Journal of Computer and System Sciences 82 (2016) 23–44

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Tree decomposition-based indexing for efficient shortest path and nearest neighbors query answering on graphs

Fang Wei-Kleiner

Computer Science Department, Linköping University, Sweden

ARTICLE INFO

Article history:
Received 31 October 2014
Received in revised form 20 April 2015
Accepted 15 June 2015
Available online 15 July 2015

Keywords:
Graphs algorithms
Graph indexing
Shortest path
Tree decomposition
k Nearest Neighbors problems

ABSTRACT

We propose TEDI, an indexing for solving shortest path, and k Nearest Neighbors (kNN) problems. TEDI is based on the tree decomposition methodology. The graph is first decomposed into a tree in which the node contains vertices. The shortest paths are stored in such nodes. These local shortest paths together with the tree structure constitute the index of the graph. Based on this index, algorithms can be executed to solve the shortest path, as well as the kNN problem more efficiently. Our experimental results show that TEDI offers orders-of-magnitude performance improvement over existing approaches on the index construction time, the index size and the query answering.

1. Introduction

Querying and manipulating large scale graph-like data have attracted much attention in the database community, due to the wide application areas of graph data, such as ranked keyword search, XML databases, bioinformatics, social network, and ontologies. In this paper, we consider two fundamental graphs problems. (1) shortest path query answering; (2) k Nearest Neighbors problems.

In a ranked keyword search scenario over structured data, people usually give scores by measuring the link distance between two connected elements. If more than one path exists, it is desirable to retrieve the shortest distance between them, because shorter distance normally means higher rank of the connected elements [1–3]. In a social network application such as Facebook, registered users can be considered as vertices and edges represent the friend relationship between them. Normally two users are connected through different paths with various lengths. The problem of retrieving the shortest path among the users efficiently is of great importance.

As the online mapping and routing services such as Google Maps and other mobile apps become more and more popular, the need of the real time computation for these services is of vital importance. One of the most frequently used services is, for instance, to compute the shortest path between two locations. The typical setting of such road network problems is that we consider the map as an undirected graph where the locations are the nodes and the paths between the locations are modeled as edges. Another interesting problem is to find the nearest POIs (Point of Interest) given the user's current location. POIs can be, for instance, petrol stations, restaurants, etc.

The above road network problem can be formulated as the k Nearest Neighbors (kNN) graph problem, where $G = (V, E)$ is a weighted and undirected graph, and $u \in V$ is the vertex representing the users location, while $S \subseteq V$ representing the POIs. Note that though under the same name, we consider here the kNN problem over an undirected weighted graph. It

E-mail address: fangwise@gmail.com.

http://dx.doi.org/10.1016/j.jcss.2015.06.008
0022-0000/

differs from the kNN problem in spatial databases, where the objects are identified with the coordinates and the Euclidean distance is used as the distance measurement. It does not refer to the kNN problem in machine learning either.

For both of the above problems, there exist obvious solutions. For instance, Dijkstra's algorithm [4] computes the single source shortest paths (SSSP) of a given vertex $v \in G$ with the increasing order by maintaining a priority queue. The time complexity is $O(m + n \log n)$, where n is the number of vertices and m the number of edges in G. However, if the graph is of large size, the efficiency of query answering is expected to be improved. Obviously, query answering can be performed in constant time, if the shortest distances of the graph is pre-computed. However, the space overhead is n^2, if all pairs shortest distances have to be stored. Moreover, if the shortest paths are stored as well, a space overhead of n^3 is required, which is not affordable if n is of large value. Therefore, appropriate indexing and query scheme for answering shortest path queries must find the best trade-off between these two extreme methods.

In this paper, we propose **TEDI (TreE Decomposition based Indexing)**, an indexing and query processing scheme for more efficient graph query processing. Briefly stated, we first *decompose* the graph G into a tree in which each node contains a set of vertices in G. These tree nodes are called *bags*. Different from other partitioning based methods, there are overlappings among the bags, i.e., for any vertex v in G, there can be more than one bag in the tree which contains v. However, it is required that all these related bags constitute a connected subtree (see Definition 2 for the formal definition).

Based on the tree decomposition, we can execute the shortest path search in a bottom-up manner and the query time is decided by the height and the bag cardinality of the tree, instead of the size of the graph. If both of these two parameters are small enough, the query time can be substantially improved. Of course, in order to compute the shortest paths along the tree, we have to pre-compute the *local* shortest paths among the vertices in every bag of the tree. This constitutes the major part of the index structure of the TEDI scheme. Our main contributions are the following:

- **Solid theoretical background.** TEDI is based on the well-known concept of the tree decomposition, which is proved being of great importance in computational complexity theory. The abundant theoretical results provide a solid background for designing and correctness proofs of the TEDI algorithms.
- **Flexibility of balancing the time and space efficiency.** From the proposed tree decomposition algorithm, we discover an important correlation between the query time and the index size. If less space for the index is available, we can reduce the index size by increasing parameter k during the tree decomposition process, while the price to pay is longer query time. On the other hand, if the space is less critical, and the query time is more important, we can decrease k to achieve higher efficiency of the query answering. This flexibility offered by TEDI enables the users to choose the best time/space trade-off according to the system requirements.

We conducted experiments for both algorithms on real and synthetic datasets and compared the results with the other existing approaches. The experimental results confirm our theoretical analysis by demonstrating that TEDI offers orders-of-magnitude performance improvement over existing algorithms. Moreover, we conducted the experiments over large graphs such as DBLP and road networks. The encouraging results show that TEDI scales well on large datasets.

The rest of the paper is organized as follows: in Section 2 we formally define the shortest path query problem, as well as kNN problem over weighted graphs. In Section 3 we introduce the formal definitions of the tree decomposition and prove the theoretical results regarding the shortest path query answering. In Section 4 we present a practical tree decomposition algorithm and analyse the complexity. In the last two sections, we present the algorithm and complexity analysis, as well as the experimental results for the shortest path query (Section 5), and the kNN algorithm (Section 6), respectively. We survey related works in Section 8 and conclude in Section 9.

This article is based on two conference papers. Shortest distance computation over the tree decomposition-based indexes was published in [5]; k Nearest Neighbor computation was published in [6].

2. Problem definition

An undirected graph is defined as $G = (V, E)$, where $V = (0, 1, \ldots, n-1)$ is the vertex set and $E \subseteq V \times V$ is the edge set. Let $n = |V|$ be the number of vertices and $m = |E|$ be the number of edges. In this paper, we consider only undirected graphs, where $\forall u, v \in V : (u, v) \in E \Leftrightarrow (v, u) \in E$ holds. Each edge $(u, v) \in E$ is associated with a nonnegative weight between the two neighboring nodes, denoted as $w(u, v)$. A path P from node u to v is a sequence of nodes $P = (u, u_1, \ldots, u_l, v)$, where $(u, u_1), \ldots, (u_l, v) \in E$ and the length $|P|$ is the sum of the weight of all edges on P. We define the shortest distance $sdist(u, v)$ between two vertices u and v as the length of the shortest path $SP(u, v)$ of u and v.

Definition 1 *(k Nearest Neighbors (kNN) query).* Let $G = (V, E)$ be an undirected graph and $u \in V$. Let further $S \subseteq V$ and k be an integer such that $0 < k < |S|$. The k Nearest Neighbors (kNN) of u with respect to S is the set of vertices $S' = (s_1, \ldots, s_k)$, where $S' \subseteq S$, such that $sdist(u, s_1) + \ldots + sdist(u, s_k)$ is minimum. That is, $\forall x \in S \setminus S', \forall y \in S'$: $sdist(u, x) \geq sdist(u, y)$.

Intuitively, the problem of kNN over the weighted graph is a special form of *one point shortest path* computation, where the shortest distances from the source vertex u to the vertices in S are computed. In essence, we aim at computing the shortest distances from the starting vertex u to the vertices in S, and selecting k nearest neighbors in S with respect to u. A straight forward solution by computing the shortest distance one by one has the drawback that the performance