Efficient Top-k Shortest-Path Distance Queries on Large Networks by Pruned Landmark Labeling

Takuya Akiba[†], **Takanori Hayashi**[†], **Nozomi Nori**[‡], **Yoichi Iwata**[†] and **Yuichi Yoshida**[§‖]
†The University of Tokyo, 113-0033, Tokyo, Japan ‡Kyoto University, 606-8501, Kyoto, Japan
§National Institute of Informatics, 101-8430, Tokyo, Japan
‖Preferred Infrastructure, Inc., 113-0033, Tokyo, Japan
{t.akiba,thayashi,y.iwata}@is.s.u-tokyo.ac.jp, nozomi@ml.ist.i.kyoto-u.ac.jp, yyoshida@nii.ac.jp

Abstract

We propose an indexing scheme for top-k shortest-path distance queries on graphs, which is useful in a wide range of important applications such as network-aware searches and link prediction. While many efficient methods for answering standard (top-1) distance queries have been developed, none of these methods are directly extensible to top-k distance queries. We develop a new framework for top-k distance queries based on *2-hop cover* and then present an efficient indexing algorithm based on the recently proposed *pruned landmark labeling* scheme. The scalability, efficiency and robustness of our method is demonstrated in extensive experimental results. Moreover, we demonstrate the usefulness of top-k distance queries by applying them to link prediction, the most fundamental graph problem in the AI and Web communities.

Introduction

The shortest-path distance between vertices in a network is a fundamental concept in graph theory and is widely applied in the AI and Web communities. For example, because the distances between vertices indicate the relevance among the vertices, they can identify other users or contents that best match a user's intent in socially-sensitive searches (Vieira et al. 2007; Yahia et al. 2008; Maniu and Cautis 2013). In context-aware searches, they are used to assign higher ranks to web pages more related to the currently visited web page (Ukkonen et al. 2008; Potamias et al. 2009).

However, there is a fundamental drawback of basing relevance on distance alone. Specifically, distances should be integers and the diameters of real-world networks are typically small (Watts and Strogatz 1998). Such small diameter greatly reduce the number of possible distances and preclude the full use of the underlying structure.

This problem is clearly depicted in Figure 1. In each graph in the figure, the distance between the pair of black vertices is four. Hence, based on distance alone, the black pairs in all three graphs have the same similarity. However, the pair in graph (c) seems more tightly connected than the pairs in graphs (a) and (b), since this pair is connected by a greater number of shortest paths.

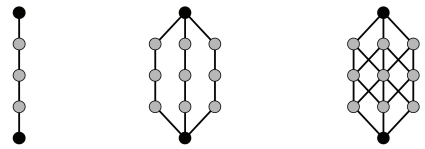


Figure 1: Examples of connection between two vertices.

Table 1: Distances and top-k distances between the two black vertices in Figure 1.

Graph	(Top-1) Distance	Top-k Distances
(a)	4	[4, 6, 6, 6, 6, 8, 8, ...]
(b)	4	[4, 4, 4, 6, 6, 6, 6, ...]
(c)	4	[4, 4, 4, 4, 4, 4, 4, ...]

This intuitive concept can be naturally implemented by adopting the *top-k shortest paths* and *top-k distances* (formally defined later). Table 1 presents the top-k distances between the pair of black vertices in each graph of Figure 1. Although the pairs in each graph are separated by the same distance, their top-k distances markedly vary, providing a potential means of distinguishing these three graph structures.

However, determining the top-k distances between vertices is computationally expensive. The naive approach is to apply a variant of Dijkstra's algorithm that visits the same vertex k times. This approach consumes $O((n+m)k)$ and $O((n \log n + m)k)$ time on unweighted and weighted graphs, respectively, where n and m are the numbers of vertices and edges, respectively. In the above-mentioned applications, the top-k distances must be interactively computed for many vertex pairs on large social and web graphs, requiring a much faster algorithm. Eppstein (Eppstein 1998) improved the time complexity to $O(n+m+k)$ and $O(n \log n + m+k)$ on unweighted and weighted graphs respectively, but his algorithm remains prohibitively slow.

Contribution

To resolve this issue, we propose an indexing method for answering the top-k distances. The proposed method is an indexing method, i.e., it first constructs a data structure called an *index* from a graph and then top-k distances between arbitrary pairs of vertices are rapidly obtained using the index. To our knowledge, we present the first indexing method to top-k distance inquiry.

Our method is built on the recently proposed *pruned landmark labeling*, an indexing scheme that answers shortest-path distances (Akiba, Iwata, and Yoshida 2013). However, modifying this method to answer top-k distances is nontrivial because the number of paths becomes crucial, bringing the new challenge of carefully avoiding double counts. Moreover, it requires several interesting ideas in order to keep the scalability.

As shown later in our experiments, our method can construct indices from large graphs comprising millions of vertices and tens of millions of edges within a reasonable running time. Having obtained the indices, we can compute the top-k distances within a few microseconds, six orders of magnitude faster than existing methods, which require a few seconds to compute these distances.

Moreover, to illustrate the importance of the top-k distances, we apply our method to the *link prediction problem* (Liben-Nowell and Kleinberg 2003), a well-studied problem in AI and Web communities. We empirically show that the support vector machine (SVM) with the top-k distances as its feature outperforms a number of baseline methods including singular value decomposition and random walk with restart. We emphasize that our indexing method enables the first use of the top-k distances for such tasks. The results also indicate the feasibility of top-k distances in other tasks, such as network-aware searching.

Our implementation of the proposed indexing method is publicly available from the first author's web page. We hope that our public code will enable further exploration of top-k distances in various applications.

Related Work

Distance Indices. Although numerous indexing methods for computing shortest-path distances have been proposed (Cheng and Yu 2009; Xiao et al. 2009; Wei 2010; Akiba, Sommer, and Kawarabayashi 2012; Jin et al. 2012; Fu et al. 2013; Akiba, Iwata, and Yoshida 2013), none of these methods can directly answer top-k distance queries.

Pruned Labeling Algorithms. Pruned labeling was first proposed for distance queries on complex networks (Akiba, Iwata, and Yoshida 2013). Then, specialization and extensions have been proposed for reachability queries on directed acyclic graphs (Yano et al. 2013), distance queries on road networks (Akiba et al. 2014), and distance queries on dynamic graphs (Akiba, Iwata, and Yoshida 2014).

Other Vertex Similarities. The performance of applications related to graph mining can also be enhanced by features other than top-k distances, such as *random walk with restart (RWR)*. As straightforward iterative algorithms are precluded by their high computational cost, several approximation methods have been proposed (Jeh and Widom 2003; McSherry 2005; Sun et al. 2005; Tong, Faloutsos, and Pan 2006). Despite sacrificing accuracy for efficiency, these methods remain prohibitively time-expensive for computing the RWR scores for many vertex pairs on large networks in real-time applications, such as network-aware searches.

Similar arguments apply to other walk-based similarities such as *SimRank* (Jeh and Widom 2002) and *commuting time* (Lovász 1996). In contrast, as we shall experimentally demonstrate, such large networks are efficiently handled by our method for answering top-k distances. We also believe that top-k distances provide features with different properties from them, which can be used as complementary features for those other vertex similarities.

Some of *graph kernels* (Smola and Kondor 2003), in particular, those based on the *graph Laplacian* such as the *regularized Laplacian kernel* can also be used to assign relevance scores for vertex pairs (Ito et al. 2005). However, the computational cost of graph kernels is even more infeasible for large graphs.

Preliminaries

The current study focus on networks that are modeled as graphs. To simplify our discussion, we consider only undirected and unweighted graphs first. However, as discussed later, our method is easily extendible to directed and weighted graphs.

Let $G = (V, E)$ be a graph with a vertex set V and an edge set E. We denote the number of vertices $|V|$ and the number of edges $|E|$ by n and m, respectively. We assume that vertices are uniquely represented by integers, enabling natural comparisons of two vertices $u, v \in V$ by expressions such as $u < v$ or $u \leq v$.

An *internal vertex* of a path refers to a vertex in the path that is not an endpoint of it. Let $\mathcal{P}$ be a set of paths. The *i-th shortest path in $\mathcal{P}$* refers to the i-th path in P, ordered by length, where ties are broken arbitrarily.

For a pair of vertices (s, t), let $\mathcal{P}_{st}$ be the set of all (unnecessarily simple) paths between s and t. For a vertex v, let $\mathcal{P}_{st}^{>v}$ be the set of paths in $\mathcal{P}_{st}$ whose internal vertices are all larger than v. Similarly, let $\mathcal{P}_{st}^{\not>v}$ be the set of paths in $\mathcal{P}_{st}$ such that at least one internal vertex is smaller than or equal to v. Then for two vertices s and t, the *i-th shortest path between s and t* is the i-th shortest path in $\mathcal{P}_{st}$. Let $d_{i\text{-th}}(s, t)$, $d_{i\text{-th}}^{>v}(s, t)$, and $d_{i\text{-th}}^{\not>v}(s, t)$ denote the length of the i-th shortest path in $\mathcal{P}_{st}$, $\mathcal{P}_{st}^{>v}$, and $\mathcal{P}_{st}^{\not>v}$, respectively. If the size of the corresponding set is less than i, then we set them to ∞. We define $d_{i\text{-th}}^{\geq v}(s, t)$ and $d_{i\text{-th}}^{\not\geq v}(s, t)$ similarly.

Problem Definition

In this paper, we propose an indexing method that, given a graph G and a positive integer k, construct an index to quickly answer the following query.

Problem 1 (Top-k Distance Query).
***Given**: A pair of vertices (s, t).*
***Answer**: An array $(d_{1\text{st}}(s, t), d_{2\text{nd}}(s, t), \ldots, d_{k\text{-th}}(s, t))$.*

Proposed Method

This section describes our proposed method and show its correctness. We also suggest several important techniques for practical performance enhancement.

Data Structure

The data structure and query algorithm of the proposed method are based on the general framework of *2-hop cover* (Cohen et al. 2002), which is designed for standard (top-1) distance queries. However, as normal distance queries do not consider the number of paths, the main challenge in processing top-k distance queries is preventing multiple counts of the same path. To this end, we require a more involved framework.

For each vertex v, our method precomputes and stores the following two labels:

- *Distance label* $L(v)$, comprising a set of pairs (u, δ) of a vertex and a path length. If we gather lengths in $L(v)$ associated with a vertex u, they should form the sequence $(d_{\text{1st}}^{>v}(v,u), d_{\text{2nd}}^{>v}(v,u), \ldots, d_{\ell\text{-th}}^{>v}(v,u))$ for some $1 \leq \ell \leq k$.
- *Loop label* $C(v)$, constituting a sequence of k integers $(\delta_1, \delta_2, \ldots, \delta_k)$. This sequence should equal $(d_{\text{1st}}^{\geq v}(v,v), d_{\text{2nd}}^{\geq v}(v,v), \ldots, d_{k\text{-th}}^{\geq v}(v,v))$.

An *index* is a pair $I = (L, C)$, where L and C are the sets of distance labels $\{L(v)\}_{v \in V}$ and loop labels $\{C(v)\}_{v \in V}$, respectively.

Query Algorithm

Given an index $I = (L, C)$ and a pair of vertices (s, t), we compute the top-k distances between s and t as follows. First, we compute the following multiset.

$$\Delta(I, s, t) = \{\delta_{sv} + \delta_{vv} + \delta_{vt} \mid (v, \delta_{sv}) \in L(s), \\ \delta_{vv} \in C(v), (v, \delta_{vt}) \in L(t)\}.$$

Intuitively, we first move from s to v, then loop back to v several steps later, and finally move from v to t. Note that from the definition of distance labels and loop labels, every internal vertex in the path from s to t (except v itself) is larger than v.

Let $\text{QUERY}(I, s, t)$ denote the smallest k elements in the multiset $\Delta(I, s, t)$. If $|\delta(I, s, t)| < k$, the remaining entries are filled with ∞. Our answer to the query (s, t) is $\text{QUERY}(I, s, t)$.

Indexing Algorithm

Our index constructing algorithm is summarized in Algorithm 1. We first compute the loop label $C(v)$ for every vertex v. We then construct the distance labels L by conducting a *pruned BFS* from each vertex.

Algorithm for Computing Loop Labels. We construct the loop labels as follows. For each vertex v, using vertices larger than or equal to v, we perform a modified version of breadth first search (BFS). In the BFS, each vertex may be visited up to k times. The first k visits to the vertex v gives the distance sequence $d_{\text{1st}}^{\geq v}(v,v), d_{\text{2nd}}^{\geq v}(v,v), \ldots, d_{k\text{-th}}^{\geq v}(v,v)$.

Algorithm 1 Indexing Algorithm

```
1: procedure ConstructIndex(G)
2:     for i = 1 to n do Compute C(v_i) using the modified BFS.
3:     L(v) ← ∅ for all v ∈ V.
4:     for i = 1 to n do PrunedBFS(G, v_i).
5:     return (C, L).
```

Algorithm 2 Pruned Top-k BFS from $v \in V$.

```
1: procedure PrunedBFS(G, v)
2:     Q ← a queue with only one element (v, 0).
3:     while Q is not empty do
4:         Dequeue (u, δ) from Q.
5:         if δ < max(Query((L, C), v, u)) then
6:             Add (v, δ) to L(u).
7:             for all w ∈ V such that (u, w) ∈ E, w > v do
8:                 Enqueue (w, δ + 1) onto Q.
```

The modified BFS returns to the starting vertex long before all vertices in the graph have been visited. Consequently, the running time is very small in practice and empirically estimated as $O(nk)$ in total from experiments.

Algorithm for Computing Distance Labels. We assume that vertices in V are ordered as $v_1, v_2, \ldots, v_n$. Then for each $1 \leq i \leq n$, we perform a *pruned BFS* from v_i (Algorithm 2). The pruned BFS is essentially a modified version of the BFS from v that visits the same vertex at most k times. The crucial difference is the non-trivial pruning; that is, when visiting a vertex u at distance δ, the process is discontinued if δ is larger than or equal to the k-th shortest distance computable by the current index (L, C) (Line 5).

We roughly estimate the time complexity. Let l be the average size of labels. We visit $O(nl)$ vertices in total, traversing $O(\frac{m}{n})$ edges on average and evaluating a query in $O(l)$ time (by using the fast pruning technique introduced later). Thus, the total time complexity of this part is $O(ml + nl^2)$. In our experiments, l was a few hundred.

Proof of Correctness

The correctness of our method is shown as follows. Let L_i denote the set of distance labels L after the i-th pruned BFS from v_i. We define $L_0(v) = \emptyset$ for any v. Let I_i denote pair (L_i, C) of the partially constructed set of distance labels and the set of loop labels. We prove the following lemma.

Lemma 1. *For every integer i where $0 \leq i \leq n$, and every pair of vertices (s, t), $\text{QUERY}(I_i, s, t) = (d_{\text{1st}}^{\not> v_i}(s,t), d_{\text{2nd}}^{\not> v_i}(s,t), \ldots, d_{k\text{-th}}^{\not> v_i}(s,t))$ holds.*

Proof. We prove the claim by induction on i. When $i = 0$, we have $\text{QUERY}(I_i, s, t) = (\infty, \infty, \ldots, \infty)$ and the claim clearly holds. Suppose that the claim holds for every $i' < i$. For a fixed pair of vertices (s, t) where $s \neq t$, we validate the claim for i and the pair (s, t).

Note that we can already compute $\text{QUERY}(I_{i-1}, s, t) = (d_{\text{1st}}^{\not> v_{i-1}}(s,t), d_{\text{2nd}}^{\not> v_{i-1}}(s,t), \ldots, d_{k\text{-th}}^{\not> v_{i-1}}(s,t))$. Let $\mathcal{P}$ denote the set of paths P such that (i) P is in $\mathcal{P}_{st}^{>v_{i-1}}$, (ii) P passes through v_i, and (iii) the length of P is smaller than

$d_{k\text{-th}}^{\not\succ v_{i-1}}(s,t)$. Let $\mathcal{P}'$ be the first k elements in $\mathcal{P}$. It suffices to show that, after the i-th pruned BFS, we can also compute the distances of paths in $\mathcal{P}'$.

Let $P \in \mathcal{P}'$. We can split P into three parts P_{sv_i}, $P_{v_iv_i}$, and P_{v_it}. Here, P_{sv_i} denotes the subsequence of P from s to the first appearance of v_i in P, $P_{v_iv_i}$ denotes the subsequence of P from the first appearance of v_i to the final appearance of v_i in P, and P_{v_it} denotes the subsequence of P from the last appearance of v_i in P to t. Note that $P_{v_iv_i}$ must be among the first k elements in $\mathcal{P}_{v_iv_i}^{>v_i}$; otherwise shorter k paths are possible and $P \in \mathcal{P}'$ is contradicted. Hence, $C(v_i)$ must include the length of $P_{v_iv_i}$.

Now we observe that the BFS from v_i along path P_{v_it} is not pruned in the i-th pruned BFS (and similarly for P_{sv_i}). To illustrate by contradiction, suppose that the BFS is pruned at some vertex u on path P_{v_it}. In this case, there exist at least k paths in $\mathcal{P}_{v_iu}^{\not\succ v_{i-1}}$ shorter than δ, where δ is the distance from v_i to u in the BFS. For each of these k paths, we concatenate P_{sv_i}, $P_{v_iv_i}$, and the suffix of P_{v_it} from u to t. Then, we obtain k paths in $\mathcal{P}_{st}^{\not\succ v_{i-1}}$ that are shorter than P, and therefore shorter than $d_{k\text{-th}}^{\not\succ v_{i-1}}(s,t)$ from condition (iii). Hence, we reach a contradiction. □

Corollary 1. *At the end of Algorithm 1, we can correctly answer top-k distance queries using the constructed index.*

Techniques for Efficient Implementation

We introduce several key techniques for practical performance improvement.

Vertex Ordering Strategy. By properly selecting the order of vertices from which we conduct pruned BFSs, our pruning can drastically reduce the search space and label sizes by exploiting the structure of real-world networks, greatly enhancing the efficiency of the proposed method. This is possible because the real networks contain highly centralized vertices (sometimes called *hubs*). As a heuristic vertex ordering strategy, vertices are selected in order of decreasing degrees. Further discussion is provided in (Akiba, Iwata, and Yoshida 2013).

Fast Pruning. When constructing distance labels, many queries are evaluated for pruning. However, when conducting a pruned BFS from a vertex v, queries are limited to "Are there more than k paths of length less than δ between v and u?" Given this restriction, we can reduce the query time. For each vertex w in the distance label of v, we can precompute the number $c_{w,\delta'}$ of paths between v and w of length not exceeding δ' using the loop label $C(w)$. Suppose that we have reached vertex u in the pruned BFS conducted from v. We can then compute the number of paths between v and u of length less than δ as $\sum_{(w,\delta',c)\in L(u)} c \cdot c_{w,\delta-\delta'}$.

Merged Queue Entries. When a (pruned) BFS is performed from a vertex v, rather than pair (u,δ), which denotes the existence of a path of length δ between v and u, triplets (u,δ,c) are pushed onto the queue. These triples specify that c paths of length δ exist between v and u, This technique enables the simultaneous handling of many paths, and significantly reduces the number of pushes onto the queue. Hence, it significantly reduces the running time.

Merged Label Entries. Related to the above technique, instead of pairs (u,δ), which denotes that there is a path of length δ between v and u, triplets (u,δ,c) are stored in distance labels. These triplets indicate that c paths of length δ exist between v and u. A similar technique is applicable to loop labels.

Extensions

Directed graphs. If the input graph is a directed graph, we compute and store two distance labels $L_{\text{IN}}(v)$ and $L_{\text{OUT}}(v)$ for each vertex v, where $L_{\text{IN}}(v)$ and $L_{\text{OUT}}(v)$ contain the distances from and to v, respectively.

Weighted graphs. For weighted graphs, we can replace the pruned BFS by pruned Dijkstra's algorithm. In this scheme, the queue used in Algorithm 2 is replaced by a priority queue. The time complexity becomes $O(ml + nl(\log n + l))$.

Experimental Evaluation

In this section, we show the scalability, efficiency and robustness of the proposed method by experimental results using real-world networks.

Setup

Environment. All experiments were conducted on a Linux server with Intel Xeon X5670 (2.93 GHz) and 48 GB of main memory. The proposed method was implemented in C++. The implementation will be made publicly available online.

Datasets. The target applications of the proposed method are graph mining tasks such as network-aware searching and link prediction. Therefore, our experiments were conducted on publicly available real-world social and web graphs[1][2][3][4][5]. The sizes and types of these graphs are listed in Table 2. We treated all the graphs as unweighted undirected graphs.

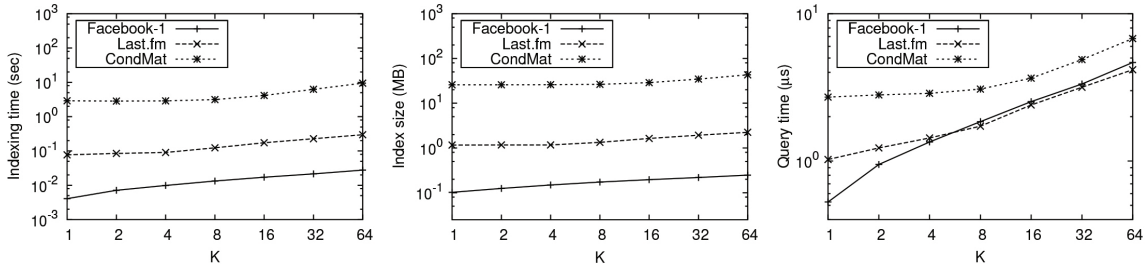
Algorithms. As there are no previous indexing methods for top-k distances, the proposed method was evaluated against the following two algorithms without precomputation.

- The first is the BFS-based naive approach, which uses a FIFO queue in the graph search, but which allows at most k visits to each vertex. This algorithm was also implemented in C++ by the authors.
- The second is Eppstein's algorithm (Eppstein 1998), which theoretically attains near-optimal time complexity. We adopted the C++ implementation of Jon Graehl[6].

[1] http://lovro.lpt.fri.uni-lj.si/support.jsp
[2] http://grouplens.org/datasets/hetrec-2011/
[3] http://snap.stanford.edu/
[4] http://socialnetworks.mpi-sws.org/datasets.html
[5] http://law.di.unimi.it/datasets.php (Boldi and Vigna 2004)
[6] http://www.ics.uci.edu/ eppstein/pubs/p-kpath.html

Table 2: Dataset information and performance of the proposed and existing methods on real-world datasets ($k = 8$).

Dataset				Top-k PLL (this work)			BFS	Eppstein
Name	Type	$\|V\|$	$\|E\|$	Indexing time	Index size	Query time		
Facebook-1	Social	334	2,218	13.7 ms	178.6 KB	1.9 μs	227.1 μs	378.4 μs
Last.fm	Social	1,892	12,717	125.3 ms	1.3 MB	1.7 μs	1.6 ms	7.5 ms
GrQc	Social	5,242	14,496	152.9 ms	2.7 MB	1.6 μs	2.2 ms	7.3 ms
HepTh	Social	9,877	25,998	631.2 ms	7.8 MB	2.2 μs	5.5 ms	16.5 ms
CondMat	Social	23,133	186,936	3.2 s	26.4 MB	3.1 μs	15.2 ms	158.8 ms
Facebook-2	Social	63,732	1,545,686	239.0 s	716.8 MB	15.2 μs	117.6 ms	2.7 s
YouTube-1	Social	1,157,828	4,945,382	624.3 s	2.3 GB	5.1 μs	1.5 s	7.0 s
YouTube-2	Social	3,238,848	18,512,606	1627.1 s	9.6 GB	3.9 μs	5.0 s	41.1 s
NotreDame	Web	325,729	1,497,134	52.3 s	617.7 MB	2.9 μs	249.8 ms	1.7 s
Stanford	Web	281,903	2,312,497	42.5 s	230.0 MB	1.7 μs	454.9 ms	2.9 s
BerkStan	Web	685,230	7,600,595	108.7 s	1.0 GB	1.9 μs	643.3 ms	10.8 s
Indo	Web	1,382,906	16,539,644	2695.3 s	6.0 GB	12.1 μs	1.4 s	25.4 s



(a) Indexing time (b) Index size (c) Query time

Figure 2: Effect of k on indexing time, index size, and query time.

Indexing Time and Index Size

The high scalability of the proposed method is evident from the index construction time and constructed index size reported in Table 2. Indices were constructed from large social and web graphs comprising tens of millions of edges (YouTube-2 and Indo) in one hour. The index sizes are below 10 GB, easily accommodated by the main memories of modern commodity computers.

While the index construction of all datasets was consistently efficient, we observe that the indexing time does not depend on graph size alone. The efficiency of the proposed method relies on the efficiency of pruning, and is thus related to network properties such as degree distribution and clustering coefficient. However, because the graphs of real-world social, web, computer and biological networks exhibit similar qualitative properties, the proposed method is robust and consistently efficient. The same argument is valid for index size.

Figure 2a and 2b illustrate the effect of k on the indexing time and index size in the proposed method. Both are relatively insensitive to the value of k.

Query Time

The proposed method generally answers queries within microseconds, very much faster than the other algorithms (Table 2). Indeed for the largest dataset, YouTube-2, the query time was six orders of magnitude faster than those of the BFS-based and Eppstein algorithms. In our experiments the BFS-based method was faster than Eppstein's algorithm. This is due to the big constant factor hidden in the O-notation of the time complexity of Eppstein's algorithm, as it involves complex data structure manipulation.

Figure 2c plots the query time as a function of k. Although the query time increased with k, it remained sufficiently fast at high k.

Application to Link Prediction

This section demonstrates the usefulness of the proposed method by applying it to the link prediction problem (Liben-Nowell and Kleinberg 2003). In particular, we confirm that top-k distances can contribute to prediction precision improvement. Note that our indexing method enables the first use of the top-k distances for such tasks, because top-k distances must be computed for many pairs of vertices during training and evaluation.

We selected link prediction as it is one of the most fundamental and popular problems on graphs in the AI and Web communities. However, the results suggest the applicability of top-k distances to other graph tasks such as network-aware searching.

Table 3: Predictive performance (AUC) of the method based on top-k distances and several baseline methods on the link prediction problem. Statistically significant winners (by paired t-test with $p < 0.05$) are highlighted in bold font.

Dataset	CN	Jaccard	Adamic	Preferential	Combined	SVD	RWR	Top-k	Top-1
Facebook-1	0.806	0.812	0.817	0.754	0.890	0.792	0.873	**0.901**	0.808
Facebook-2	0.776	0.777	0.777	0.875	0.755	0.823	**0.949**	0.931	0.931
Last.fm	0.596	0.597	0.603	0.831	0.861	0.644	0.844	**0.876**	0.802
GrQc	0.658	0.658	0.658	0.709	0.793	0.791	0.802	**0.824**	0.799
HepTh	0.546	0.546	0.547	0.686	0.714	0.774	0.779	**0.817**	0.775
CondMat	0.763	0.763	0.764	0.749	0.877	0.875	0.900	**0.929**	0.896

Setup

Prediction Settings. We randomly sampled 60% edges for training and reserved the remaining 40% for evaluation. The task was to predict the hidden evaluation edges given training edges. The sampling, prediction and evaluation procedures were performed 10 times on each datasets.

As an evaluation metric for predictive performance, we used *AUC* (area under the ROC curve). Generally, AUC is defined as the probability of predictions for positive examples larger than those for negative ones in test set. Since the tested datasets contain only positive, our investigation constitutes a "positive-and-unlabeled" case. Therefore, in our AUC evaluation, our test set is regarded as a subset of positive links. Withholding these links, we applied the evaluated methods to the dataset, treating the withheld links as no-links. We evaluated AUC as the probability that a randomly sampled withheld link has a higher predicted relational strength than a randomly sampled no-link vertex pair.

Method Based on Top-k Distances. We used the top-k distances as features in a support vector machine (SVM). More precisely, assuming $(\delta_1, \delta_2, \ldots, \delta_k)$ as the top-k distances of a vertex pair, we defined the features of the pair as k values, where the i-th value is given by $1/\sqrt{\delta_i}$. The value of k was tuned among $\{2^0, 2^1, \ldots, 2^6\}$, using one sample as a development dataset. Moreover, for each dataset we tested both linear SVM and non-linear SVM (with the RBF kernel) and selected the better performing one using the development dataset. The results on the development dataset are not included in the actual evaluation.

Baseline Methods. As baseline methods, we selected four methods commonly used in link prediction; (1) *CN* (Common neighbors), (2) *Jaccard*, (3) *Adamic*, (4) *Preferential* (Preferential Attachment). The scores of these methods were used as link scores. We also considered (5) a *Combined* method, in which the link scores of the four baseline methods were used as the features in SVM. We further compared our method with (6) *SVD* (singular value decomposition) (Golub and Loan 1996) and (7) *RWR* (Random Walk with Restart), and (8) *Top-1* distance. In SVD, the link score is calculated by cosine similarity based on the latent vectors. The number of latent dimension in SVD was tuned among $\{2^0, 2^1, \ldots, 2^8\}$, again using one sample as a development dataset. We adopted the RWR parameters of (He et al. 2004; Tong, Faloutsos, and Pan 2006). More specifically, the restart probability was set to 0.95 and the number of iterations was set to 50. The top-1 distance method corresponds to the above method with $k = 1$; its purpose was to evaluate the true importance of the top-k ($k > 1$) distances.

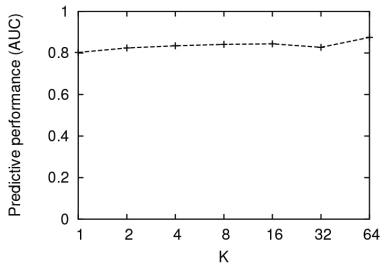


Figure 3: Effect of k on AUC, evaluated on the Last.fm dataset

Datasets. We used the six smaller social networks extracted from the graphs in the previous section, as some of the baseline methods (such as RWR) were too computationally expensive.

Results

Predictive Performance. Table 3 shows the mean AUCs for each method and dataset. The top-k distance method outperformed all the other methods for almost all the datasets. The exception was Facebook-2, for which the method based on top-k distances performed comparably to the top performer, RWR. Moreover, in most of the datasets, the top-k distances yielded higher performance than the top-1 (usual) distance.

Parameter Sensitivity. We experimentally studied the effect of the parameter k. Figure 3 plots the AUCs of the proposed method for the Last.fm dataset with k $\{2^0, 2^1, \ldots, 2^6\}$. We observe that the predictive performance was stable with respect to k.

Conclusion

In this study, we proposed a new indexing method that quickly answers top-k distance queries on large networks. Indeed, we present the first practical indexing method for top-k distances. The efficiency, scalability and robustness of the method was evaluated in extensive experiments on real-world social and web graphs. Moreover, by applying the method to link prediction, we indicated the practicability of top-k distance queries to various applications.

Acknowledgments

Takuya Akiba, Nozomi Nori and Yoichi Iwata are supported by Grant-in-Aid for JSPS Fellows (256563, 269329 and 256487, respectively). Takanori Hayashi and Yuichi Yoshida are supported by JST, ERATO, Kawarabayashi Large Graph Project. Yuichi Yoshida is supported by JSPS Grant-in-Aid for Young Scientists (B) (No. 26730009) and MEXT Grant-in-Aid for Scientific Research on Innovative Areas (No. 24106003).

References

Akiba, T.; Iwata, Y.; Kawarabayashi, K.; and Kawata, Y. 2014. Fast shortest-path distance queries on road networks by pruned highway labeling. In *ALENEX*, 147–154.

Akiba, T.; Iwata, Y.; and Yoshida, Y. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, 349–360.

Akiba, T.; Iwata, Y.; and Yoshida, Y. 2014. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *WWW*, 237–248.

Akiba, T.; Sommer, C.; and Kawarabayashi, K. 2012. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *EDBT*, 144–155.

Boldi, P., and Vigna, S. 2004. The webgraph framework I: compression techniques. In *WWW*, 595–602.

Cheng, J., and Yu, J. X. 2009. On-line exact shortest distance query processing. In *EDBT*, 481–492.

Cohen, E.; Halperin, E.; Kaplan, H.; and Zwick, U. 2002. Reachability and distance queries via 2-hop labels. In *SODA*, 937–946.

Eppstein, D. 1998. Finding the k shortest paths. *SIAM J. Computing* 28(2):652–673.

Fu, A. W.-C.; Wu, H.; Cheng, J.; Chu, S.; and Wong, R. C.-W. 2013. Is-label: an independent-set based labeling scheme for point-to-point distance querying on large graphs. *PVLDB* 6(6):457–468.

Golub, G. ., and Loan, C. . 1996. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences.

He, J.; Li, M.; jiang Zhang, H.; Tong, H.; and Zhang, C. 2004. Manifold-ranking based image retrieval. In *MM*, 9–16.

Ito, T.; Shimbo, M.; Kudo, T.; and Matsumoto, Y. 2005. Application of kernels to link analysis. In *KDD*, 586–592.

Jeh, G., and Widom, J. 2002. Simrank: a measure of structural-context similarity. In *KDD*, 538–543.

Jeh, G., and Widom, J. 2003. Scaling personalized web search. In *WWW*, 271–279.

Jin, R.; Ruan, N.; Xiang, Y.; and Lee, V. 2012. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *SIGMOD*, 445–456.

Liben-Nowell, D., and Kleinberg, J. 2003. The link prediction problem for social networks. In *CIKM*, 556–559.

Lovász, L. 1996. Random walks on graphs: A survey. In *Combinatorics, Paul Erdős is Eighty*, volume 2. János Bolyai Mathematical Society. 353–398.

Maniu, S., and Cautis, B. 2013. Network-aware search in social tagging applications: Instance optimality versus efficiency. In *CIKM*, 939–948.

McSherry, F. 2005. A uniform approach to accelerated pagerank computation. In *WWW*, 575–582.

Potamias, M.; Bonchi, F.; Castillo, C.; and Gionis, A. 2009. Fast shortest path distance estimation in large networks. In *CIKM*, 867–876.

Smola, A. J., and Kondor. 2003. Kernels and regularization on graphs. In *COLT*, 144–158.

Sun, J.; Qu, H.; Chakrabarti, D.; and Faloutsos, C. 2005. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, 418–425.

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

Ukkonen, A.; Castillo, C.; Donato, D.; and Gionis, A. 2008. Searching the wikipedia with contextual information. In *CIKM*, 1351–1352.

Vieira, M. V.; Fonseca, B. M.; Damazio, R.; Golgher, P. B.; Reis, D. d. C.; and Ribeiro-Neto, B. 2007. Efficient search ranking in social networks. In *CIKM*, 563–572.

Watts, D. J., and Strogatz, S. H. 1998. Collective dynamics of 'small-world' networks. *Nature* 393(6684):440–442.

Wei, F. 2010. Tedi: efficient shortest path query answering on graphs. In *SIGMOD*, 99–110.

Xiao, Y.; Wu, W.; Pei, J.; Wang, W.; and He, Z. 2009. Efficiently indexing shortest paths by exploiting symmetry in graphs. In *EDBT*, 493–504.

Yahia, S. A.; Benedikt, M.; Lakshmanan, L. V. S.; and Stoyanovich, J. 2008. Efficient network aware search in collaborative tagging sites. *PVLDB* 1(1):710–721.

Yano, Y.; Akiba, T.; Iwata, Y.; and Yoshida, Y. 2013. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *CIKM*, 1601–1606.