such relationship, we can easily derive the lower and upper bounds for RWR based on the lower and upper bounds for PHP.

Random walk with restart (also known as personalized PageRank) [20] is a widely used proximity measure. RWR can be described as follows. From a node i, the random walker can walk to its neighbors with probabilities proportional to the edge weights. In each step, it has a probability of c to return to i, where c is a constant. The proximity of node i w.r.t. q is defined as the stationary probability that the walker will finally stay at q. RWR can be defined recursively as $\mathbf{r}_i = (1-c)\sum_{j\in N_i} p_{j,i}\mathbf{r}_j$, if $i \neq q$, and $\mathbf{r}_q = (1-c)\sum_{j\in N_q} p_{j,q}\mathbf{r}_j + c$, where c $(0 < c < 1)$ is the constant restart probability.

Let RWR(i) and PHP(i) represent the proximity of node i based on RWR and PHP respectively. We first show that RWR and PHP have the following relationship.

THEOREM 6. $\text{RWR}(i) = \frac{\text{RWR}(q)}{w_q} \cdot w_i \cdot \text{PHP}(i)$

PROOF. Based on the recursive definition of RWR, we have $\frac{\text{RWR}(i)}{w_i} = (1-c)\sum_{j\in N_i} \frac{w_{ij}}{w_j}.\frac{\text{RWR}(j)}{w_i} = (1-c)\sum_{j\in N_i} \frac{w_{ij}}{w_i}.\frac{\text{RWR}(j)}{w_j}$, for $\forall i \neq q$. So, we have $\frac{\text{RWR}(i)}{w_i} = (1-c)\sum_{j\in N_i} p_{i,j}.\frac{\text{RWR}(j)}{w_j}$, for $\forall i \neq q$. This degree normalized RWR, $\frac{\text{RWR}(i)}{w_i}$, has the same recursive equation as PHP with decay factor $(1-c)$. So we have $\frac{\text{RWR}(i)}{w_i\cdot\text{PHP}(i)} = \frac{\text{RWR}(q)}{w_q\cdot\text{PHP}(q)}$. □

Based on Theorem 6, we have that $\text{RWR}(i) \propto w_i \cdot \text{PHP}(i)$ when the query node q is fixed. Suppose node $b \in \delta S$ has the largest PHP proximity value. Based on Theorem 1, we have $\text{PHP}(i) \leq \text{PHP}(b), \forall i \in \bar{S}$. Let $w(\bar{S})$ denote the maximum degree of unvisited nodes in $\bar{S}$. We have $w_i \cdot \text{PHP}(i) \leq w(\bar{S}) \cdot \text{PHP}(i) \leq w(\bar{S}) \cdot \text{PHP}(b)$. Therefore, if we maintain the maximum degree of the unvisited nodes, we can develop upper bound for the proximity values of unvisited nodes.

Specifically, we can apply FLoS to RWR as follows. In Algorithm 3, change line 1 to $u = \arg\max_{i\in\delta S^{t-1}} \frac{w_i}{2}(\underline{\mathbf{r}}_i^{t-1} + \overline{\mathbf{r}}_i^{t-1})$. In Algorithm 6, we select k nodes in $S^t \backslash \delta S^t \backslash \{q\}$ with largest $w_i \cdot \underline{\mathbf{r}}_i^t$ values, and change the termination criteria to $(\min_{i\in K} w_i \cdot \underline{\mathbf{r}}_i^t \geq \max_{i\in S^t\backslash K\backslash\{q\}} w_i \cdot \overline{\mathbf{r}}_i^t)$ and $(\min_{i\in K} w_i \cdot \underline{\mathbf{r}}_i^t \geq w(\bar{S}) \cdot \max_{i\in\delta S^t} \overline{\mathbf{r}}_i^t)$. All other processes remain the same.

6. 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 [7] and R-MAT model [4] with different parameters. All programs are written in C++. All experiments are performed on a server with 32G memory, Intel Xeon 3.2GHz CPU, and Redhat 4.1.2 OS.

6.1 State-of-the-art Methods

The measures we use include PHP, EI, THT, and RWR. 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 [16]. It guarantees to find the exact top-k nodes. The graph embedding (GE) method can answer the query in constant time after embedding [22]. 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 [9]. K-dash is the state-of-the-art matrix-based method for RWR which guarantees result exactness [8]. Note that K-dash and GE can only be applied on two medium-sized real graphs because of the expensive preprocessing step.

Dynamic neighborhood expansion (DNE) method applies a best-first expansion strategy to find the top-k nodes using PHP [21]. 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, 5] in local search, and guarantees the exactness of the top-k results [3]. Since PHP and EI are equivalent in terms of ranking, we can compare the methods for PHP and EI directly. LS_RWR also applies the push style method [2, 5] in local search [18]. It returns approximate results. LS_EI is based on LS_RWR and has similar performance [18]. LS_THT is a local search method for THT [17]. In the experiments, we set the truncated length to 10. The decay factor in PHP and the restart probability in RWR and EI are set to 0.5.

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	[16]	Exact
	DNE	Local search	[21]	Approx.
	NN_EI	Local search	[3]	Exact
	LS_EI	Local search	[18]	Approx.
FLoS_RWR	GI_RWR	Global iteration	[16]	Exact
	GE_RWR	Graph embedding	[22]	Approx.
	Castanet	Improved GI	[9]	Exact
	K-dash	Matrix inversion	[8]	Exact
	LS_RWR	Local search	[18]	Approx.
FLoS_THT	GI_THT	Global iteration	[16]	Exact
	LS_THT	Local search	[17]	Approx.

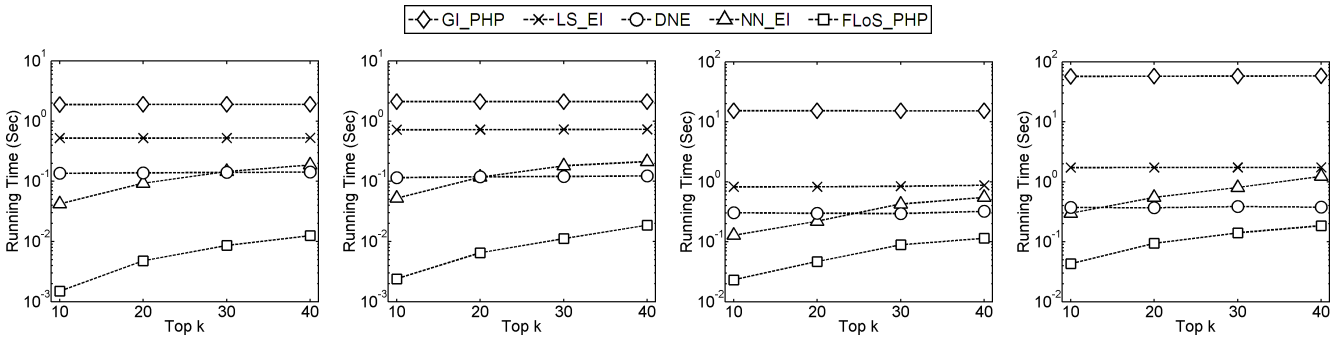
6.2 Evaluation on Real Graphs

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 due to the space limit.

6.2.1 Evaluation of FLoS_PHP

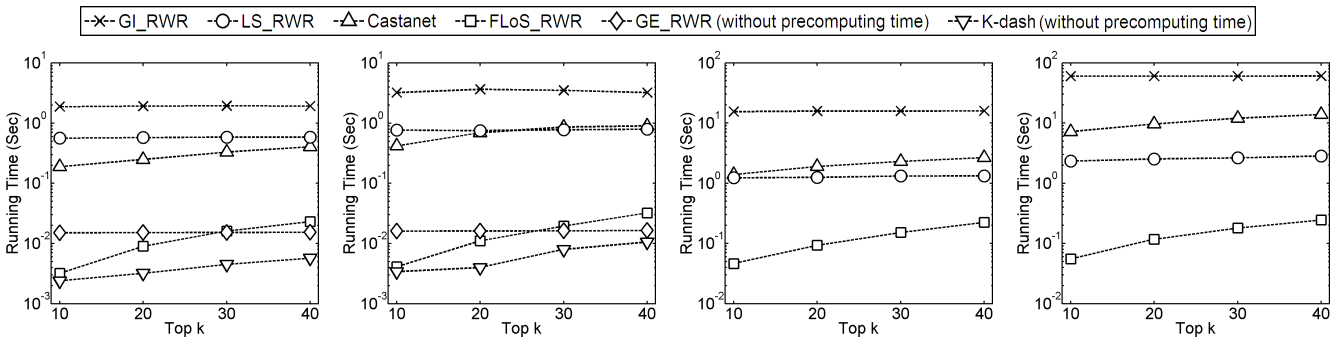
Figure 7 shows the running time of different methods for PHP. The running time of DNE is almost a constant for different k, because it visits a fixed number of nodes. The running time of NN_EI increases when k increases. FLoS_PHP is more efficient than NN_EI, which demonstrates that the bounds of FLoS 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.

Figure 9(a) 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.



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

Figure 7: Running time of different methods for PHP on real graphs



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

Figure 8: Running time of different methods for RWR on real graphs

6.2.2 *Evaluation of FLoS_RWR*

Figure 8 shows the running time for RWR. K-dash has the best performance after precomputing the matrix inversion as shown in Figures 8(a) and 8(b). 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%. LS_RWR method has constant running time, and it needs tens of hours in the precomputing step to cluster the graphs.

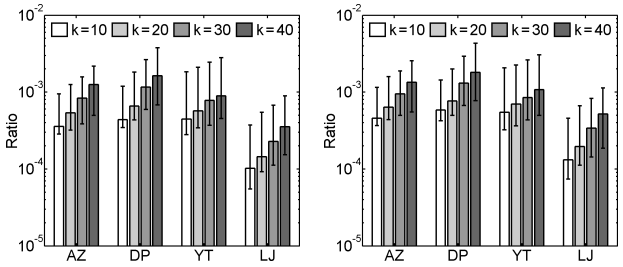
Figure 9(b) shows the ratio of the number of visited nodes of the FLoS_RWR method. The results are similar to that of Figure 9(a).

6.2.3 *Evaluation of FLoS_THT*

Figure 10 shows the running time for THT. 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.

6.3 Evaluation on In-Memory Synthetic Graphs

We use synthetic graphs with different parameters to evaluate the selected methods. Specifically, we study two types of graphs: Erdös-Rényi random graph (RAND) [7]



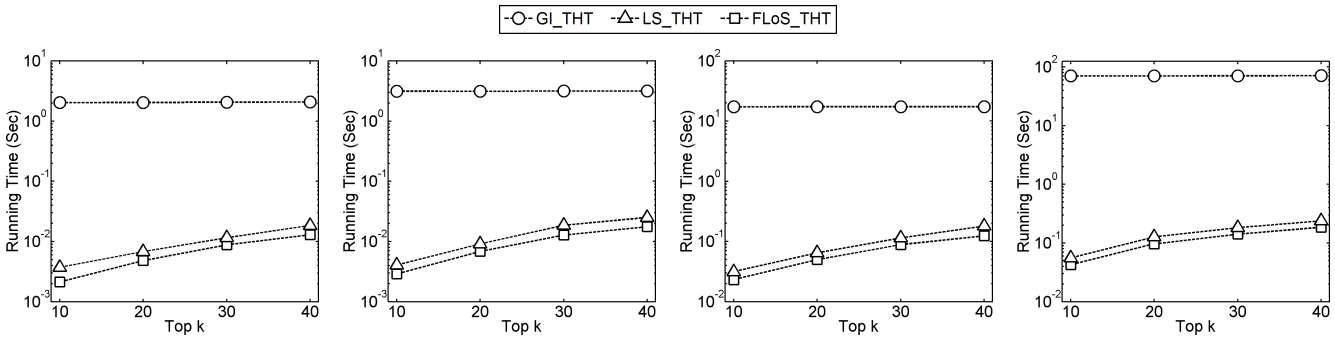
(a) FLoS_PHP (b) FLoS_RWR

Figure 9: Ratio between the number of visited nodes and the total number of nodes on real graphs

and scale-free graph based on R-MAT model [4]. 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 and RWR.

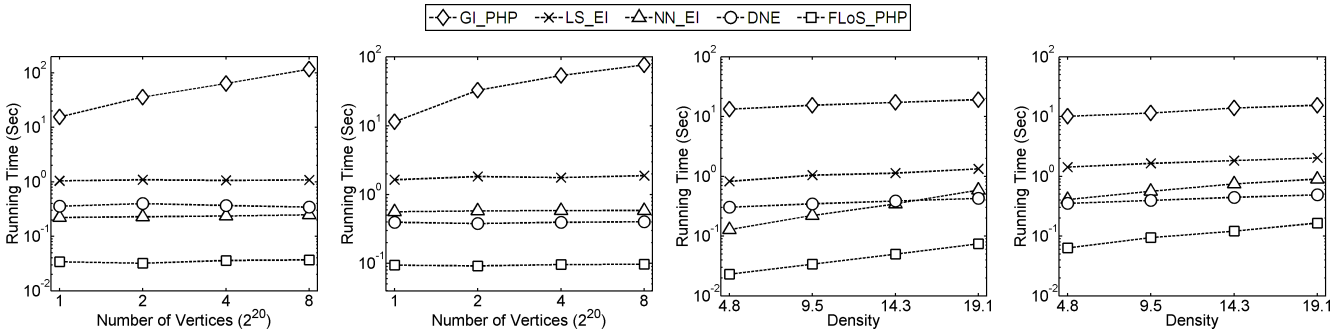
We download the graph generator available from the website *http://www.cse.psu.edu/∼madduri/software/GTgraph/* and use the default parameters to generate two series of graphs with varying size and varying density, using RAND and R-MAT respectively. The graphs with varying size have the same density but 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 and RWR 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.



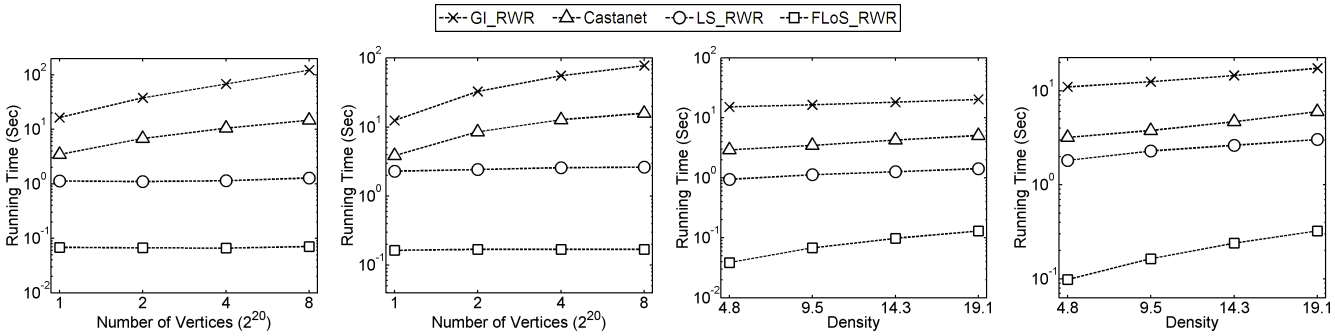
(a) AZ (b) DP (c) YT (d) LJ

Figure 10: Running time of different methods for THT on real graphs



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

Figure 11: 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

Figure 12: Running time of different methods for RWR on in-memory synthetic graphs ($k = 20$)

Table 6: Statistics of in-memory synthetic graphs

Varying size	$\|V\|$	1×2^{20}	2×2^{20}	4×2^{20}	8×2^{20}
	$\|E\|$	1×10^7	2×10^7	4×10^7	8×10^7
	Density	9.5	9.5	9.5	9.5
Varying density	$\|V\|$	1×2^{20}	1×2^{20}	1×2^{20}	1×2^{20}
	$\|E\|$	5×10^6	10×10^6	15×10^6	20×10^6
	Density	4.8	9.5	14.3	19.1

6.3.1 Evaluation of FLoS_PHP

Figure 11(a) 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. Figure 11(b) shows the running time on the series of R-MAT graphs with varying size. Similar trends are observed. Comparing Figure 11(a) and 11(b), 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.

Figure 11(c) 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. Figure 11(d) shows the running time on the series of R-MAT graphs with varying density. Similar trends are observed.

6.3.2 Evaluation of FLoS_RWR

Figure 12(a) 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%. 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. Figure 12(b) shows the running time on the series of R-MAT graphs with varying size. Similar trends are observed. Comparing Figure 12(a) and 12(b), 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.

Figure 12(c) 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. Figure 12(d) shows the running time on the series of R-MAT graphs with varying density. Similar trends are observed.

6.4 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 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.

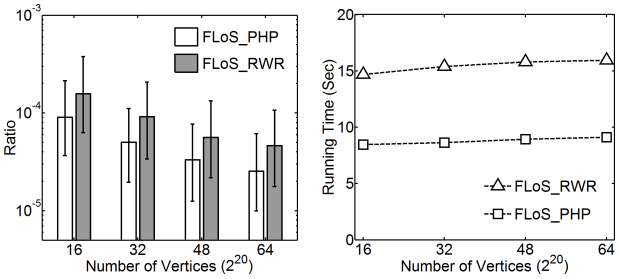
Figure 13(a) shows the running time of the FLoS_PHP and FLoS_RWR methods. From the figure, we can see that FLoS can process disk-resident graphs in tens of seconds. The reason is that FLoS only needs to find the neighbors of visited nodes and the transition probabilities on the edges. These results also verify that FLoS has almost constant running time when the number of nodes increases. Figure 13(b) 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.

7. 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.

Table 7: Statistics of disk-resident synthetic graphs

$\|V\|$	16×2^{20}	32×2^{20}	48×2^{20}	64×2^{20}
$\|E\|$	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

Figure 13: Results of FLoS_PHP and FLoS_RWR on disk-resident synthetic graphs ($k = 20$)

By exploiting the relationship among different proximity measures, we can also extend FLoS to the proximity measures having local optimum. Extensive experimental results demonstrate that FLoS enables efficient and exact query for a variety of random walk based proximity measures.

8. ACKNOWLEDGEMENTS

This work was partially supported by the National Science Foundation grants IIS-1162374, IIS-1218036, IIS-0953950, the NIH/NIGMS grant R01GM103309, and the OSC (Ohio Supercomputer Center) grant PGS0218. We would like to thank anonymous reviewers for their valuable comments.

9. REFERENCES

[1] D. Aldous and J. Fill. Reversible markov chains and random walks on graphs, 2002.

[2] P. Berkhin. Bookmark-coloring algorithm for personalized PageRank computing. *Internet Mathematics*, 3(1):41–62, 2006.

[3] P. Bogdanov and A. Singh. Accurate and scalable nearest neighbors in large networks based on effective importance. In *CIKM*, pages 523–528, 2013.

[4] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *SDM*, pages 442–446, 2004.

[5] S. Chakrabarti, A. Pathak, and M. Gupta. Index design and query processing for graph conductance search. *VLDB*, 20(3):445–470, 2011.

[6] S. Cohen, B. Kimelfeld, and G. Koutrika. A survey on proximity measures for social networks. In *Search Computing*, pages 191–206, 2012.

[7] P. Erdős and A. Rényi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 5:17–61, 1960.

[8] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa. Fast and exact top-k search for random walk with restart. *VLDB*, 5(5):442–453, 2012.

[9] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, T. Mishima, and M. Onizuka. Efficient ad-hoc search for personalized PageRank. In *SIGMOD*, pages 445–456, 2013.

[10] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka. Efficient personalized PageRank with accuracy assurance. In *KDD*, pages 15–23, 2012.

[11] Z. Guan, J. Wu, Q. Zhang, A. Singh, and X. Yan. Assessing and ranking structural correlations in graphs. In *SIGMOD*, pages 937–948, 2011.

[12] E. A. Guillemin. *Introductory circuit theory*. John Wiley & Sons, 1953.

[13] P. Lee, L. V. Lakshmanan, and J. X. Yu. On top-k structural similarity search. In *ICDE*, pages 774–785, 2012.

[14] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM*, pages 469–478, 2008.

[15] C. Meyer. *Matrix analysis and applied linear algebra*. SIAM, 2000.

[16] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[17] P. Sarkar and A. W. Moore. A tractable approach to finding closet truncated communicate time nieghbors in large graphs. In *UAI*, pages 335–343, 2007.

[18] P. Sarkar and A. W. Moore. Fast nearest-neighbor search in disk-resident graphs. In *KDD*, pages 513–522, 2010.

[19] P. Sarkar, A. W. Moore, and A. Prakash. Fast incremental proximity search in large graphs. In *ICML*, pages 896–903, 2008.

[20] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622, 2006.

[21] C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei. Evaluating geo-social influence in location-based social networks. In *CIKM*, pages 1442–1451, 2012.

[22] X. Zhao, A. Chang, A. D. Sarma, H. Zheng, and B. Y. Zhao. On the embeddability of random walk distances. *VLDB*, 6(14), 2013.

10. APPENDIX

10.1 Proofs of No Local Optimum Property for Other Proximity Measures

Effective importance (EI) [3] is the degree normalized version of RWR, which can be defined as $\mathbf{r}_i = (1-c)\sum_{j\in N_i} p_{i,j}\mathbf{r}_j$, if $i \neq q$, and $\mathbf{r}_q = (1-c)\sum_{j\in N_q} p_{q,j}\mathbf{r}_j + \frac{c}{w_q}$, where c $(0 < c < 1)$ is a constant restart probability.

LEMMA 5. *EI has no local maximum.*

PROOF. Suppose that node i is a local maximum. We have $\mathbf{r}_i = (1-c)\sum_{j\in N_i} p_{i,j}\mathbf{r}_j \leq (1-c)\sum_{j\in N_i} p_{i,j}\mathbf{r}_i = (1-c)\mathbf{r}_i < \mathbf{r}_i$. Thus we get a contradiction that $\mathbf{r}_i < \mathbf{r}_i$. □

Discounted hitting time (DHT) [18] is a variant of HT which can be defined as $\mathbf{r}_i = 1 + (1-c)\sum_{j\in N_i} p_{i,j}\mathbf{r}_j$, if $i \neq q$, and $\mathbf{r}_q = 0$, where c $(0 < c < 1)$ is a constant used to penalize the transition probability in each step.

LEMMA 6. *DHT has no local minimum.*

PROOF. The maximum discounted hitting time that a node could have is $\frac{1}{c}$, when it can never reach q. For connected graph, we have $\mathbf{r}_i < \frac{1}{c}$. Now suppose that node i is a local minimum. We have $\mathbf{r}_i = 1 + (1-c)\sum_{j\in N_i} p_{i,j}\mathbf{r}_j \geq 1 + (1-c)\sum_{j\in N_i} p_{i,j}\mathbf{r}_i = 1 + (1-c)\mathbf{r}_i$. Thus $\mathbf{r}_i \geq \frac{1}{c}$, which contradicts that $\mathbf{r}_i < \frac{1}{c}$. □

Truncated hitting time (THT) [17] is another variant of HT. The L-truncated hitting time only considers paths of length less than L, which can be defined as $\mathbf{r}_q = 0$ and $\mathbf{r}_i^L = 1 + \sum_{j\in N_i} p_{i,j}\mathbf{r}_j^{L-1}$, if $i \neq q$. If a node is more than L hops away from the query node, its proximity is set to be L.

LEMMA 7. *THT has no local minimum for the nodes within L hops from the query node.*

PROOF. For any node i within L hops from q, we have $\mathbf{r}_i^L < L$. Now suppose i is a local minimum, i.e., $\mathbf{r}_j^L \geq \mathbf{r}_i^L$ $(\forall j \in N_i)$. We have $\mathbf{r}_i^L = 1 + \sum_{j\in N_i} p_{i,j}\mathbf{r}_j^{L-1} = \sum_{j\in N_i} p_{i,j}(1 + \mathbf{r}_j^{L-1}) > \sum_{j\in N_i} p_{i,j}\mathbf{r}_j^L \geq \sum_{j\in N_i} p_{i,j}\mathbf{r}_i^L = \mathbf{r}_i^L$. We get a contradiction that $\mathbf{r}_i^L > \mathbf{r}_i^L$. □

LEMMA 8. *RWR has local maximum.*

PROOF. Counter examples can be used to show that RWR has local maximum, which are omitted. □

10.2 The Proof of Theorem 2

PROOF. First, we show that PHP and EI are equivalent. Suppose the decay factor in PHP is set to be $(1-c)$, and the restart probability in EI is c. Then PHP and EI have the same recursive definition for any node $i \neq q$, and we have $\frac{\text{EI}(i)}{\text{PHP}(i)} = \frac{\text{EI}(q)}{\text{PHP}(q)}$, which is a constant when q is fixed. Thus $\text{PHP}(i)$ is a linear function of $\text{EI}(i)$.

Next, we show that PHP and DHT are equivalent. Suppose the decay factors in PHP and DHT are set to be $(1-c)$ and c respectively. The relationship between PHP and DHT is $\text{PHP}(i) = 1 - c \cdot \text{DHT}(i)$, which can be proved by substituting it in the recursive definition of PHP. Thus $\text{PHP}(i)$ is a linear function of $\text{DHT}(i)$. □

10.3 The Proof of Theorem 5

PROOF. Let $\mathbf{T}$ be the original transition probability matrix of PHP. Changing the destination of transition probability $p_{i,j}$ from node j to node l is the same as adding $\mathbf{T}_{i,j}$ to $\mathbf{T}_{i,l}$ and setting $\mathbf{T}_{i,j}$ to 0. Let $\mathbf{T}'$ represent the resulting matrix. PHP proximity $\mathbf{r}$ is computed based on $\mathbf{T}$, and $\mathbf{r}'$ is based on $\mathbf{T}'$.

Let $\Delta\mathbf{r} = \mathbf{r}' - \mathbf{r}$, and $\Delta\mathbf{T} = \mathbf{T} - \mathbf{T}'$. Note that $\Delta\mathbf{T}$ has only two non-zero elements $\Delta\mathbf{T}_{i,j} = \mathbf{T}_{i,j}$ and $\Delta\mathbf{T}_{i,l} = -\mathbf{T}_{i,j}$. We have that $\Delta\mathbf{r} = c\mathbf{T}'\mathbf{r}' - c(\mathbf{T}' + \Delta\mathbf{T})\mathbf{r} = c\mathbf{T}'\Delta\mathbf{r} - c\Delta\mathbf{T}\mathbf{r} = c\mathbf{T}'\Delta\mathbf{r} + \mathbf{e}_i$, where $\mathbf{e}_i$ is a vector with the only non-zero element $\mathbf{e}_i(i) = c\mathbf{T}_{i,j}(\mathbf{r}_l - \mathbf{r}_j)$. If $\mathbf{r}_l \geq \mathbf{r}_j$, $\Delta\mathbf{r}$ must be non-negative. Otherwise, it is non-positive. This completes the proof. □

10.4 Lower and Upper Bounds of Other Proximity Measures

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 L, which is the largest possible proximity value of THT.

EI and DHT are equivalent with PHP thus there is no need to discuss the bounds.