When Hierarchy Meets 2-Hop-Labeling: Efficient Shortest DistanceQu eries on Road Networks

Dian Ouyang
CAI, FEIT, University of Technology Sydney, Australia
dian.ouyang@student.uts.edu.au

Lu Qin
CAI, FEIT, University of Technology Sydney, Australia
lu.qin@uts.edu.au

Lijun Chang
The University of Sydney, Australia
lijun.chang@sydney.edu.au

Xuemin Lin
The University of New South Wales, Australia
lxue@cse.unsw.edu.au

Ying Zhang
CAI, FEIT, University of Technology Sydney, Australia
ying.zhang@uts.edu.au

Qing Zhu
School of Information, Renmin University of China, China
zq@ruc.edu.cn

ABSTRACT

Computing the shortest distance between two vertices is a fundamental problem in road networks. Since a direct search using the Dijkstra's algorithm results in a large search space, researchers resort to indexing-based approaches. State-of-the-art indexing-based solutions can be categorized into hierarchy-based solutions and hop-based solutions. However, the hierarchy-based solutions require a large search space for long-distance queries while the hop-based solutions result in a high computational waste for short-distance queries. To overcome the drawbacks of both solutions, in this paper, we propose a novel hierarchical 2-hop index (H2H-Index) which assigns a label for each vertex and at the same time preserves a hierarchy among all vertices. With the H2H-Index, we design an efficient query processing algorithm with performance guarantees by visiting part of the labels for the source and destination based on the vertex hierarchy. We also propose an algorithm to construct the H2H-Index based on distance preserved graphs. The algorithm is further optimized by computing the labels based on the partially computed labels of other vertices. We conducted extensive performance studies using large real road networks including the whole USA road network. The experimental results demonstrate that our approach can achieve a speedup of an order of magnitude in query processing compared to the state-of-the-art while consuming comparable indexing time and index size.

KEYWORDS

Shortest distance, road network, tree decomposition

ACM Reference Format:
Dian Ouyang, Lu Qin, Lijun Chang, Xuemin Lin, Ying Zhang, and Qing Zhu. 2018. When Hierarchy Meets 2-Hop-Labeling: Efficient Shortest Distance Queries on Road Networks . In *Proceedings of 2018 International Conference on Management of Data (SIGMOD'18).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3196913

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3196913

1 INTRODUCTION

Computing the shortest network distance between two locations is one of the most fundamental problems in road networks with many applications such as GPS navigation, POI recommendation, and route planning services. A road network can be modelled as a weighted graph $G(V, E)$ with vertex set V and edge set E. Given a source vertex $s \in V$ and a destination vertex $t \in V$, a shortest distance query asks for the shortest network distance from s to t in graph G. The classic approach to answer a shortest distance query is to use the Dijkstra's algorithm. Given a shortest distance query $q = (s, t)$, the Dijkstra's algorithm traverses the vertices in graph G in non-decreasing order of their distances to s and terminates when the destination vertex t is reached. Dijkstra's algorithm can compute the shortest distance between two vertices in polynomial time. Nevertheless, when the network is large, the Dijkstra's algorithm cannot satisfy the real-time requirements for a shortest distance query since it may traverse the whole network when the two query vertices are far away from each other. Consequently, researchers resort to indexing-based approaches by investigating the characteristics of road networks such as the low-degree and planarity properties [1–3, 8–10, 16, 17, 21, 24, 27–30, 34, 36].

The state-of-the-art approaches for shortest distance query processing on road networks mainly fall into two categories, namely, hierarchy-based solutions [16, 36] and hop-based solutions [1–3]. The hierarchy-based solutions precompute a hierarchy of vertices in the road network and add some shortcuts from lower-hierarchy vertices to higher-hierarchy vertices. Given a shortest distance query $q = (s, t)$, a hierarchy-based solution starts searching from both s and t simultaneously in a bidirectional manner using the bidirectional Dijkstra's algorithm, and the search direction is confined to be only from lower-hierarchy vertices to higher-hierarchy vertices. In this way, the search space is significantly reduced. The hop-based solutions precompute a 2-hop label for each vertex in the road network. The 2-hop label for a vertex includes the shortest distances from the vertex to a subset of vertices in the road network. Given a shortest distance query $q = (s, t)$, the shortest distance of s and t can be answered using only the labels of s and t by joining the common vertices in their labels.

Motivations. By investigating the above two categories of solutions, we make the following observations. First, a hierarchy-based solution is efficient to answer a shortest distance query $q = (s, t)$ when s and t are near each other in the road network. However,

when s and t are far away from each other, a hierarchy-based solution still has to exploit a large number of vertices in the road network because the search space of the bidirectional Dijkstra's algorithm increases rapidly when the distance between s and t increases. Second, for the case where s and t are far away from each other, a hop-based solution is much more efficient than a hierarchy-based solution because the hop-based solution can answer a shortest distance query by only exploiting the labels of s and t without searching the road network. However, when s and t are near each other in the road network, a hop-based solution still needs to scan the entire labels of s and t to answer the query. In this case, a large number of useless vertices in the labels are visited. Motivated by these observations, in this paper, we propose a novel solution that can overcome the drawbacks of both hierarchy-based and hop-based solutions.

Our Idea. Our general idea is simple: Given a road network, we aim to design a label for each vertex as well as a hierarchy among all vertices in the road network. To answer a shortest distance query $q = (s, t)$, instead of exploiting all vertices in the labels of s and t, we only need to visit a subset of vertices in the labels of s and t adaptively with the help of the vertex hierarchy. Intuitively, the number of visited vertices decreases when the distance between s and t decreases. When the distance between s and t is small, only a small subset of vertices in the labels of s and t need to be visited. Existing hierarchy-based and hop-based solutions each focus on one aspect and cannot be simply combined to achieve our goal. To make our idea practically applicable, the following issues need to be addressed: (1) how tofi nd an appropriate hierarchy for all vertices; (2) how to assign labels for each vertex so that the hierarchy can be used to efficiently answer the shortest distance queries; and (3) how to efficiently compute the hierarchy and the vertex labels.

Contributions. In this paper, we answer the above questions and make the following contributions:

- We investigate the drawbacks of the existing hierarchy-based and the hop-based solutions for shortest distance query processing in road networks. We introduce a new index named Hierarchical 2-Hop Index (H2H-Index) based on the concept of tree decomposition. Based on the H2H-Index, we design an efficient algorithm to answer a shortest distance query in $O(w)$ time in the worst case, where w is the treewidth of the tree decomposition which is small for road networks. Compared to the state-of-the-art hop-based algorithms, our algorithm can avoid visiting a large number of unnecessary vertices in the labels with the assistant of the vertex hierarchy, and thus achieves a much faster query processing time while consuming a similar index space.
- We design an algorithm to construct the H2H-Index based on the concepts of distance preserved graph and distance preserved tree decomposition. We further improve the efficiency of the algorithm by computing the new labels using the partial labels generated for other vertices. Our new algorithm enforces a certain vertex visiting order, in order to maximumly reuse the existing information to reduce the overall computational cost. The index construction algorithm has a bounded worst-case time complexity and thus it can handle large road networks efficiently.
- We conduct extensive performance studies to test the performance of the proposed algorithms on real large road networks including the whole road network of the USA. The experimental results demonstrate that our algorithm can achieve a speedup of an order of magnitude in query processing compared to the state-of-the-art while consuming comparable indexing time and index size.

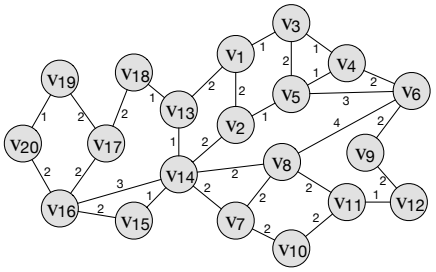


Figure 1: An Example of a Road Network G

2 PROBLEM STATEMENT

Let $G = (V, E)$ be a road network (i.e., a degree-bounded connected and weighted graph) where $V(G)$ is the set of vertices and $E(G)$ is the set of edges. We use V and E to denote $V(G)$ and $E(G)$, and use $n = |V|$ and $m = |E|$ to denote the number of vertices and edges in the road network respectively. For each vertex $v \in V$, the neighbors of v, denoted as $N(v, G)$, is defined as $N(v, G) = \{u|(u, v) \in E(G)\}$. We use $N(v)$ to denote $N(v, G)$ when the context is obvious. Each edge $e = (u, v) \in E$ is associated with a positive weight $\phi(u, v)$. A path is a sequence of vertices $p = (v_1, v_2, \ldots, v_k)$ where $(v_i, v_{i+1}) \in E$ for each $1 \leq i < k$. The weight of the path p, denoted as $\phi(p)$, is defined as $\phi(p) = \sum_{i=1}^{k-1} \phi(v_i, v_{i+1})$. Given two vertices $s, t \in V$, a shortest path p between s and t is a path starting at s and ending at t such that $\phi(p)$ is minimized. The shortest distance of s and t in G, denoted as $dist_G(s, t)$, is the weight of any shortest path between s and t. We use $dist(s, t)$ to denote $dist_G(s, t)$ if the context is obvious. For the ease of explanation, we consider G as an undirected graph in this paper, and our techniques can be easily extended to handle directed graphs.

Problem Definition. Given a road network G, a shortest distance query is defined as $q = (s, t)$ where $s, t \in V(G)$, and the answer of q is the shortest distance $dist(s, t)$. In this paper, we aim to develop effective indexing techniques so that q can be answered efficiently.

Example 2.1. Fig. 1 shows a road network $G = (V, E)$ with 20 vertices and 30 edges. The weight of each edge is marked beside the corresponding edge. For example, for edge $(v_6, v_8) \in E$ we have $\phi(v_6, v_8) = 4$. There are many paths between v_{12} and v_{19}. For example, two of them are $p_1 = (v_{12}, v_{11}, v_8, v_{14}, v_{13}, v_{18}, v_{17}, v_{19})$ and $p_2 = (v_{12}, v_9, v_6, v_5, v_2, v_{14}, v_{16}, v_{20}, v_{19})$ with $\phi(p_1) = 11$ and $\phi(p_2) = 18$. p_1 is a shortest path between v_{12} and v_{19}. Given a shortest distance query $q = (v_{12}, v_{19})$, the answer of q is $dist(v_{12}, v_{19}) = 11$.

3 EXISTING SOLUTIONS

Given a shortest distance query $q = (s, t)$ on a road network $G = (V, E)$, we can use a traditional single source shortest algorithm (e.g., the Dijkstra Algorithm) to compute the shortest distance between s and t. Nevertheless, the algorithm is inefficient because it requires a time complexity of $O(m + n \cdot \log(n))$ and may traverse the whole network to answer one query in the worst case. Therefore,

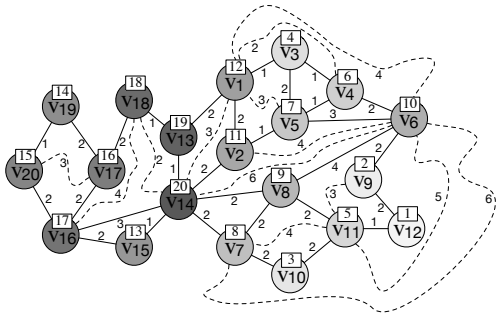


Figure 2: The index structure G_I of CH

researchers resort to indexing-based solutions to accelerate query processing. In the literature, the state-of-the-art indexing-based solutions for shortest distance query processing on road networks mainly fall into two categories, namely, hierarchy-based solutions and hop-based solutions. We briefly introduce them below.

3.1 Hierarchy-based Solution

A hierarchy-based solution is an indexing approach that imposes a hierarchy of all vertices in G by assigning each vertex v a rank $r(v)$. Based on the vertex hierarchy, it pre-computes the shortest distance among a subset of vertex pairs to reduce the computational cost of online query processing. Existing hierarchy-based solutions include contraction hierarchies (CH) [16] and arterial hierarchy (AH) [36].

Contraction Hierarchies (CH). The hierarchy-based solution is first introduced by Geisberger et al. [16] in which an approach named Contraction Hierarchies (CH) is introduced. In the index construction phase, CH first assigns a total order of vertices in G as the vertex hierarchy by assigning each vertex v a rank $r(v)$ using some predefined criteria. Then CH examines each vertex v_i following the total order. For each vertex v_i, CH visits the neighbors $N(v_i)$ of v_i in G. For each pair $v_j, v_k \in N(v_i)$, if the shortest path between v_j and v_k passes through v_i, an artificial edge (v_j, v_k) (referred to as a shortcut) is added into G with weight $\phi(v_j, v_k) = dist(v_j, v_k)$. Once all pairs of neighbors in v_i are processed, v_i is removed from the network G. The index construction phase terminates after all vertices are examined. The original network G along with all the shortcuts added in the index construction phase form the index of CH.

Given the CH index G_I which includes the network G along with the shortcuts, a shortest distance query $q = (s, t)$ can be answered using a modified bidirectional Dijkstra's algorithm. Specifically, we start searching from both s and t simultaneously in a bidirectional manner using the Dijkstra's algorithm on the constructed graph G_I. Each time when expanding from a vertex u to vertex v, we only consider those edges (u, v) such that the rank of v is higher than that of u in G_I. When the termination condition of the bidirectional Dijkstra's algorithm is satisfied, the algorithm reports the current best distance as the answer to q. Since CH enforces the expansion direction to be only from lower-ranked vertices to higher-ranked vertices, the search space of CH is significantly reduced compared to the traditional bidirectional Dijkstra's algorithm when answering a shortest distance query q.

The arterial hierarchy (AH) approach is introduced in Section 10.1 in the Appendix.

V	2-Hop Label
v_1	$\{(v_1, 0), (v_2, 2), (v_{13}, 2), (v_{14}, 3)\}$
v_2	$\{(v_2, 0), (v_{14}, 2)\}$
v_3	$\{(v_1, 1), (v_2, 3), (v_3, 0), (v_4, 1), (v_5, 2), (v_6, 3), (v_{13}, 3), (v_{14}, 4)\}$
v_4	$\{(v_1, 2), (v_2, 2), (v_4, 0), (v_5, 1), (v_6, 2), (v_{13}, 4), (v_{14}, 4)\}$
v_5	$\{(v_1, 3), (v_2, 1), (v_5, 0), (v_6, 3), (v_{14}, 3)\}$
v_6	$\{(v_1, 4), (v_2, 4), (v_6, 0), (v_{13}, 6), (v_{14}, 6)\}$
v_7	$\{(v_6, 6), (v_7, 0), (v_8, 2), (v_{14}, 2)\}$
v_8	$\{(v_6, 4), (v_8, 0), (v_{14}, 2)\}$
v_9	$\{(v_1, 6), (v_2, 6), (v_6, 2), (v_7, 7), (v_8, 5), (v_9, 0), (v_{11}, 3), (v_{13}, 8), (v_{14}, 7)\}$
v_{10}	$\{(v_6, 7), (v_7, 2), (v_8, 4), (v_{10}, 0), (v_{11}, 2), (v_{14}, 4)\}$
v_{11}	$\{(v_6, 5), (v_7, 4), (v_8, 2), (v_{11}, 0), (v_{14}, 4)\}$
v_{12}	$\{(v_1, 8), (v_6, 4), (v_7, 5), (v_8, 3), (v_9, 2), (v_{11}, 1), (v_{12}, 0), (v_{14}, 5)\}$
v_{13}	$\{(v_{13}, 0), (v_{14}, 1)\}$
v_{14}	$\{(v_{14}, 0)\}$
v_{15}	$\{(v_{14}, 1), (v_{15}, 0), (v_{16}, 2)\}$
v_{16}	$\{(v_{14}, 3), (v_{16}, 0), (v_{18}, 4)\}$
v_{17}	$\{(v_{13}, 3), (v_{14}, 4), (v_{16}, 2), (v_{17}, 0), (v_{18}, 2)\}$
v_{18}	$\{(v_{13}, 1), (v_{14}, 2), (v_{18}, 0)\}$
v_{19}	$\{(v_{13}, 5), (v_{14}, 6), (v_{16}, 3), (v_{17}, 2), (v_{18}, 4), (v_{19}, 0), (v_{20}, 1)\}$
v_{20}	$\{(v_{13}, 6), (v_{14}, 5), (v_{16}, 2), (v_{17}, 3), (v_{18}, 5), (v_{20}, 0)\}$

Table 1: 2-hop Labels for the Road Network in Fig. 1

Example 3.1. In this example, we illustrate the CH algorithm using the road network shown in Fig. 1. Suppose the vertex rank is assigned as $v_{12} < v_9 < v_{10} < v_3 < v_{11} < v_4 < v_5 < v_7 < v_8 < v_6 < v_2 < v_1 < v_{15} < v_{19} < v_{20} < v_{17} < v_{16} < v_{18} < v_{13} < v_{14}$. We mark the rank of each vertex in a box beside each vertex in Fig. 2, and we also use the darkness of each vertex to illustrate its rank: a darker color indicates a vertex with a higher rank. In index construction, the CH algorithmfi rst examines v_{12} with the lowest rank. For its two neighbors v_9 and v_{11}, since the shortest path from v_9 to v_{11} (with weight 3) passes through v_{12}, we add a shortcut (v_9, v_{11}) with weight 3 in G_I. Next, the vertex v_9 is examined, which has two neighbors v_6 and the newly added neighbor v_{11} (v_{12} is not neighbor of v_9 since it has been marked as removed). A shortcut (v_6, v_{11}) with weight 5 is added in G_I. The algorithm continues until all vertices are examined. Thefi nal index is shown in Fig. 2 where each dashed edge is a shortcut.

With the index G_I, given a shortest distance query $q = (v_3, v_{10})$, using the modified bidirectional Dijkstra's algorithm, we can expand from v_3 to v_6 through the path (v_3, v_4, v_6) with increasing vertex ranks and expand from v_{10} to v_6 through the path (v_{10}, v_{11}, v_6) with increasing vertex ranks where edge (v_{11}, v_6) is a shortcut, and thus obtain the shortest path from v_3 to v_{10} with $dist(v_3, v_{10}) = 10$.

3.2 Hop-based Solution

Given a road network $G(V, E)$, the hop-based solution aims to assign each vertex $v \in V$ a 2-hop label $L(v)$ which is a collection of pairs $(u, dist(v, u))$ where $u \in V$. Given two vertices $s, t \in V$, the shortest distance of s, t can be calculated as follows:

$$dist(s, t) = \min_{u \in L(s) \cap L(t)} dist(s, u) + dist(u, t) \quad (1)$$

In other words, the hop-based solution answers a query $q = (s, t)$ by considering all common vertices in the labels of s and t. For each such common vertex u, a distance is calculated using 2 hops from s to u and from u to t. The labels are assigned in a way to guarantee that the minimum of the two-hop distances for all $u \in L(s) \cap L(t)$ is the shortest distance from s to t. In the literature, two popular hop-based approaches are the hub-based labeling approach (HL) [1, 2] and the pruned highway labeling approach (PHL) [3]. More details of the two approaches can be found in Section 10.1 in the Appendix.

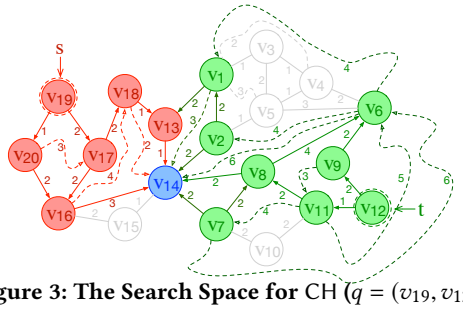


Figure 3: The Search Space for CH ($q = (v_{19}, v_{12})$)

Example 3.2. Table 1 shows the 2-hop labels for the road network in Fig. 1 computed using the hub-based labeling algorithm. The label of v_2 is $L(v_2) = \{(v_2, 0), (v_{14}, 2)\}$ which indicates that $dist(v_2, v_2) = 0$ and $dist(v_2, v_{14}) = 2$. To answer query $q = (v_5, v_{12})$, according to Eq. 1, we canfi rst compute $L(v_5) \cap L(v_{12}) = \{v_1, v_6, v_{14}\}$. Then we can obtain $dist(v_5, v_{12}) = \min\{dist(v_5, v_1) + dist(v_{12}, v_1), dist(v_5, v_6) + dist(v_{12}, v_6), dist(v_5, v_{14}) + dist(v_{12}, v_{14})\} = \min\{3 + 8, 3 + 4, 3 + 5\} = 7$.

4 2-HOP IN A HIERARCHY

4.1 Problem Analysis.

In this section, we analyze the drawbacks of the hierarchy-based solution and the hop-based solution.

Hierarchy-based Solution. The hierarchy-based solution defines a hierarchy for vertices in a road network. To answer a shortest distance query $q = (s, t)$, the hierarchy-based solution explores the road network from s and t simultaneously following the predefined order in the hierarchical index, and thus significantly reduces the search space compared to the strategy by searching from scratch on the road network. The algorithm is efficient for the case when s and t are near. Nevertheless, for a long-distance query $q = (s, t)$, i.e., s is far away from t, the algorithm may still spend a large among of search space and computational cost when searching the hierarchical index bidirectionally. We show an example to demonstrate the above cases in the following.

Example 4.1. Consider the road network in Fig. 1, the hierarchy-based index G_I is shown in Fig. 2. To compute $dist(v_1, v_6)$, we only need to search one hop-neighbors from v_1 and v_6 following the hierarchical order and compute $dist(v_1, v_6)$ efficiently using the shortcut (v_1, v_6) with weight 4. However, when we compute $dist(v_{19}, v_{12})$, the search space is shown in Fig. 3. The vertices and edges marked in the red color indicate the search space from the source vertex v_{19} and the vertices and edges marked in the blue color indicate the search space from the target vertex v_{12}. As illustrated in Fig. 2, since v_{19} is far away from v_{12}, more than half of the vertices and edges (including shortcuts) have to be visited during the search, which is costly.

Hop-based Solution. Given a road network G, the hop-based solution assigns a 2-hop label $L(v)$ for each $v \in V(G)$. To answer $q = (s, t)$, the algorithm simply joins $L(s)$ and $L(t)$ to compute the answer to the query. For a long-distance query, the algorithm avoids online searching the intermediate vertices in the path from s to t and therefore is much more efficient compared to the hierarchy-based solution. Nevertheless, for each vertex, the algorithm uses the same label to answer all queries with the vertex. When the distance between s and t is short, the algorithm may result in unnecessary computations when the label sizes of s and t are large. Below, we provide an example to demonstrate these situations.

Example 4.2. Consider the road network in Fig. 1, the 2-hop label $L(v)$ for each vertex v in the road network is shown in Table 1. To answer $q = (v_6, v_{16})$, we only need to consider the 5 vertices in $L(v_6)$ and 3 vertices in $L(v_{16})$, with a smaller search space compared to the hierarchy-based solution. However, to answer $q = (v_9, v_{12})$, we need to consider the 9 vertices in $L(v_9)$ and the 8 vertices in $L(v_{12})$ with a large amount of useless computations, whereas in the hierarchy-based solution we only need to explore the neighbor of v_{12} to answer the query.

Our Solution. Based on the above analysis, we can see that the hierarchy-based solution is inefficient when answering long-distance queries and the hop-based solution may result in large useless computations when answering short-distance queries. To handle both cases more efficiently, in this paper, we propose a hierarchical 2-hop index. Our general idea is as follows: we assign each vertex v a 2-hop label $L(v)$ and we also assign a hierarchy for all vertices in the road network. Given a shortest distance query $q = (s, t)$, instead of using all vertices in $L(s)$ and $L(t)$ to answer the query, we only pick up a subset of vertices in $L(s)$ and $L(t)$ based on the hierarchy. In general, when s and t are far away from each other, most vertices in $L(s)$ and $L(t)$ are selected, and when s and t are near each other, only a small part of $L(s)$ and $L(t)$ are selected. In this way, we can overcome the drawbacks of both the hierarchy-based and the hop-based solutions. To make the idea practically applicable, the following issues need to be addressed: (1) how tofi nd an appropriate hierarchy for all vertices, and (2) how to assign labels for each vertex so that the hierarchy can be used to efficiently answer the shortest distance queries. We will answer these questions in the following sections.

4.2 Vertex Hierarchy by Tree Decomposition

Tree decomposition, which is originally introduced by Halin [18] and rediscovered by Robertson and Seymour [26], is a way to map a graph to a tree to speed up solving certain computational problems in graphs. Many algorithmic problems, such as maximum independent set and Hamiltonian circuits that are NP-complete for arbitrary graphs, may be solved efficiently by dynamic programming for graphs of bounded treewidth, using the tree-decompositions of these graphs. An introductory survey by Bodlaender can be found in [12]. Tree decomposition naturally assigns a hierarchy among vertices. Therefore, in this paper, we use tree decomposition to define the vertex hierarchy, and we show that the hierarchy is effective in answering shortest distance queries in a road network. Given a road network $G(V, E)$, a tree decomposition of G is defined as follows [12]:

Definition 4.3. **(Tree Decomposition)** A tree decomposition of a graph $G(V, E)$, denoted as T_G, is a rooted tree in which each node $X \in V(T_G)$ is a subset of $V(G)$ (i.e., $X \subseteq V(G)$) such that the following three conditions hold:

(1) $\bigcup_{X \in V(T_G)} X = V$;
(2) For every $(u, v) \in E(G)$, there exists $X \in V(T_G)$ s.t. $u \in X$ and $v \in X$;
(3) For every $v \in V(G)$ the set $\{X | v \in X\}$ forms a connected subtree of T_G.

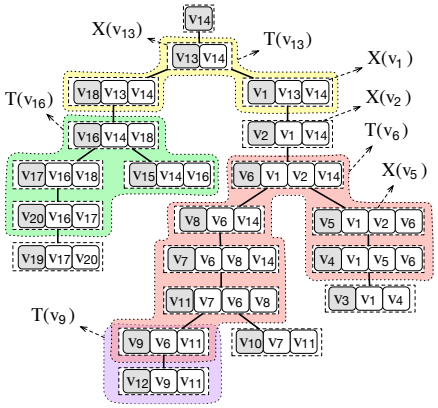


Figure 4: Tree Decomposition T_G of the Network in Fig. 1

For any $v \in V(G)$, we use $T(v)$ to denote the subtree induced by the set $\{X|v \in X\}$ in T_G, and use $X(v)$ to denote the root node of $T(v)$ in T_G.

For ease of presentation, we refer to each $v \in V(G)$ in the road network G as a *vertex* and refer to each $X \in V(T_G)$ in the tree decomposition T_G as a *node*. We consider T_G as a rooted tree by picking up a node as a root. In the following, we provide an example to illustrate the tree decomposition of a road network.

Example 4.4. Fig. 4 shows a tree decomposition T_G of the road network G in Fig. 1. There are 20 nodes in T_G. One of the nodes containing 3 vertices is $\{v_{18}, v_{13}, v_{14}\}$. For the edge $(v_{17}, v_{18}) \in E(G)$, there is a node $\{v_{17}, v_{16}, v_{18}\}$ in $V(T_G)$ containing both v_{17} and v_{18}. For the vertex $v_{13} \in V(G)$, there are three nodes in $V(T_G)$ containing v_{13}. The three nodes form a subtree $T(v_{13})$ as marked by yellow. The root of $T(v_{13})$ is denoted as $X(v_{13})$. We also mark $T(v_6)$, $T(v_9)$, $T(V_{16})$, $X(v_1)$, $X(v_2)$, and $X(v_5)$ in thefi gure.

Definition 4.5. **(Treewidth and Treeheight)** Given a tree decomposition T_G of graph G, the treewidth of T_G, denoted as $w(T_G)$ is one less than the maximum size of all nodes in T_G, i.e., $w(T_G) = \max_{X \in T_G} |X| - 1$. The treeheight of T_G, denoted as $h(T_G)$, is the maximum depth of all nodes in T_G. Here the depth of a node in T_G is the distance from the node to the root node of T_G. If the context is obvious, we will use w and h to denote the treewidth and treeheight of the tree decomposition T_G respectively. The treewidth of a graph G is the minimum treewidth over all possible tree decompositions of G.

Example 4.6. For the road network G in Fig. 1, a tree decomposition T_G is shown in Fig. 4. The treewidth $w(T_G)$ of T_G is 3 since each node in T_G contains at most 4 vertices. The treeheight $h(T_G)$ of T_G is 10.

It is NP-complete to determine whether a given graph G has treewidth at most a given variable [6]. Existing techniques to compute the optimal tree decomposition with the minimum treewidth can only handle small graphs (e.g., graphs with less than $1,000$ vertices) [22]. Therefore, in this paper, we adopt a suboptimal algorithm, which is introduced in [14] with a time complexity of $O(n \cdot (w^2 + \log(n)))$. The details of the algorithm is introduced in Section 10.2 in the Appendix. The algorithm guarantees that every $X(v)$ is unique, i.e., for any $v \neq u$, we have $X(v) \neq X(u)$. Therefore, there is a one-to-one mapping from $V(G)$ to $V(T_G)$. In this paper, we assume this property holds for a tree decomposition T_G for the ease of presentation. Note that, with the one-to-one mapping, the tree decomposition is not necessarily in its simplest form. However, this will not result in any redundant information in our constructed H2H-Index which will be introduced in Section 5.

It is important to note that in a road network, we can obtain a tree decomposition with low treewidth and treeheight values. For example, in the road network for the City of New York with 264, 346 vertices and 733, 846 edges, we can obtain a tree decomposition with treewidth of only 131 and treeheight of only 320. The treewidth and treeheight values for more road networks including the road network for the whole USA are shown in Section 7.

4.3 A Naive Labeling Approach

In this subsection, we introduce a straightforward solution to solve the shortest distance queries using tree decomposition based on the concept of a vertex cut.

Definition 4.7. **(Vertex Cut)** Given a road network $G(V, E)$, a subset of vertices $C \subset V$ is a vertex cut of G if the deletion of C from G splits G into multiple connected components. A vertex set C is called the vertex cut of vertices u and v if u and v are in different connected components by the deletion of C from G.

Example 4.8. For the road network G shown in Fig. 1, the set $C = \{v_{13}, v_{14}\}$ is a vertex cut of G. C is a vertex cut of vertices v_1 and v_{18}, but it is not a vertex cut of v_1 and v_2, because after the removal of C from G, v_1 and v_{18} are in different connected components while v_1 and v_2 are in the same connected component.

Given a vertex cut C for two vertices s and t, it is obvious that every path from s to t should contain at least one vertex in C. Therefore, we can easily derive the following theorem:

Theorem4.9. *Given a road network G, let C be a vertex cut for two vertices s and t, we have:*

$$dist(s, t) = \min_{v \in C} dist(s, v) + dist(v, t)$$

According to Theorem 4.9, if we can obtain a small cut C for vertices s and t and precompute the distances from s to the vertices in C and from the vertices in C to t, we can calculate $dist(s, t)$ efficiently. In the following, we show that the tree decomposition T_G can be used to complete the task, based on the following cut property [15] of a tree decomposition.

Property1. *Given a tree decomposition T_G for a road network G, for any two vertices s and t in $V(G)$, suppose $X(s)$ is not an ancestor/decedent of $X(t)$ in T_G, let X be the lowest common ancestor (LCA) of $X(s)$ and $X(t)$ in T_G, then X is a vertex cut of s and t in G.*

Note that X may not be the minimum vertex cut of s and t in G. Using the techniques in [11], we can use $O(1)$ time to compute the LCA of any pair of nodes in T_G using $O(n)$ space.

Example 4.10. For the road network G (Fig. 1) and its tree decomposition T_G shown in Fig. 4, given two vertices v_3 and v_9, the lowest common ancestor of $X(v_3)$ and $X(v_9)$ in T_G is $X(v_6) = \{v_6, v_1, v_2, v_{14}\}$. Therefore, $\{v_6, v_1, v_2, v_{14}\}$ is a vertex cut of v_3 and v_9 in G. According to Theorem 4.9, we can get $dist(v_3, v_9) = \min\{dist(v_3, v_6) + dist(v_6, v_9), dist(v_3, v_1) + dist(v_1, v_9), dist(v_3, v_2) + dist(v_2, v_9), dist(v_3, v_{14}) + dist(v_{14}, v_9)\} = 5$.

The Naive Solution. Based on Property 1 and Theorem 4.9, given a tree decomposition T_G, we can devise a straightforward solution to answer shortest distance queries as follows:

- *Indexing.* In the indexing phase, for each $v \in V(G)$ and each ancestor $X(u)$ of $X(v)$ in T_G, we precompute the shortest distance $dist(v, x)$ for any $x \in X(u)$ and index them using a hash table. For each vertex v, at most $h \times (w+1)$ shortest distances are computed. Therefore, the total index size is bounded by $O(n \cdot h \cdot w)$.
- *Query Processing.* In the query processing phase, given a query $q = (s, t)$, we canfi rst compute the LCA X of $X(s)$ and $X(t)$ in T_G using $O(1)$ time [11]. Then we can compute $dist(s, t)$ according to Property 1 and Theorem 4.9. i.e.,

$$dist(s, t) = \min_{v \in X} dist(s, v) + dist(v, t) \quad (2)$$

Here $dist(s, v)$ and $dist(v, t)$ for every $v \in X$ have been precomputed in the indexing phase. Obviously, the query processing time is $O(c \cdot w)$ where c is the time cost to look up the distance for a specific pair of vertices in the hash table.

5 QUERY PROCESSING BY H2H-INDEX

In this section, wefi rst analyze the drawbacks of the naive solution, followed by the introduction of a more effective indexing and query processing mechanism.

Drawbacks of the Naive Solution. In the following, we discuss the drawbacks of the naive solution in terms of index size and query processing time.

- *Index Size.* The index size of $O(n \cdot h \cdot w)$ for the naive solution can be very large, which makes the approach impractical for handling large road networks, such as the whole USA road network.
- *Query Processing Time.* The query processing complexity of the naive solution using hash tables is $O(c \cdot w)$. Here, c is a constant in theory. Nevertheless, compared to array accesses, hash table lookups are usually much more expensive to answer a shortest distance query because of the following two reasons. First, the factor c in a hash lookup is usually a large value. Second, to answer a shortest distance query, we need $O(w)$ hash table lookups according to Eq. 2. Each hash table lookup may generate a random access in the memory. The random accesses are usually more expensive than sequentially accessing an array in the memory.

Hierarchical 2-Hop Index (H2H-Index). To overcome these drawbacks, we introduce a new index structure called hierarchical 2-hop index (H2H-Index). The index is designed based on the following property:

Property2. *Given a road network G, its tree decomposition T_G, and an arbitrary node $X(u) \in V(T_G)$, for any $v \in X(u) \setminus \{u\}$, $X(v)$ is an ancestor of $X(u)$ in T_G.*

Next, we define the ancestor array for a node in a tree decomposition as follows:

Definition 5.1. **(Ancestor Array)** Given a tree decomposition T_G of G, for any node $X(v) \in V(T_G)$, let $(X(w_1), X(w_2), \ldots, X(w_l))$ be the path from the root of T_G to $X(v)$ in T_G, here $X(w_1)$ is the root of T_G and $X(w_l)$ is $X(v)$. The *ancestor array* of $X(v)$, denoted as $X(v).anc$ is defined as $X(v).anc = (w_1, w_2, \ldots, w_l)$. We use $X(v).anc_i$ to denote the i-th element in $X(v).anc$ for any $1 \le i \le l$.

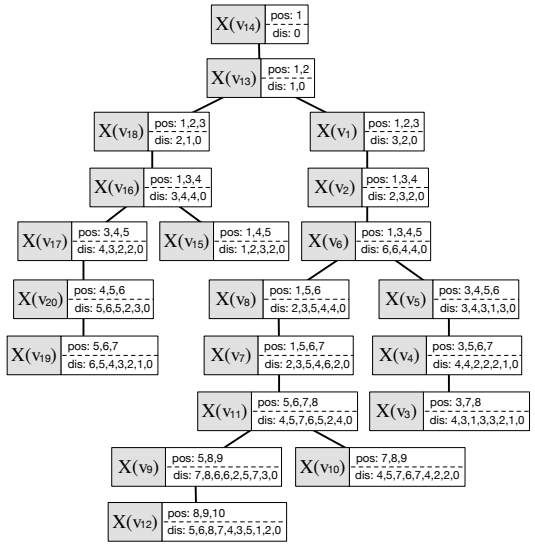


Figure 5: The H2H-Index for the Road Network in Fig. 1

Example 5.2. Given the tree decomposition T_G shown in Fig. 4, for the node $X(v_7) = \{v_7, v_6, v_8, v_{14}\}$, according to Property 2, the nodes $X(v_6)$, $X(v_8)$, and $X(v_{14})$ are all ancestors of $X(v_7)$ in T_G. The ancestor array of $X(v_7)$ is $X(v_7).anc = (v_{14}, v_{13}, v_1, v_2, v_6, v_8, v_7)$.

Based on Property 2 and Definition 5.1, we now define the structure of the H2H-Index. The H2H-Index is defined on top of the tree decomposition T_G of road network G. For each node $X(v) \in V(T_G)$, suppose $X(v) = \{u_1, u_2, \ldots, u_k\}$ and $X(v).anc = (w_1, w_2, \ldots, w_l)$, according to Property 2, $X(v)$ is a subset of $X(v).anc$, i.e., $X(v) \subseteq X(v).anc$. The H2H-Index for $X(v)$ consists of two components: distance array and position array.

- *Distance Array:* the distance array for $X(v)$, denoted as $X(v).dis$, is an array defined as $X(v).dis = (dist(v, w_1), dist(v, w_2), \ldots, dist(v, w_l))$. In other words, the distance array of $X(v)$ is the array of distances from v to every vertex in $X(v).anc$. We use $X(v).dis_i$ to denote the i-th value in $X(v).dis$ for any $1 \le i \le l$. We have $X(v).dis_i = dist(v, X(v).anc_i)$.
- *Position Array:* the position array for $X(v)$, denoted as $X(v).pos$, is an array defined as $X(v).pos = (p_1, p_2, \ldots, p_k)$ where $p_i (1 \le i \le k)$ is the position of u_i in $X(v).anc$, i.e., $X(v).anc_{p_i} = u_i$. For efficiency consideration, we sort values in $X(v).pos$ in their increasing order. We use $X(v).pos_i$ to denote the i-th value in $X(v).pos$ for any $1 \le i \le k$.

Example 5.3. Fig. 5 shows the H2H-Index for the road network in Fig. 1 based on the tree decomposition shown in Fig. 4. For the node $X(v_7) = \{v_7, v_6, v_8, v_{14}\}$ with $X(v_7).anc = (v_{14}, v_{13}, v_1, v_2, v_6, v_8, v_7)$, we can calculate the distance array of $X(v_7)$ as: $X(v_7).dis = (dist(v_7, v_{14}), dist(v_7, v_{13}), dist(v_7, v_1), dist(v_7, v_2), dist(v_7, v_6), dist(v_7, v_8), dist(v_7, v_7)) = (2, 3, 5, 4, 6, 2, 0)$. We can calculate the position array of $X(v_7)$ as $X(v_7).pos = (7, 5, 6, 1)$ since v_7, v_6, v_8 and v_{14} are the 7-th, 5-th, 6-th, and 1-st elements in $X(v_7).anc$ respectively. After sorting, we get $X(v_7).pos = (1, 5, 6, 7)$.

Algorithm 1 H2H-Query(T_G, H2H-Index, $q = (s, t)$)

Input: the tree decomposition T_G, the H2H-Index, and the query $q = (s, t)$;
Output: the shortest distance $dist(s, t)$.

1: $X \leftarrow$ the LCA of $X(s)$ and $X(t)$ in T_G;
2: $d \leftarrow +\infty$;
3: **for all** $i \in X.pos$ **do**
4: $\quad d \leftarrow \min(d, X(s).dis_i + X(t).dis_i)$;
5: **return** d;

The following theorem shows the space complexity of the H2H-Index. We introduce how to construct the H2H-Index efficiently in the next section.

Theorem5.4. *The space complexity of H2H-Index for network G is $O(n \cdot h)$, where h is the treeheight of the tree decomposition T_G.*

Since the treeheight h for a road network is usually small, the space $O(n \cdot h)$ is practical for large real road networks.

Query Processing using H2H-Index. We now demonstrate how to use the H2H-Index to process a shortest distance query $q = (s, t)$. The algorithm is based on the following theorem:

Theorem5.5. *Given the tree decomposition T_G of a road network G, the H2H-Index, and a query $q = (s, t)$, let X be the LCA of $X(s)$ and $X(t)$ in T_G, we have:*

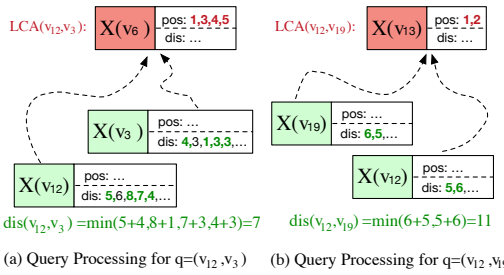
$$dist(s, t) = \min_{i \in X.pos} X(s).dis_i + X(t).dis_i$$

With Theorem 5.5, we are now ready to design the query processing algorithm. Given a shortest distance query $q = (s, t)$, the algorithm to answer q is shown in Algorithm 1. The algorithm first computes the LCA of $X(s)$ and $X(t)$ in T_G using $O(1)$ time [11] (line 1), then based on Theorem 5.5, we can answer q using only the distance array of $X(s)$ and $X(t)$ and position array of X (line 2-5). In the following, we show the time complexity of Algorithm 1.

Theorem5.6. *The time complexity of query processing using Algorithm 1 in a road network G is $O(w)$, where w is the treewidth of the tree decomposition T_G.*

It is important to note that compared to the naive query processing algorithm introduced in Section 4.3, in Algorithm 1, we can avoid looking up vertices using hash tables, and instead, we only need to sequentially scan $X.pos$, $X(s).dis$ and $X(t).dis$ to compute the result, since the positions in $X.pos$ are sorted in their increasing order. Therefore, Algorithm 1 is much more efficient than the naive algorithm. In the following, we use an example to demonstrate the query processing algorithm.

Example 5.7. Given the H2H-Index shown in Fig. 5 for the road network G, the query processing for queries $q = (v_{12}, v_3)$ and $q = (v_{12}, v_{19})$ is shown in Fig. 6 (a) and Fig. 6 (b) respectively. For query $q = (v_{12}, v_3)$, wefi rst get LCA$(v_{12}, v_3) = X(v_6)$ in T_G. Since $X(v_6).pos = (1, 3, 4, 5)$, we can compute $dist(v_{12}, v_3) = \min(X(v_{12}).dis_1 + X(v_3).dis_1, X(v_{12}).dis_3 + X(v_3).dis_3, X(v_{12}).dis_4 + X(v_3).dis_4, X(v_{12}).dis_5 + X(v_3).dis_5) = \min(5+4, 8+1, 7+3, 4+3) = 7$. Similarly, for $q = (v_{12}, v_{19})$ we can first get LCA$(v_{12}, v_{19}) = X(v_{13})$ in $T(G)$ with $X(v_{13}).pos = (1, 2)$, and then compute $dist(v_{12}, v_{19}) = \min(X(v_{12}).pos_1 + X(v_{19}).pos_1, X(v_{12}).pos_2 + X(v_{19}).pos_2) = 11$.



(a) Query Processing for q=(v_{12},v_3) (b) Query Processing for q=(v_{12},v_{19})

Figure 6: Query Processing Examples using the H2H-Index

Algorithm 2 operator $\ominus(G, v)$

Input: graph G and vertex $v \in V(G)$;
Output: the graph of $G \ominus v$.

1: $H \leftarrow G$;
2: **for all** pair of vertices $u, w \in N(v)$ **do**
3: $\quad$ **if** $(u, w) \notin E(G)$ **then**
4: $\quad\quad$ insert edge (u, w) with $\phi(u, w) \leftarrow \phi(u, v) + \phi(v, w)$ in H;
5: $\quad$ **else if** $\phi(u, v) + \phi(v, w) < \phi(u, w)$ **then**
6: $\quad\quad$ update $\phi(u, w) \leftarrow \phi(u, v) + \phi(v, w)$ in H;
7: **return** H;

6 H2H-INDEX CONSTRUCTION

In this section, we show how to construct the H2H-Index efficiently. Wefi rst introduce the distance preserved graph (DP-Graph) and show how to compute the H2H-Index using the DP-Graphs. Then we propose an improved algorithm with the assistance of the existing computed labels.

6.1 Index Construction using DP-Graph

In order to design an efficient algorithm to construct the H2H-Index, wefi rst introduce the concept of the distance preserved graph in the following.

Definition 6.1. **(DP-Graph)** Given a road network G, a graph G' is called a Distance Preserved Graph (DP-Graph) if $V(G') \subseteq V(G)$, and for any pair of vertices $u \in V(G')$ and $v \in V(G')$, we have $dist_{G'}(u, v) = dist_G(u, v)$. We use $G' \sqsubseteq G$ to denote that G' is a DP-Graph of G.

We can easily derive the following lemma:

Lemma6.2. *$G_1 \sqsubseteq G_2$ and $G_2 \sqsubseteq G_3 \Rightarrow G_1 \sqsubseteq G_3$.*

DP Vertex Elimination. Given a graph G and a vertex $v \in V(G)$, the Distance Preserved Vertex Elimination (DP Vertex Elimination) operation for v in G transform G into another graph as follows: For every pair of neighbors (u, w) of v in G, if $(u, w) \notin E(G)$, a new edge (u, w) with weight $\phi(u, w) = \phi(u, v) + \phi(v, w)$ is inserted. Otherwise, if $\phi(u, v) + \phi(v, w) < \phi(u, w)$, the weight of edge (u, w) is updated as $\phi(u, v) + \phi(v, w)$. Then v is removed. We use $G \ominus v$ to denote the DP Vertex Elimination operation for v in G. The algorithm for the $\ominus$ operator is shown in Algorithm 2. The following lemma shows that the DP Vertex Elimination operation results in a DP-Graph of the original graph G.

Lemma6.3. *$G \ominus v \sqsubseteq G$.*

DP Tree Decomposition. We introduce Distance Preserved Tree Decomposition (DP Tree Decomposition) using the DP vertex elimination operator. Generally speaking, DP tree decomposition is

Algorithm 3 DPTreeDecomposition($G(V, E)$)

Input: A road network $G(V, E)$;
Output: DP Tree decomposition T_G.

1: $H \leftarrow G$; $T_G \leftarrow \emptyset$;
2: **for** $i = 1$ **to** $|V|$ **do**
3: $v \leftarrow$ the node in H with smallest degree;
4: $X(v) \leftarrow$ the star from v to each $u \in N(v, H)$;
5: Create a node $X(v)$ in T_G;
6: $H \leftarrow H \ominus v$;
7: $\pi(v) \leftarrow i$;
8: **for all** $v \in V(G)$ **do**
9: **if** $|X(v)| > 1$ **then**
10: $u \leftarrow$ the vertex in $X(v) \setminus \{v\}$ with smallest π value;
11: Set the parent of $X(v)$ be $X(u)$ in T_G;
12: **for all** $v \in V(G)$ **do**
13: Sort vertices in $X(v)$ in decreasing order of π values;
14: $X(v).\phi_i \leftarrow \phi(v, x_i)$ where x_i is the i-th vertex in $X(v)$ for all $1 \le i \le |X(v)|$;
15: **return** T_G;

similar to the tree decomposition introduced in Section 4.2. The only difference is that in each tree node $X(v)$ in the tree decomposition T_G, we not only keep the vertices in $X(v)$, but also preserve the weights from v to vertices in $X(v)$.

The detailed algorithm for DP tree decomposition is shown in Algorithm 3. The algorithm framework is similar to the tree decomposition algorithm proposed in [7]. The detailed algorithm is provided in Section 10.2 in the Appendix. Algorithm 3 iteratively eliminates the vertex v with the smallest degree in graph H (line 3). For each v, we create a node $X(v)$ in T_G. Here, $X(v)$ is a star that contains not only the vertices $\{v\} \cup N(v, H)$, but also the edges (v, u) with weight $\phi(v, u)$ for all $u \in N(v, H)$ (line 4-5). Here, a *star* is defined as a tree of depth 1. In line 6, we eliminate v from H using the DP vertex elimination operator $\ominus$, and in line 7, we maintain the order π for all vertices. In line 8-11, we construct the tree structure by assigning the parent node for each $X(v)$. The process is the same as that in the tree decomposition algorithm proposed in [7] (Algorithm 6 in the Appendix). After this, for efficiency consideration, for each node $X(v)$ in T_G, we sort the vertices in $X(v)$ in decreasing order of their π values (line 13), and we use an array $X(v).\phi$ to preserve the weights from v to the vertices in $X(v)$ (line 14). Here, we suppose $\phi(v, v) = 0$. We use $X(v).\phi_i$ to denote the distance from v to the i-th vertex in $X(v)$. Finally, we return T_G as the tree decomposition.

According to Lemma 6.2 and Lemma 6.3, we know that after every vertex elimination in Algorithm 3, the graph H is a DP-graph of G, i.e., $H \sqsubseteq G$.

Lemma6.4. *The time complexity of Algorithm 3 is* $O(n \cdot (w^2 + \log(n)))$ *and the space complexity of Algorithm 3 is* $O(n \cdot w)$.

Example 6.5. Fig. 7 shows the DP tree decomposition T_G for the road network G in Fig. 1. To construct T_G, wefi rst select v_{12}, and create a node $X(v_{12})$ in T_G which contains a star with two edges (v_{12}, v_9) and (v_{12}, v_{11}) with $\phi(v_{12}, v_9) = 2$ and $\phi(v_{12}, v_{11}) = 1$. We eliminate v_{12} by adding an edge (v_9, v_{11}) with $\phi(v_9, v_{11}) = 3$, and set $\phi(v_{12}) = 1$. The process stops when all vertices are eliminated. For node $X(v_{12})$, after sorting the vertices in $X(v_{12})$ in decreasing order of their π values, we obtain the order v_{11}, v_9, v_{12} with $\phi(v_{12}, v_{11}) = 1$, $\phi(v_{12}, v_9) = 2$, and $\phi(v_{12}, v_{12}) = 0$. Therefore, $X(v_{12}).\phi = (1, 2, 0)$ as shown in Fig. 7.

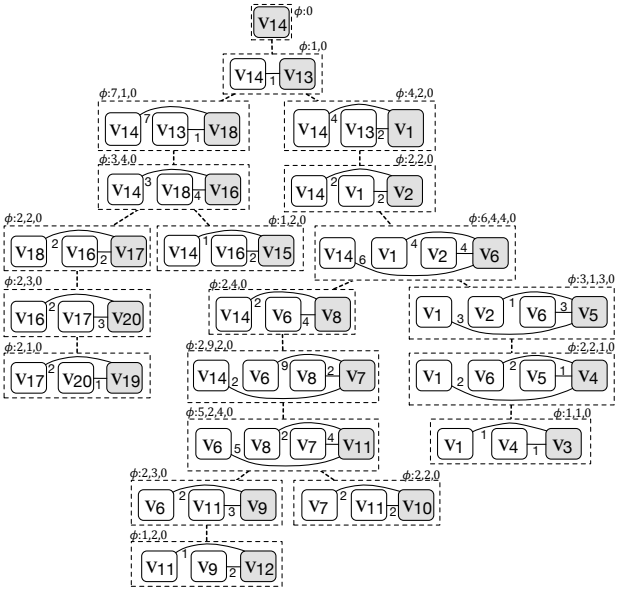
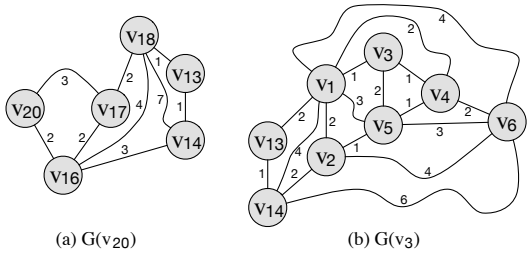


Figure 7: Distance Preserved Tree Decomposition T_G



(a) G(v_{20}) (b) G(v_3)

Figure 8: DP-Graphs $G(v_{20})$ and $G(v_3)$

Given the DP tree decomposition T_G, for each $v \in V(G)$, we define a special graph $G(v)$ as follows.

Definition 6.6. Given a DP tree decomposition T_G for the road network G, for each $v \in V(G)$, the graph $G(v)$ is defined as the union of all the stars for nodes in the path from $X(v)$ to the root of T_G.

The number of edges in each $G(v)$ is at most $w \times h$ since each star has at most w edges and the number of nodes from $X(v)$ to the root of T_G is at most h. Therefore, we can use $O(w \cdot h)$ time to construct $G(v)$ for each vertex v. In the following, we provide an example to illustrate $G(v)$.

Example 6.7. Given the DP tree decomposition T_G in Fig. 7, for vertex v_{20}, the path from $X(v_{20})$ to the root $X(v_{14})$ consists of 6 nodes $X(v_{20}), X(v_{17}), X(v_{16}), X(v_{18}), X(v_{13})$, and $X(v_{14})$. We merge the stars in the 6 nodes and obtain the graph $G(v_{20})$ shown in Fig. 8 (a) with 6 vertices and 9 edges. Similarly, we can obtain graph $G(v_3)$ as shown in Fig. 8 (b).

The following lemma shows the property that $G(v)$ is a DP-graph of G for any $v \in V(G)$:

Lemma6.8. $G(v) \sqsubseteq G$ *for any* $v \in V(G)$.

Algorithm 4 H2H-Naive(G)

Input: The road network $G(V, E)$;
Output: The H2H-Index.

1: $T_G \leftarrow$ DPTreeDecomposition(G);
2: **for all** $v \in V(G)$ **do**
3: $X(v).pos_i \leftarrow$ the position of v_i in $X(v).anc$ where v_i is the i-th vertex in $X(v)$ for all $1 \le i \le |X(v)|$;
4: Construct $G(v)$ according to Definition 6.6;
5: Compute $dist_{G(v)}(v, u)$ for each vertex u in $G(v)$;
6: **for** $i = 1$ **to** $|X(v).anc|$ **do**
7: $X(v).dis_i \leftarrow dist_{G(v)}(v, X(v).anc_i)$;
8: **return** $X(v).dis$ and $X(v).pos$ for all $v \in V(G)$;

The Index Construction Algorithm. With Definition 6.6 and Lemma 6.8, we are now ready to design the index construction algorithm. The pseudocode of the algorithm is shown in Algorithm 4. Recall that for each $v \in V(G)$, the H2H-Index consists of two components, the position array $X(v).pos$ and the distance array $X(v).dis$. After computing the DP tree decomposition T_G (line 1), we construct the two arrays for each $v \in V(G)$ (line 2). Wefi rst probe each vertex v_i of $X(v)$ in the sorted order in T_G, and assign $X(v).pos_i$ to be the position of v_i in the ancestor array $X(v).anc$ in T_G (line 3). Then, we construct $G(v)$ according to Definition 6.6 (line 4). After this, we can compute the single source shortest distances $dist_{G(v)}(v, u)$ from v to all vertices u in $G(v)$ using the Dijkstra's Algorithm (line 4). By Definition 6.6 and Lemma 6.8, for each i from 1 to $|X(v).anc|$, we can assign $X(v).dis_i$ as the shortest distance from v to $X(v).anc_i$ in $G(v)$ (line 6-7). Finally, the distance array and position array for all vertices in G are returned as the H2H-Index.

Theorem6.9. *The time complexity of Algorithm 4 to construct the H2H-Index is* $O(n \cdot (h \cdot w \cdot \log(h) + \log(n)))$.

In the following, we provide an example to illustrate Algorithm 4:

Example 6.10. Suppose the DP tree decomposition T_G in Fig. 7 is computed for road network G in Fig. 1. We show how to compute $X(v_{20}).pos$ and $X(v_{20}).dis$. From T_G we know that $X(v_{20}) = (v_{16}, v_{17}, v_{20})$ and $X(v_{20}).anc = (v_{14}, v_{13}, v_{18}, v_{16}, v_{17}, v_{20})$. The positions of v_{16}, v_{17}, and v_{20} in $X(v_{20}).anc$ are 4, 5, and 6, respectively. Therefore, we have $X(v_{20}).pos = (4, 5, 6)$. The constructed $G(v_{20})$ is shown in Fig. 8 (a). Using the Dijkstra's Algorithm, we can compute the shortest distance from v_{20} to all vertices in $X(v_{20}).anc$ in graph $G(v_{20})$ as $dist(v_{20}, v_{14}) = 5$, $dist(v_{20}, v_{13}) = 6$, $dist(v_{20}, v_{18}) = 5$, $dist(v_{20}, v_{16}) = 2$, $dist(v_{20}, v_{17}) = 3$, $dist(v_{20}, v_{20}) = 0$. Therefore, we can get $X(v_{20}).dis = (5, 6, 5, 2, 3, 0)$.

6.2 Leveraging Partial Labels in Indexing

In this subsection, we show how to further optimize the index construction algorithm in Algorithm 4. We consider two aspects to improve the algorithm. First, for each vertex v, we need to construct the DP-graph $G(v)$ and compute the single source shortest distances from v to all other vertices in $G(v)$, which is costly. We consider how to compute the labels without explicitly building the DP-graph $G(v)$. Second, the vertex labels are computed independently in Algorithm 4. To save computational cost, we will consider cost sharing when computing the labels for different vertices. Note that the graph $G(v)$ is a supergraph of $G(u)$ for any $u \in X(v).anc$. This means that we can reuse the label information in $X(u)$ when computing the label for $X(v)$. To do this, we need to compute the vertex labels in a top-down manner in T_G. Wefi rst introduce the following lemma:

Algorithm 5 H2H(G)

Input: The road network $G(V, E)$;
Output: The H2H-Index.

1: $T_G \leftarrow$ DPTreeDecomposition(G);
2: **for** $X(v) \in V(T_G)$ in a top-down manner **do**
3: Suppose $X(v) = (x_1, x_2, \ldots, x_{|X(v)|})$;
4: **for** $i = 1$ **to** $|X(v)|$ **do**
5: $X(v).pos_i \leftarrow$ the position of x_i in $X(v).anc$;
6: **for** $i = 1$ **to** $|X(v).anc| - 1$ **do**
7: $X(v).dis_i \leftarrow +\infty$;
8: **for** $j = 1$ **to** $|X(v)| - 1$ **do**
9: **if** $X(v).pos_j > i$ **then** $d \leftarrow X(x_j).dis_i$;
10: **else** $d \leftarrow X(X(v).anc_i).dis_{X(v).pos_j}$;
11: $X(v).dis_i = \min\{X(v).dis_i, X(v).\phi_j + d\}$;
12: $X(v).dis_{|X(v).anc|} \leftarrow 0$;
13: **return** $X(v).dis$ and $X(v).pos$ for all $v \in V(G)$;

Lemma6.11. *For any* $1 \le i < |X(v).anc|$, *we have*

$$X(v).dis_i = \min_{1 \le j < |X(v)|} X(v).\phi_j + dist(x_j, X(v).anc_i)$$

Here, x_j *is the j-th vertex in* $X(v)$.

Based on Lemma 6.11, we need to compute $dist(x_j, X(v).anc_i)$ for each $x_j \in X(v)$ and $1 \le i < |X(v).anc|$. In order to compute $dist(x_j, X(v).anc_i)$ without exploring the graph $G(v)$, we consider two cases:

- *Case 1:* $X(v).pos_j > i$. In this case, $X(X(v).anc_i)$ is an ancestor of $X(x_j)$ in T_G. Therefore, we have

$$dist(x_j, X(v).anc_i) = dist(x_j, X(x_j).anc_i) = X(x_j).dis_i$$

  In addition, since we traverse $X(v)$ in a top-down manner, $X(x_j).dis_i$ has been computed when computing the label of $X(v)$. Therefore, $X(x_j).dis_i$ can be directly used as $dist(x_j, X(v).anc_i)$.
- *Case 2:* $X(v).pos_j \le i$. In this case, $X(x_j)$ is an ancestor of $X(X(v).anc_i)$ in T_G. Therefore, we have

$$dist(x_j, X(v).anc_i) = X(X(v).anc_i).dis_{X(v).pos_j}$$

  Similar to case 1, since we traverse $X(v)$ in a top-down manner, $X(X(v).anc_i).dis_{X(v).pos_j}$ has been computed when computing the label of $X(v)$. Therefore, $X(X(v).anc_i).dis_{X(v).pos_j}$ can be directly used as $dist(x_j, X(v).anc_i)$.

Example 6.12. Given the DP tree decomposition T_G in Fig. 7, for vertex v_4, we have $X(v_4) = (v_1, v_6, v_5, v_4)$, $X(v_4).\phi = (2, 2, 1, 0)$ and $X(v_4).anc = (v_{14}, v_{13}, v_1, v_2, v_6, v_5, v_4)$. We can compute that $X(v_4).pos = (3, 5, 6, 7)$. Now let us compute $X(v_4).dis_4$, i.e., the shortest distance from v_4 to v_2. Suppose the distance arrays for all ancestors of $X(v_4)$ in T_G has been computed. Based on Lemma 6.11, we can get that $X(v_4).dis_4 = \min\{2 + dist(v_1, v_2), 2 + dist(v_6, v_2), 1 + dist(v_5, v_2)\}$. To compute $dist(v_1, v_2)$, we know $X(v_4).pos_1 = 3 < 4$, which is case 2. Therefore, we can derive that $dist(v_1, v_2) = X(X(v_4).anc_4).dis_{X(v_4).pos_1} = X(v_2).dis_3 = 2$. To compute $dist(v_6, v_2)$, we know $X(v_4).pos_2 = 5 > 4$, which is case 1. Therefore, we can derive that $dist(v_6, v_2) = X(v_6).dis_4 = 4$. Similarly, to compute $dist(v_5, v_2)$, we know $X(v_4).pos_3 = 6 > 4$, which is case 1. Therefore, we can get $dist(v_5, v_2) = X(v_5).dis_4 = 1$. Consequently, we can get $X(v_4).dis_4 = \min\{2 + 2, 2 + 4, 1 + 1\} = 2$.

$X(v_4).anc$	v_{14}	v_{13}	v_1	v_2	v_6	v_5	$X(v_4).\phi$
v_{14}	0	1	3	$X(v_2).dis_1$:2	6	3	-
v_{13}	1	0	2	$X(v_2).dis_2$:3	6	4	-
v_1	3	2	0	$X(v_2).dis_3$:2	4	3	ϕ_1:2
v_2	2	3	2	$X(v_2).dis_4$:0	4	1	-
v_6	6	6	4	$X(v_6).dis_4$:4	0	3	ϕ_2:2
v_5	3	4	3	$X(v_5).dis_4$:1	3	0	ϕ_3:1
$X(v_4).dis$:	4	4	2	2	2	1	-

Figure 9: Process of Calculating $X(v_4).dis$

With Lemma 6.11, we are ready to design the optimized algorithm to construct the H2H-Index. The pseudocode of the algorithm is shown in Algorithm 5. After computing the DP tree decomposition T_G, we examine all nodes $X(v)$ in T_G in a top-down manner (line 2). Supposing x_i is the i-th vertex in $X(v)$ (line 3), wefi rst compute the position array $X(v).pos$ by its definition (line 4-5). Then, for each $1 \leq i < |X(v).anc|$, we compute $X(v).dis_i$ (line 6). We use Lemma 6.11 to compute $X(v).dis_i$ and consider the two cases based on a comparison between $X(v).pos_j$ and i (line 7-11). We set the last value in the distance array to be 0 since it is the distance from v to itself (line 12). Finally, we return the distance and position arrays for all nodes as the H2H-Index (line 13). We can derive the following theorem:

Theorem6.13. *The time complexity of Algorithm 5 to construct the H2H-Index is* $O(n \cdot (\log(n) + h \cdot w))$.

Algorithm 5 improves Algorithm 4 in two aspects. First, it totally avoids materializing the DP-graphs $G(v)$ for each $v \in V(G)$. Second, by re-using the partial labels, Algorithm 5 reduce the $\log(h)$ factor compared to Algorithm 4.

Example 6.14. We show how to compute $X(v_4).dis$ using Algorithm 5 for the road network G in Fig. 1. The DP tree decomposition T_G is shown in Fig. 7 with $X(v_4) = (v_1, v_6, v_5, v_4)$. First, we can get $X(v_4).anc = (v_{14}, v_{13}, v_1, v_2, v_6, v_5, v_4)$ and $X(v_4).pos = (3, 5, 6, 7)$. In Fig. 9, thefirst 7 lines and 7 columns show a 6×6 matrix M. For each $1 \leq j \leq 6$ and $1 \leq i \leq 6$, if $j \leq i$, we set $M_{j,i}$ to be $X(X(v_4).anc_j).dis_i$. Otherwise, we set $M_{j,i}$ to be $X(X(v_4).anc_i).dis_j$. We also show $X(v_4).\phi = (2, 2, 1)$ for vertices v_1, v_6, and v_5 in $X(v_4)$. As shown in Example 6.12, we can compute $X(v_4).dis_4 = \min\{X(v_4).\phi_1 + X(v_2).dis_3,\ X(v_4).\phi_2 + X(v_6).dis_4,\ X(v_4).\phi_3 + X(v_5).dis_4\} = \min\{4, 6, 2\} = 2$. The cells that involve the calculation have been marked gray. In the same way, we can calculate $X(v_4).dis_i$ for $i \in \{1, 2, 3, 5, 6\}$. We also know $X(v_4).dis_7 = 0$. Therefore, we get $X(v_4).dis = (4, 4, 2, 2, 2, 1, 0)$.

7 PERFORMANCE STUDIES

Algorithms. We compare our proposed algorithm with the state-of-the-art algorithms for shortest distance query processing in road networks. We implement and compare six algorithms:

- CH: Contraction Hierarchy [16];
- AH: Arterial Hierarchy [36];
- HL: Hub-based Labeling [2];
- PHL: Pruned Highway Labeling [3];
- H2H-Naive: H2H-Index Construction using Algorithm 4.
- H2H: H2H Query Processing (Algorithm 1) and Index Construction (Algorithm 5);

All algorithms are implemented in C++ and compiled with GNU GCC 4.4.7. All experiments are conducted on a machine with an Intel Xeon 2.8GHz CPU and 256 GB main memory running Linux (Red Hat Linux 4.4.7, 64bit).

Name	Region	# Vertices n	# Edges m	h	w	$\log(n)/w$	$\sqrt{n}/w$	td(sec)
NY	New York City	264,346	733,846	320	131	0.041	3.92	5.6
COL	Colorado	435,666	1,057,066	377	168	0.033	3.93	5.9
FLA	Florida	1,070,376	2,712,798	411	155	0.039	6.67	15.2
CAL	California	1,890,815	4,657,742	679	291	0.022	4.73	31.3
E-US	Eastern US	3,598,623	8,778,114	918	391	0.017	4.85	70
W-US	Western US	6,262,104	15,248,146	1032	386	0.018	6.48	136
C-US	Central US	14,081,816	34,292,496	1816	687	0.010	5.46	1161
US	United States	23,947,347	58,333,344	1776	762	0.010	6.42	1298

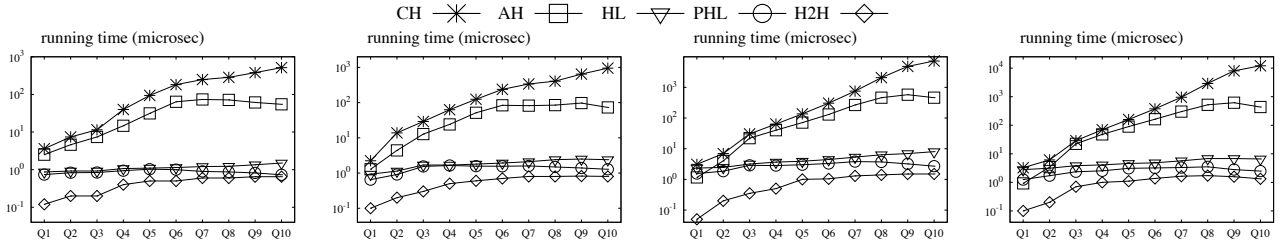
Table 2: Datasets Used in the Experiments

Datasets. We use ten publicly available real road networks in the US, downloaded from DIMACS[1]. Table 2 provides the data details in which the largest dataset is the whole road network in the US. We use distance as the edge weight for each dataset. We show the treeheight h and treewidth w obtained by Algorithm 3, and the time td for tree decomposition. To demonstrate how w grows with n, we also show the values of $\log(n)/w$ and $\sqrt{n}/w$. As shown in Table 2, for each dataset, the treeheight and treewidth are much smaller than the number of vertices in the dataset. When n increases, $\log(n)/w$ has a decreasing trend and $\sqrt{n}/w$ has an increasing trend, i.e., the growth rate of w is between $\log(n)$ and $\sqrt{n}$. Therefore, w grows sub-linearly with the size of the network.

Exp-1: Varying Query Distance. In this experiment, we test the query efficiency of the algorithms by varying the distance between the source and target vertices in the query. Specifically, for each dataset, we generate 10 groups of queries $Q_1, Q_2, \ldots, Q_{10}$ as follows: we set l_{min} to be 1 kilometer, and set l_{max} to be the maximum distance of any pair of vertices in the map. Let $x = (l_{max}/l_{min})^{1/10}$. For each $1 \leq i \leq 10$, we generate 10,000 queries to form Q_i, in which the distance of the source and target vertices for each query fall in the range $(l_{min} \times x^{i-1}, l_{min} \times x^i]$. For each algorithm, we record the average query processing time for the 10,000 queries in the corresponding query set.

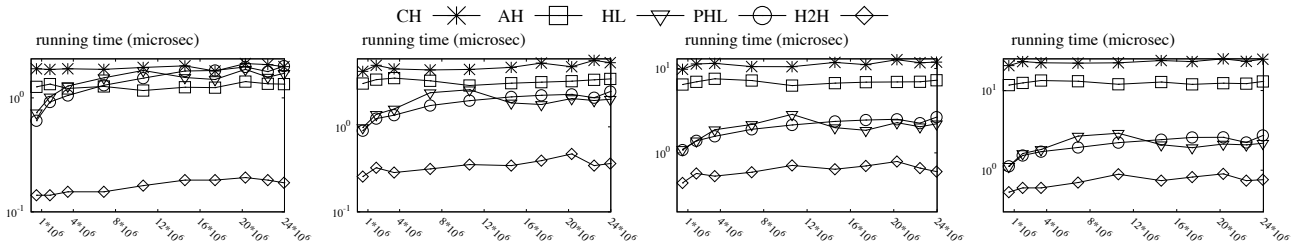
The experimental results when varying the query from Q_1 to Q_{10} are shown in Fig. 10 (a) - (d). We show the results for 4 of the 8 datasets. The results for the other 4 datasets are provided in the Appendix. We make the following observations. First, when the distance increases, the query processing time tend to increase in all datasets for all thefi ve algorithms CH, AH, HL, PHL, and H2H. This is because when the distance between two vertices s and t increases, more edges are involved in the shortest path between s and t. Therefore, longer time is needed to answer the shortest distance query (s, t). Second, when the distance between s and t is relatively large, AH is 3-5 times faster than CH, HL is more than an order of magnitude faster than AH, PHL is 2-3 times faster than HL, and our algorithm H2H is 1.5-2 times faster than PHL. As analyzed in Section 4, the hop-based solutions HL and PHL are much faster than the hierarchy-based solutions CH and AH when $dist(s, t)$ is large, since it can avoid probing the graphs by using the labels directly. Our solution H2H is faster than the hop-based solution since H2H can further reduce the processing cost by visiting only a subset of the labels. Third, when the distance between s and t is relatively small, the two hop-based solutions HL and PHL have similar performance and the two hierarchy-based solutions CH and AH have similar performance. In some datasets such as *C-US* (Fig. 10 (c)) and *US* (Fig. 10 (d)), the hierarchy-based solutions CH and AH

[1] http://www.dis.uniroma1.it/challenge9/download.shtml



(a) *COL* (b) *CAL* (c) *C-US* (d) *US*

Figure 10: Query Processing Time (Varying Query Distance)



(a) Q_1 (b) Q_2 (c) Q_3 (d) Q_4

Figure 11: Query Processing Time (Varying Dataset Size)

can be even faster than the hop-based solutions HL and PHL. This result is consistent with the analysis in Section 4. Remarkably, in this case, our algorithm H2H can achieve a speed-up of more than an order of magnitude compared to all the other approaches. This is because when the distance s and t is small, the vertex cut of s and t in the road network becomes small. Consequently, H2H only needs to visit a small portion of the labels for s and t to answer a query. This result demonstrates the advantage of our approach.

Exp-2: Varying Dataset Size. In this experiment, we test the query scalability when the size of the dataset increases. To do this, we divide the map of the whole US into 10×10 grids. We select a 1×1 grid in the middle to generate 10 groups of queries $Q_1, Q_2, \ldots, Q_{10}$ using the same method in Exp-1 each of which contains $10,000$ shortest distance queries. Then we generate 10 road networks using the 1×1, 2×2, $\ldots$, 10×10 grids in the middle of the map and denote them as $G_1, G_2, \ldots, G_{10}$ respectively. G_1 is the selected 1×1 grid to generate the queries, and G_{10} is the whole road network of US. We have $G_1 \subset G_2 \subset \ldots \subset G_{10}$. We test the query processing time for each algorithm on each $G_i (1 \leq i \leq 10)$. For each query group $Q_i (1 \leq i \leq 10)$, we record the average processing time of the $10,000$ queries in the group.

The experimental results for Q_1 to Q_4 when varying the dataset size are shown in Fig. 11 (a) to (d) respectively. The results for Q_5 to Q_{10} are shown in the Appendix. The x-axis for eachfi gure is the number of vertices for each dataset. From the experimental results, we make the following observations. First, the query processing time for the hop-based solutions HL and PHL has an obvious increasing trend when the data size increases. The query processing time for CH, AH, and H2H is relatively more scale-independent to the data size. This is because the size of the vertex cut for s and t does not significantly increase with the increasing of the dataset size. Second, the gap between H2H and the hop-based solutions HL and PHL increases when the size of the network increases for all queries. This is because when the size of the network is larger, H2H can skip visiting more vertices in the vertex labels of s and t. Third, when the $dist(s, t)$ increases, the gap between H2H and the hop-based solutions HL and PHL decreases. This is because when $dist(s, t)$ increases, the size of the vertex cut for s and t increases while the processing time of HL and PHL is independent of the size of the vertex cut. H2H performs the best in all cases.

Exp-3: Indexing Time. In this experiment, we test the indexing time for CH, AH, HL, PHL, H2H (Algorithm 5), and H2H-Naive (Algorithm 4). Note that the indexing time for our algorithms includes both the time for obtaining the tree decomposition and the time for constructing the H2H-Index. The experimental results for the 8 road networks are shown in Fig. 12 (a). From the experimental results, we can see that when the size of the networks increases, the indexing time for all algorithms will increase. CH is most efficient in indexing. Our algorithm H2H is the second most efficient algorithm in indexing. HL is slower than H2H. AH is 2-10 times slower than HL. H2H-Naive is more than two orders of magnitude slower than H2H. For the four datasets *E-US*, *W-US*, *C-US*, and *US*, H2H-Naive cannot terminate within 30 hours and therefore we denote the time as INF. This shows the advantages of using the partial labels in the H2H algorithm. We also show the tree decomposition time separately in Table 2. From the results, we can see that the time for constructing the H2H-Index only takes up around 1/3 of the total indexing time for H2H.

Exp-4: Index Size. In this experiment, we test the index size for thefi ve approaches CH, AH, HL, PHL, and H2H. The experimental results for the 8 road networks are shown in Fig. 12 (b). According to thefi gure, we can see that when the size of the network increases, the index size for all approaches will increase. CH has the smallest index size, followed by AH. The index sizes for the three approaches HL, PHL, and H2H are comparable. This is because they all generate the vertex labels and the label should be large enough to guarantee that the query can be answered correctly using only the information from the labels for s and t for each query $q = (s, t)$. The index size of H2H is 1.5-2 times smaller than HL, because in H2H, we only need to maintain the position array and distance array without

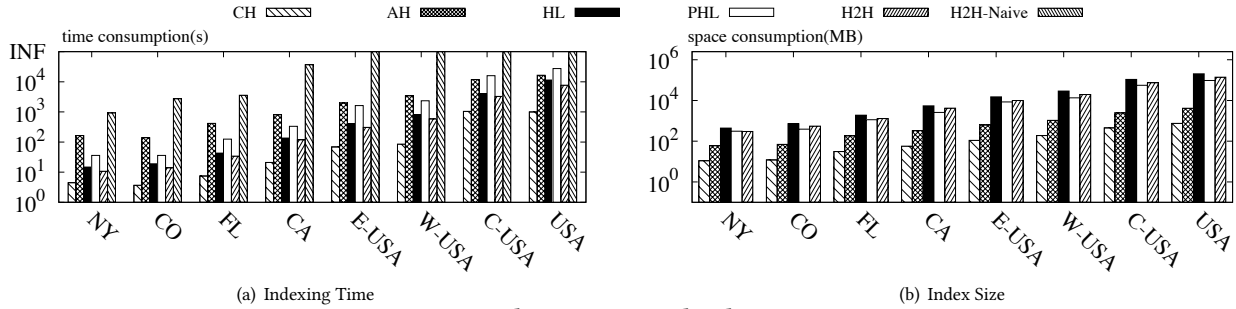


Figure 12: Indexing Time and Index Size Testing

keeping the vertex information, which make it a more compact index compared to HL.

8 RELATED WORK

Shortest distance in road networks. Shortest path/distance queries are one of the most important types of queries in road networks. When answering shortest path/distance queries in a road network, both topological information and coordinate information are used to accelerate query processing. Existing solutions mainly focus on constructing an effective index to answer shortest path/distance queries. For example, ALT [17] accelerates the A* search by the pre-computation of some shortest distances. The hierarchical technique is an important category for this problem. HiTi [21] divides the graph and constructs a hierarchical structure to accelerate query processing. Highway Hierarchies [28] and Contraction Hierarchies [16] are good methods for pruning the search space. The contraction hierarchies approach relies on a total order of vertices. By adding shortcuts from low-ordered vertices to high-ordered vertices, it constructs a toplogical order of a graph and thus reduces the search space. The hop-based labeling approach is another important category for shortest distance queries in road networks. Of these, hub-based labeling algorithms are proposed in [1, 2] based on the 2-hop index; highway-based labeling [3] is another efficient hop-based algorithm based on highway decomposition, which divides the road network by shortest paths. Given a shortest distance query, hop-based labeling can answer the query using labels only, rather than searching the graph. More details of the hierarchical-based approach and the hop-based approach can be found in Section 3. Transit Node Routing [8], TRANSIT [10], and Arterial Hierarchy [36] are algorithms which integrate coordinate information. They divide the map into grids. Transit Node Routing creates an index which contains all the distances between different grids, TRANSIT precomputes the distances for each vertex to its closest transit node in the grid, and Arterial Hierarchy is an improved algorithm of the contraction hierarchies algorithm. Other works assume a road network as a planar graph without a negative weight, and use the Quadtree and path oracle [24, 27, 29, 30] on planar graphs to solve the shortest path/distance problem. A survey can be found in [9] and some experimental results for different algorithms to answer the shortest path and distance queries in road networks are reported in [34].

Tree decomposition and treewidth. Tree decomposition, which is originally introduced by Halin [18] and rediscovered by Robertson and Seymour [26], is a mapping of a graph into a tree that can be used to define the treewidth of the graph and speed up solving certain computational problems on the graph. Many algorithmic problems, such as maximum independent set and Hamiltonian circuits that are NP-complete for arbitrary graphs, may be solved efficiently by dynamic programming for graphs of bounded treewidth, using the tree-decompositions of these graphs. An introductory survey by Bodlaender can be found in [12]. It is NP-complete to determine whether a given graph G has treewidth at most a given variable [6]. However, when the treewidth of a graph is afi xed constant, a tree decomposition for the graph with the minimum treewidth can be constructed in a time linear to the size of the graph but exponential to the treewidth [13]. This algorithm is only practical when the treewidth is very small (e.g., smaller than 10). Many heuristics are proposed to determine the treewidth of a graph, but unfortunately few of them can deal with large graphs (e.g., graphs with more than 1,000 vertices) [22]. Because of the big challenges in computing the optimal tree decomposition, in this paper, we adopt a suboptimal algorithm introduced in [14]. The details of the algorithm can be found in Section 10.2. The algorithm does not guarantee to compute a treewidth with a bounded approximation ratio. Nevertheless, since the time complexities of both our index construction and query processing algorithms are polynomial to the treewidth of the tree decomposition, the algorithm in [14] is practically applicable for our problem in large real-world road networks.

9 CONCLUSION

In this paper, we study the problem of distance query processing in road networks. We propose a novel hierarchical 2-hop index (H2H-Index) which can overcome the drawbacks of the hierarchy-based solution and hop-based solution. We can achieve $O(w)$ query processing time and $O(n \cdot h)$ index size where w and h are the treewidth and treeheight for the tree decomposition of the road network which are small in practice, and n is the number of vertices in the road network. We propose an efficient algorithm using distance preserved graphs and partial labels which constructs the H2H-Index in $O(n \cdot (log(n) + w \cdot h))$ time. The experimental results demonstrate that our approach can achieve a speedup of an order of magnitude in query processing compared to the state-of-the-art while consuming comparable indexing time and index size.

Acknowledgements. Lu Qin is supported by DP160101513. Lijun Chang is supported by DE150100563 and DP160101513. Xuemin Lin is supported by NSFC61672235, DP170101628, and DP180103096. Ying Zhang is supported by FT170100128 and DP180103096. Qing Zhu is supported by the grants of the National Natural Science Foundation of China No. 61070053 and the teaching reform grants of Renmin University of China No.2972163992.

10 APPENDIX

10.1 More Existing Solutions

Arterial Hierarchy (AH). The Arterial Hierarchy (AH) approach is proposed by Zhu et al. [36]. The AH algorithm is inspired by CH, but it produces shortcuts by imposing the 4×4 grids on the network by exploiting some 2-dimensional spatial properties in the network. Query processing of AH is similar to that of CH which adopts the bidirectional Dijkstra's algorithm by enforcing the expansion direction to be only from lower-ranked vertices to higher-ranked vertices. AH is shown to be more efficient than CH when answering shortest distance queries since both network information and spatial information are utilized in AH. More details of the AH algorithm can be found in [36].

Hub-based Labeling (HL). One of the most efficient hop-based solutions for shortest distance query processing in a road network is hub-based labeling, proposed by Abraham et al. [1, 2]. The algorithm aims to use the hub vertices as the label of a vertex. The authors show that for road networks, the effective label of a vertex $v \in V(G)$ can be obtained from the upward search space of a CH query by considering u as a source vertex. The quality of the labeling depends on the vertex order for CH. The authors observe that a good order can be obtained by greedily picking the vertices that hit the highest number of shortest paths.

Pruned Highway Labeling (PHL). The Pruned Highway labeling approach is proposed by Akiba et al. [3]. In this work, instead of using hubs as the labels for each vertex, the algorithm uses paths as the labels, aiming to encode more information in each label. In the indexing phase, the algorithm decomposes the road network into disjoint shortest paths, and then computes a label for each vertex v which contains the distance from v to vertices in a small subset of the computed paths. This guarantees that any shortest distance query $q = (s, t)$ can be answered by hopping from s to a path in $L(s) \cap L(t)$ and then hopping from the path to t. The pruned labeling technique [4] is used in indexing to reduce the label size.

10.2 Tree Decomposition Computation

We briefly introduce the tree decomposition algorithm which is introduced in [14]. The pseudocode of the algorithm is shown in Algorithm 6. The algorithm consists of two phases.

- In thefi rst phase (line 2-8), all nodes in T_G are created. We create afi ll-in graph H which is initialized as G. Each time, we select a vertex v in H with the smallest degree and create a node in T_G consisting of v and all its neighbors in H (line 3-5). We add edges into H to make all the neighbors of v in H connected to each other and then remove v and its adjacent edges from H (line 6-7). We use π to denote a total order of vertices removed from H.
- In the second phase (line 9-12), all edges in T_G are created. For every node $X(v)$ of size larger than 1, i.e., $X(v)$ is a non-root node, wefi nd a vertex u in $X(v) \setminus \{v\}$ with the smallest π value (line 11) and simply assign the parent of $X(v)$ to be $X(u)$ (line 12).

Note that the algorithm executes in n iterations, and in each iteration, a vertex v is removed, edges among v's neighbors are created, and v along with its neighbors form a node $X(v)$ in the tree decomposition. When the algorithm terminates, n nodes in tree decomposition are created, and each of them corresponds to a vertex in the original graph G. Therefore, there is a one-to-one mapping from $V(G)$ to $V(T_G)$. The time complexity of Algorithm 6 is $O(n \cdot (w^2 + \log(n)))$. In the following, we use an example to demonstrate the process of tree decomposition.

Example 10.1. Suppose we compute the tree decomposition for the road network shown in Fig. 1 using Algorithm 6. In thefirst phase, wefi rst choose v_{12}, and create a node $X(v_{12}) = \{v_{12}, v_9, v_{11}\}$ including v_{12} and its neighbors. We then create an edge between v_9 and v_{11} and remove v_{12}. We then choose v_9. Now v_9 has an original neighbor v_6 and a newly added neighbor v_{11}. Therefore, we create a node $X(v_9) = \{v_9, v_6, v_{11}\}$, add an edge between v_6 and v_{11} and then remove v_9. The process continues until all vertices are removed. In the second phase, we create edges in T_G. For the node $X(v_{12})$, the node with the smallest order in $X(v_{12}) \setminus \{v_{12}\}$ is v_9. Therefore, we assign the parent of $X(v_{12})$ to be $X(v_9)$. Thefinal tree decomposition result is shown in Fig. 4.

Algorithm 6 TreeDecomposition($G(V, E)$)

Input: A road network $G(V, E)$;
Output: Tree decomposition T_G.

1: $H \leftarrow G$; $T_G \leftarrow \emptyset$;
2: **for** $i = 1$ **to** $|V|$ **do**
3: $v \leftarrow$ the node in H with smallest degree;
4: $X(v) \leftarrow \{v\} \cup N(v, H)$;
5: Create a node $X(v)$ in T_G;
6: Add edges to H to make every pair of vertices in $N(v, H)$ connected to each other in H;
7: Remove v and its adjacent edges from H;
8: $\pi(v) \leftarrow i$;
9: **for all** $v \in V(G)$ **do**
10: **if** $|X(v)| > 1$ **then**
11: $u \leftarrow$ the vertex in $X(v) \setminus \{v\}$ with smallest π value;
12: Set the parent of $X(v)$ be $X(u)$ in T_G;
13: **return** T_G;

10.3 Proofs

Proof of Property 2. According to condition 3 of Definition 4.3, the nodes containing v form a connected subtree $T(v)$ in T_G, and the root of the subtree is $X(v)$ according to Definition 4.3. Since $X(v)$ is an ancestor of any non-root node in $T(v)$, $X(v)$ is an ancestor of $X(u)$ in $T(v)$. As a result, $X(v)$ is an ancestor of $X(u)$ in T_G. □

Proof of Theorem 5.4. The space complexity of the H2H-Index depends on the size of the two components, distance array and position array. For each node $X(v) \in T_G$, the size of $X(v).dis$ is no larger than l since the path from the root to $X(v)$ in T_G is no larger than l. The size of $X(v).pos$ is no larger than h since $X(v).pos$ is a subset of $X(v).anc$, and the size of $X(v).anc$ is no larger than h. Consequently, the size of the H2H-Index is bounded by $O(n \cdot h)$. □

Proof of Theorem 5.5. Based on the definition of the H2H-Index, suppose $X = \{u_1, u_2, \ldots, u_k\}$ and elements in X are sorted in increasing order of their positions in $X.anc$. For any decedent $X(v)$ of X in $T(G)$, $X.anc$ is a prefix of $X(v).anc$, i.e., $X.anc_i = X(v).anc_i$ for any $1 \le i \le |X.anc|$. Therefore, we have $dist(v, X.anc_i) = dist(v, X(v).anc_i) = X(v).dis_i$ for any $1 \le i \le |X.anc|$. According to Theorem 4.9 and Property 1, we know that $dist(s, t) = \min_{v \in X} dist(s, v) + dist(v, t)$. Since both $X(s)$ and $X(t)$ are decedents of X in T_G, we can derive that:

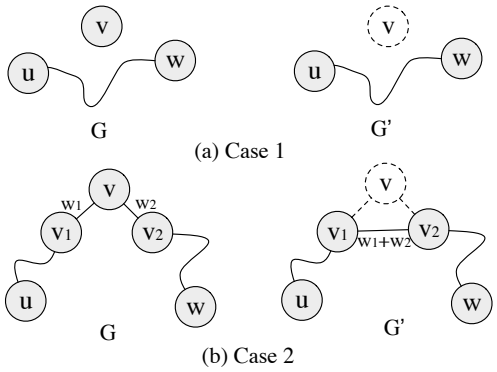


Figure 13: Proof of Lemma 6.3

$$\begin{aligned} dist(s,t) &= \min_{v \in X} dist(s,v) + dist(t,v) \\ &= \min_{i \in X.pos} dist(s, X.anc_i) + dist(t, X.anc_i) \\ &= \min_{i \in X.pos} dist(s, X(s).anc_i) + dist(t, X(t).anc_i) \\ &= \min_{i \in X.pos} X(s).dis_i + X(t).dis_i \end{aligned}$$

Therefore, the theorem holds. □

Proof of Theorem 5.6. First, line 1 of Algorithm 1 costs $O(1)$ time according to [11]. Second, lines 2-4 of Algorithm 1 costs $O(w)$ time since $|X.pos| \leq w+1$ in the tree decomposition T_G. Therefore, the overall time complexity of Algorithm 1 is $O(w)$. □

Proof of Lemma 6.2. For any pair of vertices $u, v \in V(G_1)$, since $G_1 \sqsubseteq G_2$, we have $dist_{G_1}(u,v) = dist_{G_2}(u,v)$. Since $G_2 \sqsubseteq G_3$, we further have $dist_{G_2}(u,v) = dist_{G_3}(u,v)$. Consequently, $dist_{G_1}(u,v) = dist_{G_3}(u,v)$. Therefore, the lemma holds. □

Proof of Lemma 6.3. Let G' be $G \ominus v$. For any $u \in V(G')$ and $w \in V(G')$, we consider two cases:

- *Case 1: the shortest path from u to w in G does not pass through v.* In this case, we have $dist_G(u,w) = dist_{G'}(u,w)$ because the shortest path from u to w in G is also a shortest path from u to w in G'. Therefore, the distance from u to w is preserved. This case is illustrated in Fig. 13 (a).
- *Case 2: the shortest path from u to w in G passes through v.* In this case, suppose the shortest path from u to w in G is $(u, \ldots, v_1, v, v_2, \ldots, w)$. In G', v is eliminated, and a new edge (v_1, v_2) with weight $\phi(v_1, v) + \phi(v, v_2)$ is inserted. Therefore, the path $(u, \ldots, v_1, v_2, \ldots, w)$ is a shortest path from u to w in G' with $dist_{G'}(u,w) = dist_G(u,w)$. Therefore, the distance from u to w is preserved. This case is illustrated in Fig. 13 (b).

From the these two cases, we conclude that $G' = G \ominus v$ is a DP-graph of G. □

Proof of Lemma 6.4. For time complexity, the dominant cost for each node (line 3-7 of Algorithm 3) include the operation to maintain and select the node with the smallest degree, which costs $O(m+n \cdot \log(n))$ time, and the $\ominus$ operation (Algorithm 2) which costs $O(|N(v,H)|^2) = O(w^2)$ time. Therefore, the overall time complexity of Algorithm 3 is $O(n \cdot (w^2 + \log(n)) + m) = O(n \cdot (w^2 + \log(n))$. For space complexity, for each node, we maintain a star which costs $O(w)$ space. Therefore, the space complexity of Algorithm 3 is $O(n \cdot w)$. □

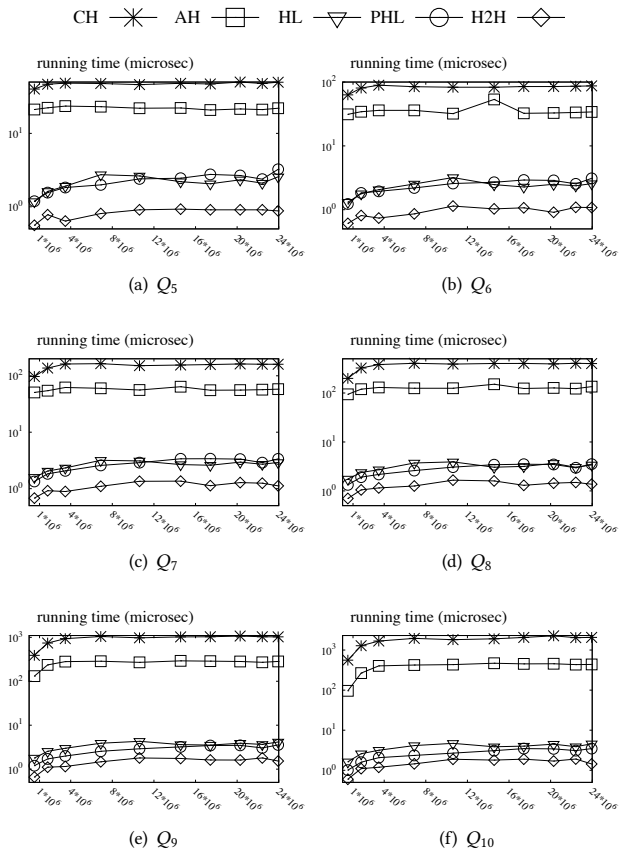
Proof of Lemma 6.8. We prove the lemma using induction by the length of the path from $X(v)$ to the root of T_G, i.e., the depth of $X(v)$ in T_G. When the depth of $X(v)$ is 0, i.e., $X(v)$ is the root of T_G, $G(v)$ only contains one vertex. Therefore, $G(v) \sqsubseteq G$. Assuming that the lemma holds when the depth of $X(v)$ is no larger than i, we now prove that the lemma holds when the depth of $X(v)$ is $i+1$. Supposing $X(v')$ is the parent of $X(v)$ in T_G, we know the depth of $X(v')$ is i. For any pair of vertices u and w in $G(v)$, we consider two cases:

- *Case 1: $u \neq v$ and $w \neq v$.* In this case, u and w are nodes in $G(v')$ and $dist_{G(v')}(w,u) = dist_G(w,u)$ by assumption. Since $G(v')$ is a subgraph of $G(v)$, we know that $dist_{G(v)}(w,u) = dist_{G(v')}(w,u) = dist_G(w,u)$.
- *Case 2: $u = v$ or $w = v$.* Without loss of generality, we suppose $w = v$. Let H be the graph in Algorithm 3 before eliminating v, we know that $X(v) \setminus \{v\}$ is the neighbor set of v in H. Since $X(u)$ is an ancestor of $X(v)$ in T_G, we know that u has not been eliminated when eliminating v from H, i.e., $u \in V(H)$. Therefore, we have $dist_H(v,u) = \min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_H(x,u)\}$. According to Lemma 6.3, we know that H is a DP-graph. Therefore, we have $dist_G(v,u) = dist_H(v,u) = \min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_H(x,u)\} = \min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_G(x,u)\}$. In addition, $X(v)\setminus\{v\}$ is also the neighbor set of v in $G(v)$. Therefore, we have $dist_{G(v)}(v,u) = \min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_{G(v)}(x,u)\}$. Since $G(v')$ is a subgraph of $G(v)$, we have $\min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_{G(v)}(x,u)\} = \min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_{G(v')}(x,u)\}$. Here, $X(v) \setminus \{v\}$ is a subset of vertices in $G(v')$, thus by assumption, we have $\min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_{G(v')}(x,u)\} = \min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_G(x,u)\}$. Based on the above analysis, we can derive:

$$\begin{aligned} dist_{G(v)}(w,u) &= dist_{G(v)}(v,u) \\ &= \min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_{G(v)}(x,u)\} \\ &= \min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_{G(v')}(x,u)\} \\ &= \min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_G(x,u)\} \\ &= \min_{x \in X(v)\setminus\{v\}}\{\phi(v,x) + dist_H(x,u)\} \\ &= dist_H(v,u) = dist_G(v,u) = dist_G(w,u) \end{aligned}$$

Based on the above analysis, we know $dist_{G(v)}(w,u) = dist_G(w,u)$ for any pair of vertices w and u in $G(v)$. Therefore, the lemma holds by induction. □

Proof of Theorem 6.9. As proved in Lemma 6.4, line 1 of Algorithm 4 requires $O(n \cdot (w^2 + \log(n)))$ time. In the for loop (line 2-7 of Algorithm 4), for each vertex v, line 3 of Algorithm 4 requires $O(w \cdot h)$ time. Line 4 of Algorithm 4 requires $O(w \cdot h)$ time since $G(v)$ is the union of at most h stars each of which has at most $w+1$ vertices. Line 5 requires $O(w \cdot h \cdot \log(h))$ time, and line 6-7 of Algorithm 4 requires $O(h)$ time. Therefore, the overall time



(a) Q_5 (b) Q_6 (c) Q_7 (d) Q_8 (e) Q_9 (f) Q_{10}

Figure 14: Query Processing Time (Varying Dataset Size)

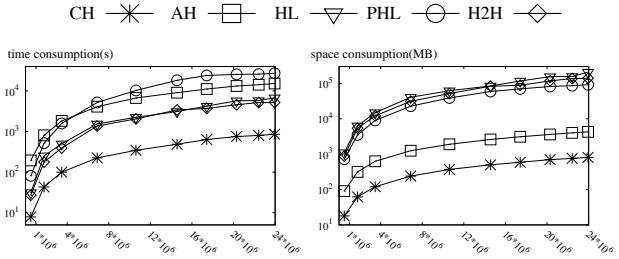
Note Size	1-3	4-6	7-15	16-30	31-100	>100
Percentage	83.46%	10.07%	4.23%	1.34%	0.74%	0.16%

Table 3: Node size distribution of T_G for the *US* dataset

complexity of Theorem 6.9 is $O(n \cdot (w^2 + h \cdot w \cdot \log(h) + \log(n))) = O(n \cdot (h \cdot w \cdot \log(h) + \log(n)))$ since $w + 1 \leq h$. □

Proof of Lemma 6.11. Since $G(v)$ is a DP-graph with all vertices in $X(v).anc$, we only need to show that the lemma holds in $G(v)$. Note that $X(v) \setminus \{v\}$ is the set of neighbors in $G(v)$. Therefore, we can derive that $X(v).dis_i = dist_{G(v)}(v, X(v).anc_i) = \min_{x \in X(v) \setminus \{v\}} \phi(v, x) + dist_{G(v)}(x, X(v).anc_i) = \min_{1 \leq j < |X(v)|} X(v).\phi_j + dist(x_j, X(v).anc_i)$. □

Proof of Theorem 6.13. As proved in Lemma 6.4, line 1 of Algorithm 5 requires $O(n \cdot (w^2 + \log(n)))$ time. In the for loop (line 2-12 of Algorithm 5), for each vertex v, line 4-5 of Algorithm 5 requires $O(w \cdot h)$ time. In the for loop from line 6 to line 12 of Algorithm 5, line 8-11 requires $O(w)$ time and the loop terminates in at most h iterations. Therefore, the for loop (line 6-12 of Algorithm 5) requires $O(w \cdot h)$ time. In summary, the overall time complexity of Theorem 6.13 is $O(n \cdot (w^2 + h \cdot w + \log(n))) = O(n \cdot (\log(n) + h \cdot w))$ since $w + 1 \leq h$. □



(a) Indexing Time (b) Index Size

Figure 15: Indexing Time and Index Size (Scalability)

10.4 Additional Experimental Results

Node Size Distribution. The node size distribution of the tree decomposition T_G for the *US* dataset is shown in Table 3. Thefirst row is the node size in different ranges, and the second row shows the percentage of tree nodes falling in the corresponding size range. According to Table 3, more than 97% of tree nodes have a size of less than 16. This explains why the tree decomposition algorithm is fast in practice.

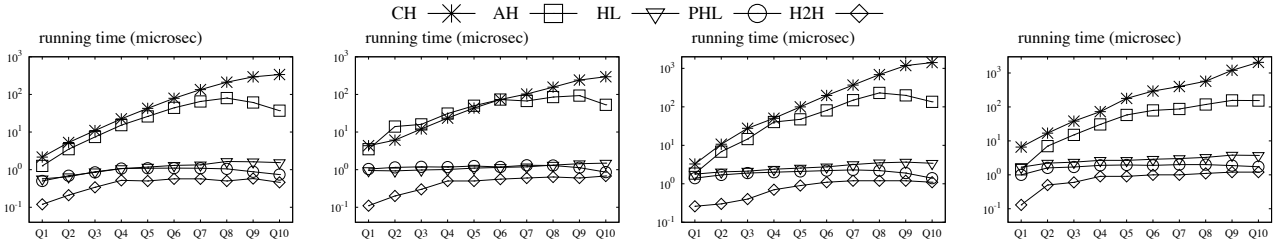
Varying Query Distance. The additional experimental results when varying query distance from Q_1 to Q_{10} for the 4 datasets *NY*, *FLA*, *E-US*, and *W-US* are shown in Fig. 16 (a) to (d) respectively. The curves are similar to those in the other 4 datasets shown in Fig. 10. Compared to the competitors CH, AH, HL, and PHL, our algorithm H2H performs best in all testing cases.

Varying Dataset Size. The additional experimental results when varying dataset size for queries Q_5 to Q_{10} are shown in Fig. 14 (a) to (f) respectively. In addition to the observations made in Fig. 11, we also observe that when $dist(s, t)$ becomes large, the gap between the query processing time for allfi ve algorithms increases when the graph size is no larger than 8×10^6, and then becomes stable when the size of the network further increases. This is because for a large $dist(s, t)$, when the size of the network is small, the size of the vertex cut between s and t significantly increases when the network size increases, and when the size of the network is large, the size of the vertex cut between s and t becomes stable when the network size further increases.

Indexing Scalability. We test the indexing time and index size when varying the dataset size (number of vertices in the road network) from 10^6 to 24×10^6. The datasets with different sizes are generated using the same method as in Exp-2 of Section 7. The experimental results are shown in Fig. 15 (a) and (b) for indexing time and index size respectively. As illustrated from the experimental results, when the dataset size increases from 10^6 to 24×10^6, both the indexing time and the index size increase stably for all algorithms. The performance of H2H compared to other algorithms are similar to that in the indexing time and index size testing in Exp-3 and Exp-4 of Section 7 respectively.

10.5 Additional Related Work

Shortest distance in other networks. In addition to road networks, the shortest path distance query is also important in other networks such as scale-free networks and temporal networks. In scale-free networks, the most efficient method for a distance query



(a) *NY* (b) *FLA* (c) *E-US* (d) *W-US*

Figure 16: Query Processing Time (Varying Query Distance)

is also based on the 2-hop index. Pruned Landmark Labeling [4] and Hop-Doubling Labeling [20] are two state-of-the-art algorithms. Pruned Landmark Labeling calculates a vertex order, and computes the labels of vertices by pruning the landmarks following the vertex order. Hop-Doubling Labeling regards edges as the initial index and calculate the 2-hop labels by doubling the length of paths in each iteration. Tree decomposition is used in shortest distance problems [5, 15, 32, 35]. However, these approaches are not tailored for road networks and therefore cannot outperform the state-of-the-art algorithms such as Hub-based Labeling on road networks. There are also some works on temporal networks. For example, Timetable labeling [31] is an indexing algorithm to solve the earliest arrival path query, latest departure path query and shortest duration path query. TopChain [33] improves Timetable Labeling by transforming the input graph into a directed acyclic graph. Other works focus on the shortest path problem with constrained condition such as [19] and [25].

10.6 Applying H2H to Scale-free Networks

It is important to note that although our approach is suitable for road networks, it does not work well on scale-free networks (such as social networks and web graphs). This is because scale-free networks are known to contain a dense core and a tree-like fringe. The core part is usually huge and hard to be decomposed using tree decomposition. This property is also observed by other works such as [23]. The property prevents our approach to work well on scale-free networks.

REFERENCES

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *Proc. of ISEA'11*, pages 230–241, 2011.

[2] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *Proc. of ESA'12*, pages 24–35, 2012.

[3] T. Akiba, Y. Iwata, K.-i. Kawarabayashi, and Y. Kawata. Fast shortest-path distance queries on road networks by pruned highway labeling. In *Proc. of ALENEX'14*, pages 147–154, 2014.

[4] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proc. of SIGMOD'13*, pages 349–360, 2013.

[5] T. Akiba, C. Sommer, and K.-i. Kawarabayashi. Shortest-path queries for complex networks: Exploiting low tree-width outside the core. In *Proc. of EDBT'12*, pages 144–155, 2012.

[6] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity offi nding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

[7] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity offi nding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

[8] J. Arz, D. Luxen, and P. Sanders. Transit node routing reconsidered. In *Proc. of SEA'13*, pages 55–66, 2013.

[9] H. Bast, D. Delling, A. V. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. In *Algorithm Engineering - Selected Results and Surveys*, pages 19–80. 2016.

[10] H. Bast, S. Funke, and D. Matijević. Transit: ultrafast shortest-path queries with linear-time preprocessing. In *9th DIMACS Implementation Challenge—Shortest Path*, 2006.

[11] M. A. Bender and M. Farach-Colton. The lca problem revisited. In *Proc. of LASTI'00*, pages 88–94, 2000.

[12] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–21, 1993.

[13] H. L. Bodlaender. A linear-time algorithm forfi nding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[14] H. L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *Graph-Theoretic Concepts in Computer Science*, pages 1–14, 2006.

[15] L. Chang, J. X. Yu, L. Qin, H. Cheng, and M. Qiao. The exact distance to destination in undirected world. *The VLDB Journal*, 21(6):869–888, 2012.

[16] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Proc. of WEA'08*, pages 319–333, 2008.

[17] A. V. Goldberg and C. Harrelson. Computing the shortest path: *A* search meets graph theory. In *Proc. of SODA'05*, pages 156–165, 2005.

[18] R. Halin. S-functions for graphs. *Journal of Geometry*, 8:171–186, 1976.

[19] R. Hassin. Approximation schemes for the restricted shortest path problem. *Math. Oper. Res.*, 17(1):36–42, 1992.

[20] M. Jiang, A. W. Fu, R. C. Wong, and Y. Xu. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *PVLDB*, 7(12):1203–1214, 2014.

[21] S. Jung and S. Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Trans. Knowl. Data Eng.*, 14(5):1029–1046, 2002.

[22] A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics*, 8:54–57, 2001.

[23] T. Maehara, T. Akiba, Y. Iwata, and K. Kawarabayashi. Computing personalized pagerank quickly by exploiting graph structures. *PVLDB*, 7(12):1023–1034, 2014.

[24] S. Mozes and C. Sommer. Exact distance oracles for planar graphs. In *Proc. of SODA'12*, pages 209–222, 2012.

[25] M. N. Rice and V. J. Tsotras. Graph indexing of road networks for shortest path queries with label restrictions. *PVLDB*, 4(2):69–80, 2010.

[26] N. Robertson and P. D. Seymour. Graph minors iii: Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[27] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *Proc. of SIGMOD'08*, pages 43–54, 2008.

[28] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest path queries. In *Proc. of ESA'05*, pages 568–579, 2005.

[29] J. Sankaranarayanan and H. Samet. Query processing using distance oracles for spatial networks. *IEEE Trans. Knowl. Data Eng.*, 22(8):1158–1175, 2010.

[30] J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial networks. *PVLDB*, 2(1):1210–1221, 2009.

[31] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou. Efficient route planning on public transportation networks: A labelling approach. In *Proc. of SIGMOD'15*, pages 967–982, 2015.

[32] F. Wei. Tedi: efficient shortest path query answering on graphs. In *Proc. of SIGMOD'10*, pages 99–110. ACM, 2010.

[33] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke. Reachability and time-based path queries in temporal graphs. In *Proc. of ICDE'16*, pages 145–156, 2016.

[34] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *PVLDB*, 5(5):406–417, 2012.

[35] Y. Xiang. Answering exact distance queries on real-world graphs with bounded performance guarantees. *The VLDB Journal*, 23(5):677–695, 2014.

[36] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou. Shortest path and distance queries on road networks: Towards bridging theory and practice. In *Proc. of SIGMOD'13*, pages 857–868, 2013.