Workload-Aware Shortest Path Distance Querying in Road Networks

Bolong Zheng[1], Jingyi Wan[1], Yongyong Gao[1], Yong Ma[1], Kai Huang[2], Xiaofang Zhou[2], Christian S. Jensen[3]
[1]Huazhong University of Science and Technology, Wuhan, China
Email: {bolongzheng, jingyiwan, yongyonggao, yongma2}@hust.edu.cn
[2]The Hong Kong University of Science and Technology, Hong Kong, China
Email: {ustkhuang, zxf}@cse.ust.hk
[3]Aalborg University, Aalborg, Denmark
Email: csj@cs.aau.dk

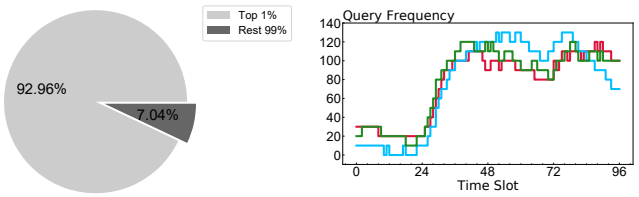
***Abstract*—Computing shortest-path distances in road networks is core functionality in a range of applications. To enable the efficient computation of such distance queries, existing proposals frequently apply 2-hop labeling that constructs a label for each vertex and enables the computation of a query by performing only a linear scan of labels. However, few proposals take into account the spatio-temporal characteristics of query workloads. We observe that real-world workloads exhibit (1) *spatial skew*, meaning that only a small subset of vertices are queried frequently, and (2) *temporal locality*, meaning that adjacent time intervals have similar query distributions. We propose a Workload-aware Core-Forest label index (WCF) to exploit spatial skew in workloads. In addition, we develop a Reinforcement Learning based Time Interval Partitioning (RL-TIP) algorithm that exploits temporal locality to partition workloads to achieve further performance improvements. Extensive experiments with real-world data offer insights into the performance of the proposals, showing that they achieve 62% speedup on average for query processing with less preprocessing time and space overhead when compared with the state-of-the-art proposals.**

I. Introduction

The near-ubiquitous use of smartphones has enabled transportation network companies to operate ride-hailing platforms that enable the servicing of massive amounts of transportation requests. Each request executes a shortest path distance query for a user-specified OD pair. It is challenging to process massive-scale shortest path distance query workloads efficiently. Extensive index-based proposals build auxiliary data structures during preprocessing, including ALT [1], HH [2], CH [3], AH [4], and TNR [5]. However, they are not viable for use in large scale road networks due to the increasing search space. A widely-adopted technique called 2-hop labeling [6] constructs a label for each vertex and is capable of processing a shortest path distance query with only a linear scan of labels. A series of proposals based on 2-hop labeling aim to further reduce the label size, including IS-Label [7], PLL [8], H2H [9], PSL [10], and BVC-PLL [11]. However, none of them are specially designed to handle massive-scale query workloads.

Motivation. Analyzing real-world query workloads from the New York dataset [12], we observe that the workloads exhibit spatio-temporal correlations.

1) *Spatial skew*. A small percentage of vertices are queried frequently within a short timespan. As shown in Fig. 1(a), the aggregate query frequencies of the top 1% vertices make up 92.96% of the total query workload, which indicates that the query overhead of the workload depends mainly on the top 1% vertices when millions of queries arrive simultaneously.
2) *Temporal locality*. When partitioning query workloads into sequences of time slots, the query frequencies of most vertices in adjacent time slots are similar. Fig. 1(b) shows the query frequencies of three vertices over time in different colors. We observe that they all have a low query frequency at 03:00 – 07:00, and a high query frequency at 08:00 – 13:00. This can be explained by commuters having similar travel patterns during peak and non-peak hours. If we can find the time intervals that have steady query frequencies, a fine-grained query frequency distribution can be modeled.



(a) Top 1% high frequency vertices (b) Query frequency distribution

Fig. 1. Characteristics of Real-World Workloads

Existing 2-hop labeling based proposals [3], [8], [9], [13], [14] attempt to reduce the label size by utilizing road-network topology, such as the hierarchical structure of a road network. However, they treat all vertices equally and do not consider spatial skew in workloads. Therefore, if we construct a workload-aware label index that reduces the label sizes of high frequency vertices, the total query overhead can be reduced significantly. In addition, if we identify time intervals with a fine-grained query frequency distribution, we can construct a workload-aware label index for each time interval to cope with different high frequency vertices, which enables further query processing speedups.

Following to the above observations, we first develop a workload-aware pruned landmark labeling (wPLL) method and a workload-aware hierarchical 2-hop (wH2H) method by

TABLE I
SUMMARY OF NOTATION

Notation	Definition
$G = (V, E, W)$	A road network
$n = \|V\|$	The number of vertices
$m = \|E\|$	The number of edges
$\mathsf{Dist}(s, t)$	Shortest path distance between s and t
$q(s, t)$	Shortest distance query on s and t
f_v	The query frequency of vertex v
V_H	A set of high frequency vertices
V_L	A set of low frequency vertices
$L(v)$	The label of v
$\mathsf{cost}(v)$	The query cost of v

modifying the state-of-the-art proposals PLL [8] and H2H [9], respectively. Although these methods enable more efficient querying owing to their smaller label sizes of high frequency vertices, they increase the space consumption and preprocessing time.

To overcome these limitations, we develop a workload-aware core-forest label index (WCF) that exploits the advantages of wPLL and wH2H. WCF maintains a core-forest structure that consists of two layers. The upper level is the core that builds wPLL on an overlay graph for high frequency vertices by taking both topological centrality and query frequency into consideration. The lower level employs a tree decomposition and obtains a set of sub-graphs from the road network for low frequency vertices. A forest is constructed by building a tree index for each sub-graph. In addition, to exploit the temporal locality in the workloads, we propose a reinforcement learning based time interval partitioning (RL-TIP) that partitions workloads that can then be processed separately with improved performance.

We make the following major contributions:

- We propose a data-driven solution WCF that accelerates workload query processing by exploiting predicted query frequencies in workloads.
- We develop a reinforcement learning based method RL-TIP that partitions workloads into appropriate time intervals, within which WCF is applied to further accelerate the query workload processing.
- We report on an extensive performance study using real-world data that covers the state-of-the-art algorithms and offers evidence that WCF is able to achieve 62% speedup on average for query processing with less preprocessing time and space overhead.

The rest of the paper is organized as follows. We formalize the problem in Section II. Sections III and IV present the two methods wPLL and wH2H, and Sections V and VI cover WCF and RL-TIP. The experimental study is presented in Section VII. Section VIII reviews related work, and Section IX concludes the paper. Pseudocodes are given in a technical report [15].

II. PRELIMINARIES

We proceed to cover the preliminary knowledge and the problem definition. Frequently used notation is summarized in Table I.

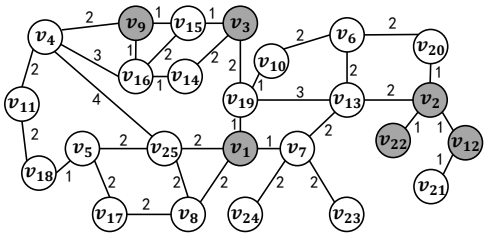


Fig. 2. A Running Example

A. Problem Setting

Definition 1 (Road Network)**.** *We model a road network as a weighted graph $G = (V, E, W)$, where V is a set of n vertices, E is a set of m edges, and $W : E \to \mathbb{R}^+$ assigns a non-negative weight $w(v_i, v_j)$ for each edge $(v_i, v_j) \in E$ that captures the edge length or cost.*

For ease of presentation, we first discuss the proposed methods on undirected road networks, and then extend the discussion on directed road networks in Sec. V-E.

Definition 2 (Shortest Path Distance)**.** *Given a source vertex s and a destination vertex t, a path between s and t, denoted as $\mathsf{P}(s, t) = \langle s, \ldots, v_i, v_{i+1}, \ldots, t\rangle$, is a sequence of vertices. Let $v_1 = s$ and $v_k = t$, we have $(v_i, v_{i+1}) \in E$ for $1 \le i < k$.*

The shortest path between s and t, denoted as $\mathsf{SP}(s, t)$, is the path between s and t with the minimum sum of edge weights. The shortest path distance between s and t, denoted as $\mathsf{Dist}(s, t)$, is the sum of the edge weights of $\mathsf{SP}(s, t)$.

Definition 3 (Shortest Path Distance Query)**.** *A shortest path distance query $q(s, t)$ takes a source vertex s and a destination vertex t as parameters and returns the shortest path distance $\mathsf{Dist}(s, t)$.*

Example 1. *Fig. 2 shows a toy road network with 25 vertices and 34 edges. The shortest path $\mathsf{SP}(v_3, v_9) = \langle v_3, v_{15}, v_9\rangle$, so $q(v_3, v_9)$ returns $\mathsf{Dist}(v_3, v_9) = 2$.*

B. 2-hop Labeling

The 2-hop labeling technique [6] is adopted widely in existing approaches to shortest path distance computing [7]–[11], [16]. To allow efficient query processing, 2-hop labeling assigns each vertex v a label $L(v)$ that contains a set of key/distance value pairs $(o, \mathsf{Dist}(v, o))$. We call the projection of $L(v)$ on the keys hops, i.e., $hop(v) = \{o|(o, \mathsf{Dist}(v, o)) \in L(v)\}$, and we define the projection of $L(v)$ on the values as $spd(v) = \{\mathsf{Dist}(v, o)|(o, \mathsf{Dist}(v, o)) \in L(v)\}$.

To guarantee query processing correctness, the label index must obey the 2-hop cover property: for any vertex pair s and t, the intersection of the labels of s and t contains at least one hop o on the shortest path $\mathsf{SP}(s, t)$. Given a shortest path distance query $q(s, t)$, the shortest path distance is computed as follows:

$$\mathsf{Dist}(s, t) = \min_{o \in hop(s) \cap hop(t)} \{\mathsf{Dist}(s, o) + \mathsf{Dist}(o, t)\} \quad (1)$$

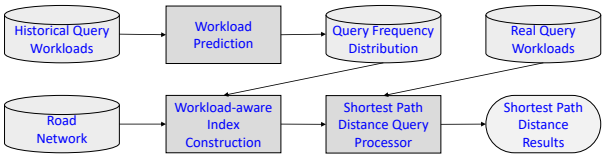


Fig. 3. Framework Overview

Therefore, we compute $\mathsf{Dist}(s,t)$ by scanning $L(s)$ and $L(t)$. The query time is $O(|L(s)|+|L(t)|)$.

Example 2. *Fig. 4 shows a label index of the running example. To compute* $\mathsf{Dist}(v_3,v_9)$*, we have to probe* $L(v_3)$ *and* $L(v_9)$*, where* $hop(v_3) \cap hop(v_9) = \{v_1, v_3\}$*. So* $\mathsf{Dist}(v_3,v_9) = \min\{3+5, 0+2\} = 2$.

C. Problem Definition

Let $\mathsf{cost}(q_i)$ be the query cost of q_i. Assume a query workload $Q = \{q_i\}$, the goal is to minimize the workload query cost.

$$\mathsf{cost}(Q) = \sum\nolimits_{q_i \in Q} \mathsf{cost}(q_i) \tag{2}$$

Using 2-hop labeling, the query cost of each $q_i(s_i,t_i)$ is $O(|L(s_i)|+|L(t_i)|)$. Since we have to probe entries in both $L(s_i)$ and $L(t_i)$, s_i and t_i are symmetric and are treated equally. Let V_Q be the set of vertices in Q, i.e., $V_Q = \cup\{s_i,t_i\}$. Let f_v be the frequency (number of occurrences) of $v \in V_Q$. We reform ultra Eq. 2 as follows.

$$\begin{aligned}\mathsf{cost}(Q) &= \sum\nolimits_{q_i \in Q} \mathsf{cost}(q_i) = \sum\nolimits_{q_i \in Q} \mathsf{cost}(s_i) + \mathsf{cost}(t_i) \\ &= \sum\nolimits_{v \in V_Q} \mathsf{cost}(v) \cdot f_v \approx \sum\nolimits_{v \in V_Q} |L(v)| \cdot f_v\end{aligned} \tag{3}$$

D. Framework Overview

From Eq. 3, we know that the query cost of workload depends on the label sizes and query frequencies of vertices. We can reduce the cost by taking into account the query frequency distribution among vertices. In addition, we can reduce the cost by reducing the label size, especially that of the vertices with high query frequency. Therefore, the framework in Fig. 3 includes three components: (1) *workload prediction*, (2) *workload-aware index construction*, and (3) *shortest path distance query processing*.

The *workload prediction* module learns a model based on historical query workload data. Extensive deep learning methods [17]–[19] are proposed to predict taxi OD demands. As this line of studies is orthogonal to our work, we use a well-adopted model GraphWaveNet [19] to predict the distribution of OD query pairs and transform it into the query frequency distribution of vertices. The *workload-aware index construction* module constructs label indexes for vertices based on the predicted workload. The *shortest path distance query processing* module takes real query workloads as input and returns the results of the queries.

v_1	$(v_1,0)$	v_{14}	$(v_1,5),(v_3,2),(v_9,2),(v_{14},0),(v_{16},1)$
v_2	$(v_1,5),(v_2,0),(v_3,7)$	v_{15}	$(v_1,4),(v_3,1),(v_9,1),(v_{15},0)$
v_3	$(v_1,3),(v_3,0)$	v_{16}	$(v_1,6),(v_3,3),(v_9,1),(v_{16},0)$
v_4	$(v_1,6),(v_3,4),(v_4,0),(v_9,2)$	v_{17}	$(v_1,4),(v_4,7),(v_5,2),(v_8,2),(v_{17},0)$
v_5	$(v_1,4),(v_4,5),(v_5,0),(v_9,7)$	v_{18}	$(v_1,5),(v_4,4),(v_5,1),(v_9,6),(v_{18},0)$
v_6	$(v_1,4),(v_2,3),(v_3,5),(v_6,0)$	v_{19}	$(v_1,1),(v_2,5),(v_3,2),(v_6,3),(v_{19},0)$
v_7	$(v_1,1),(v_2,4),(v_6,4),(v_7,0)$	v_{20}	$(v_1,6),(v_2,1),(v_3,7),(v_6,2),(v_{20},0)$
v_8	$(v_1,2),(v_4,6),(v_5,4),(v_8,0)$	v_{21}	$(v_1,7),(v_2,2),(v_3,9),(v_{12},1),(v_{21},0)$
v_9	$(v_1,5),(v_3,2),(v_9,0)$	v_{22}	$(v_1,6),(v_2,1),(v_3,8),(v_{22},0)$
v_{10}	$(v_1,2),(v_2,5),(v_3,3),(v_6,2),(v_{10},0),(v_{19},1)$	v_{23}	$(v_1,3),(v_2,6),(v_6,6),(v_7,2),(v_{23},0)$
v_{11}	$(v_1,7),(v_3,6),(v_4,2),(v_5,3),(v_9,4),(v_{11},0),(v_{18},2)$	v_{24}	$(v_1,3),(v_2,6),(v_6,6),(v_7,2),(v_{24},0)$
v_{12}	$(v_1,6),(v_2,1),(v_3,8),(v_{12},0)$	v_{25}	$(v_1,2),(v_4,4),(v_5,2),(v_8,2),(v_9,6),(v_{25},0)$
v_{13}	$(v_1,3),(v_2,2),(v_3,5),(v_6,2),(v_{13},0),(v_{19},3)$		

Fig. 4. Labels of wPLL

III. A Vertex Ordering Approach

PLL [8] computes labels in a vertex order that considers only topological centrality. However, since the frequency distribution of queries in a workload is skewed, it is beneficial to consider also the query frequency when determining vertex importance. Therefore, we propose a workload-aware pruned landmark labeling (wPLL) method that exploits both topological centrality and query frequency for vertex ordering.

A. Workload-aware Vertex Ordering

PLL finds a small 2-hop label index efficiently by processing vertices in descending order of centrality. The notions of degree, betweenness, and closeness can be used to measure vertex centrality for ordering. The order is crucial for the performance, and the betweenness has proven to be the most effective for PLL [20].

Intuitively, on the one hand, high centrality vertices are likely to cover many shortest paths. On the other hand, according to the properties of PLL, high centrality vertices usually have small labels, since they are considered as hops and are pushed to the labels of low centrality vertices. If we process the vertices with high query frequencies first, they may have small label sizes. This can reduce the workload query cost significantly. Therefore, we propose a simple yet effective ordering that computes the vertex importance $\sigma(v)$ by considering both centrality and query frequency as follows.

$$\sigma(v) = \beta \cdot f_v^* + (1-\beta) \cdot b_v^*, \tag{4}$$

where $f_v^* = (f_v - f_{min})/(f_{max} - f_{min})$ is the normalized query frequency of v, and f_{max} and f_{min} are the maximum and minimum query frequencies. Next, $b_v^* = (b_v - b_{min})/(b_{max} - b_{min})$ is the normalized betweenness of v, where b_{max} and b_{min} are the maximum and minimum betweenness values. The parameter

β is used to balance the effects of the centrality and the query frequency. We adopt SamPG [21] to compute the betweenness of vertices. We process the vertices in descending order of vertex importance, and the hop pushing process that generates labels is the same as PLL. Interested readers may refer to reference [8] for more details.

Example 3. *For simplicity, we assign the vertex ID based on its betweenness centrality, i.e., v_1 has the highest betweenness. Assume a workload $Q = \langle q_1(v_1, v_3), q_2(v_3, v_9), q_3(v_1, v_3), q_4(v_3, v_9), q_5(v_1, v_3), q_6(v_1, v_2), q_7(v_{12}, v_9), q_8(v_2, v_{22})\rangle$. We have $f_{v_3} = 5$, $f_{v_1} = 4$, $f_{v_9} = 3$, $f_{v_2} = 2$, $f_{v_{12}} = 1$, $f_{v_{22}} = 1$. Setting $\beta = 0.1$, we compute the vertex importance based on Eq. 4. The vertices are processed in descending order of $\sigma(v)$, i.e., $\langle v_1, v_3, v_2, v_9, v_{22}, v_{12}, v_4, v_5, v_6, v_7, v_8, v_{16}, v_{19}, v_{25}, v_{13}, v_{15}, v_{14}, v_{10}, v_{17}, v_{18}, v_{11}, v_{20}, v_{24}, v_{23}, v_{21}\rangle$. After processing all vertices, we obtain the labels in Fig. 4. Accordingly, we have $|L(v_3)| = 2$, $|L(v_1)| = 1$, $|L(v_9)| = 3$, $|L(v_2)| = 3$, $|L(v_{12})| = 4$, and $|L(v_{22})| = 4$. We assume that the unit cost of probing an entry is 1, the workload cost $\mathsf{cost}(Q) = 2\times5+1\times4+3\times3+3\times2+4\times1+4\times1 = 37$.*

IV. A Tree Decomposition Approach

We proceed to present a workload-aware hierarchical 2-hop index (wH2H) that extends H2H [9] to process query workloads efficiently. H2H uses a tree decomposition that imposes a vertex ordering among vertices and maps a road network G to a tree T, which enables to answer $q(s,t)$ efficiently when s and t are near each other. We observe that the labels of the vertices near the root of T are small. Intuitively, if we place the vertices with high query frequencies near the root, the workload query cost can be reduced.

A. Tree Decomposition

Let T be the tree decomposition of a road network $G = (V, E)$. Let V_T be the set of nodes in T. Each node of T is denoted by $X(v)(v \in V) \in V_T$ that contains a subset of V, and the following conditions hold:

1) $\cup X(v) = V$.
2) For each edge $(u, u') \in E$, there exists a node $X(v)$ such that both $u, u' \in X(v)$.
3) For each vertex $u \in V$, the set $\{X(v)|u \in X(v)\}$ forms a connected subtree of T.

For clarity, we refer to each $v \in V$ in road network G as a vertex and to each $X(v) \in V_T$ in tree decomposition T as a node. The tree width $\omega(T)$ is one less than the maximum size of all nodes, i.e., $\omega(T) = \max|X(v)| - 1$, where $|X(v)|$ is the number of vertices in $X(v)$. The tree height $h(T)$ is the maximum distance from all nodes to the root.

In order to answer a shortest path distance query, each node $X(v)$ has two arrays: a position array $pos(v)$ and a distance array $dis(v)$. For each $u \in X(v)$, $pos(v)$ records the position of $X(u)$ in T, i.e., the height of $X(u)$ in T. Next, $dis(v)$ saves the distances from $X(v)$ to all its ancestors $anc(v)$ (from the root node to $X(v)$). The computation of $dis(v)$ is performed in a top-down manner from the root node to itself.

The shortest path distance computation relies on two properties of the tree decomposition.

1) Let $\mathsf{LCA}(s,t)$ be the lowest common ancestor of $X(s)$ and $X(t)$. For any two vertices $s, t \in V$, $\mathsf{SP}(s,t)$ goes through at least one vertex in $\mathsf{LCA}(s,t)$.
2) Given a node $X(s)$ in T, for any $v \in X(s) \setminus \{s\}$, $X(v)$ is an ancestor of $X(s)$ in T.

Given a query $q(s,t)$, we first find $\mathsf{LCA}(s,t)$ in T with $O(1)$ time [22]. Then we compute $\mathsf{Dist}(s,t)$ according to the above two properties as follows.

$$\mathsf{Dist}(s,t) = \min_{i \in pos(\mathsf{LCA}(s,t))} \{dis(s)[i] + dis(t)[i]\} \quad (5)$$

Example 4. *Fig. 5 shows a tree decomposition of the running example. For $q(v_{12}, v_{22})$, we first find $\mathsf{LCA}(v_{12}, v_{22}) = X(v_2)$ and $pos(v_2) = \{1, 2, 3\}$. Then we have $\mathsf{Dist}(v_{12}, v_{22}) = \min\{8+8, 6+6, 1+1\} = 2$ based on Eq. 5.*

B. Vertex Ordering for Tree Decomposition

H2H adopts a MinDegree method [23] for tree decomposition with time cost $O(n \cdot (\omega^2 + \log n))$. The main idea is to process each v and create $X(v)$ in ascending order of degree, so the nodes with high degrees are near to the root. To place the high frequency vertices near to the root, a straight-forward approach is to compute the vertex importance by considering both the query frequency and the degree as follows,

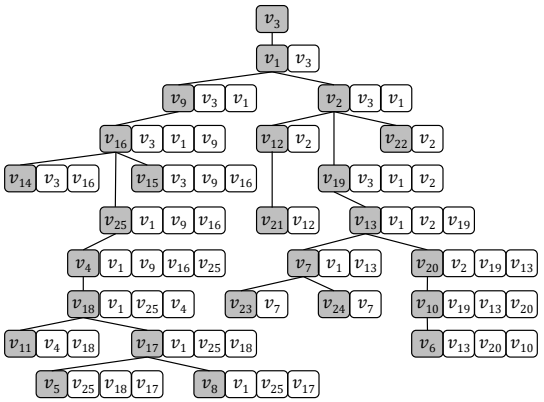
$$\sigma(v) = \gamma \cdot f_v^* + (1-\gamma) \cdot d_v^*, \quad (6)$$

where $d_v^* = d_v/d_{max}$, d_v is the degree when processing v during the tree decomposition, and d_{max} is the maximum degree. The intuition why MinDegree processes vertices in ascending order of degree is that the fewest edges are added in each step, yielding a tree with small width ω and height h. Therefore, if we process vertices in ascending order of the vertex importance defined in Eq. 6, both ω and h increase thus increasing the additional space costs.

To address this issue, we develop a block partitioning technique that first sorts the vertices in ascending order of the query frequency and then partitions them into blocks. Each block has a maximum size η, so we have $N = \lceil n/\eta \rceil$ blocks. The vertex importance is re-computed as follows.

$$\sigma(v) = \gamma \cdot \frac{\sum_{u \in B_i} f_u - \sum_{o \in B_1} f_o}{\sum_{u' \in B_N} f_{u'} - \sum_{o \in B_1} f_o} + (1-\gamma) \cdot d_v^*, \quad (7)$$

where B_i is the block that v_i belongs to, B_1 is the first block with the vertices with the lowest frequencies and B_N is the N-th block with the vertices with the highest frequencies. Using Eq. 7, we process the vertices from different blocks in an order determined by both the degree and the query frequency, and we process vertices from the same block in an order determined by only the degree, which effectively reduces ω and h. Note that when the block size $\eta = 1$, Eq. 7 degrades to Eq. 6.



(a) Tree Decomposition

V	pos(v)	dis(v)	V	pos(v)	dis(v)
v_1	1, 2	3, 0	v_9	1, 2, 3	2, 5, 0
v_2	1, 2, 3	7, 5, 0	v_{12}	3, 4	8, 6, 1, 0
v_3	1	0	v_{15}	1, 3, 4, 5	1, 4, 1, 2, 0
v_4	2, 3, 4, 5, 6	9, 6, 2, 3, 4, 0	v_{22}	3, 4	8, 6, 1, 0
v_7	2, 5, 6	4, 1, 4, 2, 2, 0	···	·····	·····

(b) $pos(v)$ and $dis(v)$

Fig. 5. Index Structure of wH2H

C. MinImp Method

We propose a MinImp method that processes each v and creates $X(v)$ in ascending order of the vertex importance in Eq. 7. We first initialize a graph $G' = (V', E')$ as a copy of G. For each vertex v, we enumerate its neighbor pairs. Let $N(v)$ be the set of neighbors of v. For any $u, u' \in N(v)$, if edge $(u, u') \notin E'$, an edge (u, u') with weight $w'(u, v) + w'(v, u')$ is inserted into G'. If edge $(u, u') \in E'$, its weight $w'(u, u')$ is updated by $\min\{w'(u, u'), w'(u, v) + w'(v, u')\}$. This ensures that the lengths of the shortest paths that pass (u, u') remain correct after removing v. After we enumerate all the neighbor pairs of v, whose time cost is $O(\omega^2)$, we remove v from G' and create a node $X(v) = \{v\} \cup N(v)$. Note that the degrees of v's neighbors change after we remove v. Therefore, we update the vertex importance by Eq. 7 and maintain a min-heap to keep the vertices in ascending order of importance. The cost of the importance update is $O(\log n)$. When we obtain all nodes, we proceed to connect the nodes. We set the node $X(u)$ to be the parent of a node $X(v)$ if u is the neighbor of v with the smallest vertex importance after processing v. Finally, we obtain a one-to-one mapping from V to V_T.

Example 5. *Consider the workload Q in Example 3, the vertices are partitioned into 13 blocks with $\gamma = 0.2$ and $\eta = 2$. As shown in Fig. 5(a), the vertex importance of vertices in $V_Q = \{v_1, v_2, v_3, v_9, v_{12}, v_{22}\}$ exceeds that of vertices in $V - V_Q$. So nodes $X(v_1)$, $X(v_2)$, $X(v_9)$, $X(v_{12})$, and $X(v_{22})$ are placed near the root $X(v_3)$. For $q_1(v_1, v_3)$, we have $\mathsf{cost}(q_7(v_1, v_3)) = 1$ since $X(v_3) \in anc(v_1)$. Similarly, we have $\mathsf{cost}(q_2(v_3, v_9)) = \mathsf{cost}(q_6(v_1, v_2)) = 1$. The workload cost $\mathsf{cost}(Q) = 9$.*

V. Hybrid Approach

We proceed to introduce the workload-aware core-forest label index (WCF) that exploits both the advantages of wPLL and wH2H. Although wPLL and wH2H both enable making good use of the spatial skew in workloads to improve performance significantly, they nevertheless have their own pros and cons that are analyzed as follows.

- *Space overhead*. Given a road network G with tree width ω, the space overhead of wPLL is $O(n \cdot \omega \log n)$, and that of wH2H is $O(n \cdot h)$. Note that the tree height h usually exceeds $\omega \log n$.
- *Index construction cost*. wPLL takes $O(\omega m \log n + \omega^2 n \log^2 n)$ time, while wH2H takes $O(nh\omega + n \log n)$ time to construct the index. As the tree decomposition is fast and the top-down distance computation saves the cost, wH2H is more efficient than wPLL at index construction.
- *Query cost*. For queries with short distances, wH2H is faster. For queries with long distances, they are nearly the same.
- *Applicability*. wPLL is preferable for complex graphs with high degrees, while wH2H is preferable for graphs with low degrees, such as the road network [24], [25].

For the best of both worlds, we use wPLL for high frequency vertices to reduce the query cost, and use wH2H for low frequency vertices to decrease the index construction cost.

A. Core-forest Structure Extraction

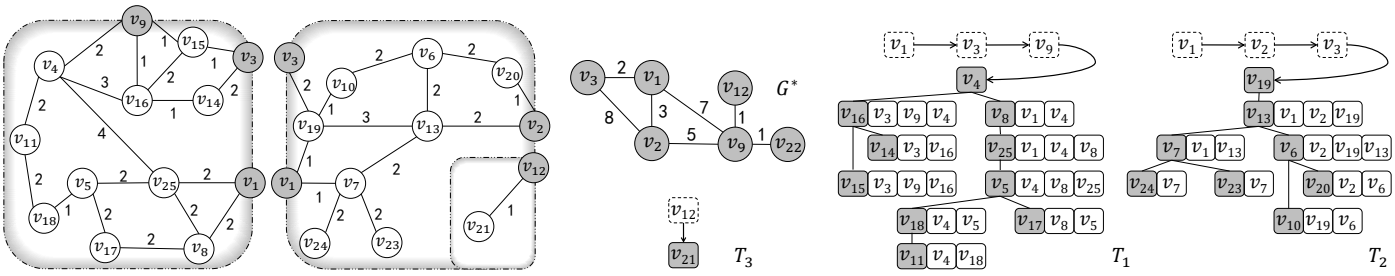
To construct the WCF index, we first extract the hierarchical core-forest structure and then build indexes for the core and the forest separately.

Given a workload Q, suppose the vertices of V_Q are sorted in descending order of the query frequency. Let V_H be the set of high frequency vertices where $n_H = |V_H|$, and let V_L be the set of low frequency vertices where $V_L = V - V_H$ and $n_L = |V_L| = n - n_H$. We take the top α percent of vertices in V_Q as the high frequency vertices, i.e., $\alpha = |V_H|/|V_Q|$. We set α to 100% by default since experiments suggest that when $\alpha = 100\%$, the algorithm achieves the best performance.

Intuitively, we apply a tree decomposition method that isolates the vertices in V_L from the vertices in V_H. By removing V_L from road network G, the remaining V_H constructs the core that is a dense overlay graph G^* with high degrees.

Fortunately, $|V_H|$ is usually small compared with $|V|$, so wPLL is an appropriate choice of method for constructing a label index for V_H. For the removed V_L, a forest is generated during the tree decomposition, where each tree constructs a label index by using wH2H. Therefore, we are able to reduce the total query cost by improving the query performance on V_H significantly, while sacrificing the query performance on V_L only moderately. However, if we remove all vertices in V_L, the average degree of G^* may become extremely large, which affects the query performance of G^* negatively. To avoid this issue, we set a parameter ω_{max} that confines the degree of the vertex to be removed in each iteration.

To extract the hierarchical core-forest structure from $G' = (V', E')$ where $G' = G$, we develop a RelaxedMinDegree method for the tree decomposition. At each iteration,



(a) Sub-graphs of Tree Decomposition (b) Core-forest Structure

Fig. 6. Core-forest Structure Construction

it removes a vertex $v \in V_L$ with the minimum degree as follows. For any $u, u' \in N(v)$, if edge $(u, u') \notin E'$, an edge (u, u') with weight $w'(u, v) + w'(v, u')$ is added into G'. If edge (u, u') already exists in G', then $w'(u, u') = \min\{w'(u, u'), w'(u, v) + w'(v, u')\}$. Note that we only consider the degree for vertex ordering since the frequency is or close to 0 in V_L. Compared to MinImp method, we have three main differences:

1) We only remove low frequency vertices from V_L.
2) The process terminates when all vertices in V_L are removed or the degree of the vertex to be removed exceeds ω_{max}.
3) We propose a new degree update strategy when removing a vertex that reduces the time cost from $O(\log n)$ to $O(1)$. Specifically, we construct an inverted list instead of a min-heap for the vertices in V_L. Initially, we partition the vertices into different lists based on the degree, and we scan the lists in ascending order of the degree. When we remove a vertex, we update the locations of its neighbors in the inverted list.

After tree decomposition, we note that the remaining graph G^* contains not only the vertices in V_H but also a few vertices in V_L due to the parameter ω_{max}. For simplicity, we still use V_H to denote the vertices in G^* and V_L to denote the truly removed vertices when the context ensures that there is no ambiguity.

We connect the nodes generated by RelaxedMinDegree in a bottom-up manner similarly to how this is done for wH2H. The difference is that when we encounter a node $X(v)$ such that all vertices in $X(v)$ are in V_H except v, we set $X(v)$ to be a root node and a tree T_i is constructed. In other words, the last removed vertex of each tree is surrounded by a set of vertices in V_H. We call these vertices in $X(v) \setminus \{v\}$ the border vertices, denoted by VB_i, that separate the nodes of T_i from those of the other trees. After we process all nodes, a forest is constructed. As we confine the degrees of the removed vertices by ω_{max}, the tree width $\omega_i \leq \omega_{max}$ for each tree T_i.

Theorem 1. *The time complexity of the core-forest structure construction is* $O(n_L \cdot \omega_{max}^2)$.

Proof. Since the degree update of RelaxedMinDegree takes $O(1)$ time, the edge update costs $O(\omega_{max}^2)$ time. As we remove n_L vertices, the proof follows. □

Example 6. *Fig. 6 shows the core-forest structure of Fig. 2. The trees* $\{T_1, T_2, T_3\}$ *form a forest, where the border vertices are on the top of the root of each tree. The core* G^* *contains* $V_H = \{v_1, v_2, v_3, v_9, v_{12}, v_{22}\}$.

B. Index Construction

We proceed to introduce the label index construction for the core-forest structure, where we use $\mathcal{L}_H$ and $\mathcal{L}_L$ to denote the labels of the vertices in V_H and V_L, respectively.

For high frequency vertices in V_H, we apply wPLL to the core graph G^*. For each vertex $v \in V_H$, the label of v, denoted by $\mathcal{L}_H(v)$, contains a hop array $hop(v)$ and a distance array $spd(v)$, as shown in Fig. 7(a).

For low frequency vertices in V_L, we introduce the label of each vertex $v \in V_L$ in a tree T_i. Let $X(v_r)$ be the root node of T_i. The label $\mathcal{L}_L(v_r)$ maintains three arrays: a position array $pos(v_r)$, a distance array $dis(v_r)$, and a border vertex array $brd(v_r)$, as shown in Fig. 7(b). The border vertex array $brd(v_r)$ records all border vertices of VB_i. Specifically, we take the ID of v_r as the first element of $pos(v_r)$, i.e., $pos(v_r)[1] = id(v_r)$. Note that $pos(v_r)$ is used to identify the border vertices of VB_i. In addition, we use it to check if two low frequency vertices both are in the tree rooted at v_r. The remaining elements of $pos(v_r)$ and $dis(v_r)$ are the same as for wH2H—they store the positions and distances of corresponding border vertices. To ensure the correctness of shortest path distance computations of vertex pairs in T_i, we consider all the border vertices of VB_i as ancestors of the root v_r. Therefore, the tree height $h(T_i)$ increases by $|VB_i|$. For a non-root node $X(v)$, the label $\mathcal{L}_L(v)$ maintains two arrays: a position array $pos(v)$ and a distance array $dis(v)$, as shown in Fig. 7(b). Like the root node $X(v_r)$, the first element of $pos(v)$ is the ID of v_r.

As does wH2H, WCF computes the distance array in a top-down manner. Specifically, the distance array of each border vertex is computed by $\mathcal{L}_H$. The distance arrays of remaining low frequency vertices are computed in the same way as for wH2H. Fortunately, the index construction of each tree is mutually exclusive, which enables the parallel computation.

Theorem 2. *The space cost of WCF is* $O(\omega^* \cdot n_H \log n_H + n_L \cdot h)$.

Proof. For $v \in V_H$, the space complexity of $\mathcal{L}_H(v)$ is $O(\omega^* \cdot \log n_H)$. For $v \in V_L$, the space complexity of $\mathcal{L}_L(v)$ is $O(h)$.

So the space complexity of WCF is $O(n + \omega^* \cdot n_H \log n_H + n_L \cdot h) = O(\omega^* \cdot n_H \log n_H + n_L \cdot h)$. □

C. Query Processing

To answer a shortest path distance query $q(s,t)$, we classify queries into three categories: (1) H-H queries. Both s and t are high frequency vertices, where $s \in V_H$ and $t \in V_H$. (2) H-L queries. Here, s is a low frequency vertex and t is a high frequency vertex, where $s \in V_L$ and $t \in V_H$. (3) L-L queries. Both s and t are low frequency vertices, where $s \in V_L$ and $t \in V_L$.

Lemma 1. *Given two vertices $s, t \in V_H$, $\mathsf{Dist}(s,t)$ computed using $\mathcal{L}_H(s)$ and $\mathcal{L}_H(t)$ is correct.*

Proof. When we remove a vertex in RelaxedMinDegree, the remaining graph G^* is distance preserving [9]. For any pair of vertices $s, t \in V_H$, we have $\mathsf{Dist}_{G^*}(s,t) = \mathsf{Dist}_G(s,t)$. □

For an H-H query, $\mathsf{Dist}(s,t)$ computed using $\mathcal{L}_H(s)$ and $\mathcal{L}_H(t)$ is correct according to Lemma 1.

For an H-L query, we assume that T_i is the tree of $X(s)$ and VB_i is the set of border vertices of T_i. We first find the root v_r of T_i in $pos(s)[1]$. Then we determine whether t is a border vertex of T_i by traversing $brd(v_r)$. If t is a border vertex, $X(t)$ is an ancestor of $X(s)$ in T_i, and $\mathsf{Dist}(s,t)$ is obtained from $\mathcal{L}_L(s)$ directly. Otherwise, for each border vertex $u \in VB_i$, we obtain $\mathsf{Dist}(s,u)$ directly and compute $\mathsf{Dist}(u,t)$ by using $\mathcal{L}_H(u)$ and $\mathcal{L}_H(t)$. Finally, we have $\mathsf{Dist}(s,t) = \min_{u \in VB_i}\{\mathsf{Dist}(s,u) + \mathsf{Dist}(u,t)\}$.

Lemma 2. *Given two nodes $X(s)$ and $X(t)$ in the same tree, $\mathsf{Dist}(s,t)$ computed using $\mathcal{L}_L(s)$ and $\mathcal{L}_L(t)$ is correct.*

Proof. As s and t are symmetric, we only consider s. To prove the lemma, we only need to prove the correctness of $dis(s)$. Let G_i be a subgraph of G where the border vertices are VB_i. For any $s' \in anc(s)$, if $\mathsf{SP}(s,s')$ does not pass any border vertex, $\mathsf{Dist}_{G_i}(s,s') = \mathsf{Dist}_G(s,s')$. If $\mathsf{SP}(s,s')$ traverses the border, it must traverse at least two border vertices. Without loss of generality, suppose $\mathsf{SP}(s,s') = \langle s, \ldots, u, \ldots, u', \ldots, s' \rangle$ traverses two border vertices u, u'. As we use $\mathcal{L}_H$ to compute $\mathsf{Dist}(u,u')$, we have $\mathsf{Dist}_{G_i}(u,u') = \mathsf{Dist}_G(u,u')$. Therefore, $\mathsf{Dist}_{G_i}(s,s') = \mathsf{Dist}_{G_i}(s,u) + \mathsf{Dist}_G(u,u') + \mathsf{Dist}_{G_i}(u',s') = \mathsf{Dist}_G(s,s')$. So $dis(s)$ is computed correctly in a top-down manner. □

For an L-L query, we assume that T_i and T_j are the trees of s and t, respectively. We first determine whether s and t are in the same tree. If $T_i = T_j$, we compute $\mathsf{Dist}(s,t)$ in the same way as in wH2H according to Lemma 2. Otherwise, we employ a hop pushing on-the-go process that pushes the labels of border vertices in T_i and T_j to s and t, respectively, and increases $spd(s)$ and $spd(t)$ with the corresponding distances from s and t to the border vertices. Finally, we compute $\mathsf{Dist}(s,t)$ based on Eq. 1.

Theorem 3. *The time costs of processing an H-H query, an H-L query, and an L-L query are $O(\omega^* \cdot \log n_H)$, $O(\omega_{max} \cdot \omega^* \log n_H)$, and $O(\omega_{max} \cdot \omega^* \log n_H)$, respectively.*

v_1	$(v_1,0),(v_3,3)$	v_9	$(v_3,2),(v_9,0)$
v_2	$(v_1,5),(v_2,0),(v_3,7)$	v_{12}	$(v_1,6),(v_2,1),(v_3,8),(v_{12},0)$
v_3	$(v_3,0)$	v_{22}	$(v_1,6),(v_2,1),(v_3,8),(v_{22},0)$

(a) Labels of Core

$\mathbf{V}_L$		**pos**	**dis**	**brd**
root	v_4	4, 1, 2, 3, 4	6, 4, 2, 0	1, 3, 9
	v_{19}	19, 1, 2, 3, 4	1, 5, 2, 0	1, 2, 3
	v_{21}	21, 1, 2	1, 0	12
others	v_7	19, 1, 5, 6	1, 4, 4, 2, 2, 0	
	v_{14}	4, 3, 5, 6	5, 2, 2, 4, 1, 0	
	v_{23}	19, 6, 7	3, 6, 6, 4, 4, 2, 0	
	v_{24}	19, 6, 7	3, 6, 6, 4, 4, 2, 0	
	...	.....	.....	

(b) Labels of Forest

Fig. 7. Labels of Core and Forest in WCF

Proof. For an H-H query, the time cost is directly proportional to the label size of $\mathcal{L}_H$, so the time cost is $O(\omega^* \cdot \log n_H)$. For an H-L query, the worst case is that t is not a border vertex of T_i. For each border vertex u, the time cost of computing $\mathsf{Dist}(u,t)$ is $O(\omega^* \cdot \log n_H)$. As the number of border vertices $|VB_i| \leq \omega_{max}$, the time cost is $O(\omega_{max} \cdot \omega^* \cdot \log n_H)$. For an L-L query, the worst case is that $X(s)$ and $X(t)$ are in different trees. The time cost of the hop pushing on-the-go process is $O(\omega_{max} \cdot \omega^* \cdot \log n_H)$. The computation of $\mathsf{Dist}(s,t)$ is the same as for the H-H query. Then the time cost of an L-L query is $O(\omega_{max} \cdot \omega^* \cdot \log n_H + \omega^* \cdot \log n_H) = O(\omega_{max} \cdot \omega^* \cdot \log n_H)$. □

Example 7. *Given the workload Q in Example 3, we show the labels in Fig. 7 when $\beta = 0.1$ and $\omega_{max} = 5$. We illustrate three types of queries. For the H-H query $q(v_3, v_9)$, we return $\mathsf{Dist}(v_3,v_9) = spd(v_3)[1] + spd(v_9)[1] = 0 + 2 = 2$. For the H-L query $q(v_3, v_{14})$, we find the root node $X(v_4)$ in $pos(v_{14})[1]$. Then we get $brd(v_4)[2] = 3$ by traversing $brd(v_4)$ and the corresponding position is 2, i.e., v_3 is a border vertex in the same tree as v_{14}, so we have $\mathsf{Dist}(v_3,v_{14}) = dis(v_{14})[2] = 2$. For the H-L query $q(v_2, v_{14})$, as v_2 is not in $brd(v_4) = \{1,3,9\}$, we compute $\mathsf{Dist}(v_2,v_1) = 5, \mathsf{Dist}(v_2,v_3) = 7$, and $\mathsf{Dist}(v_2,v_9) = 9$. So $\mathsf{Dist}(v_2,v_{14}) = \min\{5+5, 7+2, 9+2\} = 9$. For the L-L query $q(v_7, v_{14})$, we find $pos(v_7)[1] \neq pos(v_{14})[1]$, so we have to push labels of the boundary vertices to v_7 and v_{14}. The results are shown in Fig. 8. Then we get $\mathsf{Dist}(v_7,v_{14}) = \min\{4+2, 1+5\} = 6$. For the L-L query $q(v_{23}, v_{24})$, we find $pos(v_{23})[1] = pos(v_{24})[1] = 19$ and $\mathsf{LCA}(v_{23},v_{24}) = X(v_7)$. Therefore, we compute $\mathsf{Dist}(v_{23},v_{24}) = \min\{3+3, 4+4, 2+2\} = 4$ based on Eq. 5.*

We have $|\mathcal{L}_H(v_3)| = 1$, $|\mathcal{L}_H(v_1)| = 2$, $|\mathcal{L}_H(v_9)| = 2$, $|\mathcal{L}_H(v_2)| = 3$, $|\mathcal{L}_H(v_{12})| = 4$, and $|\mathcal{L}_H(v_{22})| = 4$. So $\mathsf{cost}(Q) = 1 \times 5 + 2 \times 4 + 2 \times 3 + 3 \times 2 + 4 \times 1 + 4 \times 1 = 33$.

D. WCF-Variant Method

The main performance bottleneck of L-L queries is that we have to traverse the border vertices and push their labels at the query stage since different trees are involved. Therefore,

V_L		pos	dis	(hop, spd)
root	v_4	4, 1, 1	0	$(v_1,6),(v_3,4),(v_9,2)$
	...	.....	.....	.....
others	v_5	4, 0, 1, 2, 3, 4	5, 4, 2, 0	$(v_1,4),(v_3,7),(v_9,7)$
	v_7	19, 1, 2, 3	2, 2, 0	$(v_1,1),(v_2,4),(v_3,4)$
	v_{14}	4, 1, 2, 3	4, 1, 0	$(v_1,5),(v_3,2),(v_9,2)$
	v_{23}	19, 0, 3, 4	4, 4, 2, 0	$(v_1,3),(v_2,6),(v_3,6)$
	v_{24}	19, 0, 3, 4	4, 4, 2, 0	$(v_1,3),(v_2,6),(v_3,6)$
	...	.....	.....	.....

Fig. 8. Labels in WCF-Variant

we develop a WCF-Variant that performs the hop pushing in the preprocessing stage in parallel. The difference from WCF is that each low frequency vertex saves labels of the border vertices. This trade-off accelerates the L-L query processing while increasing the space overhead and preprocessing time slightly.

Example 8. *The labels of low frequency vertices constructed by WCF-Variant are shown in Fig. 8. Note that the second element in pos(v) captures whether $X(v)$ contains high frequency vertices. As we do not have to compute the boundary vertices, the root node v_4 does not store brd, and the lengths of pos(v_4) and dis(v_4) are shortened. For H-H queries, H-L queries, and L-L queries that cover different trees, we use Eq. 1 directly to compute distance. For L-L queries where two vertices are in the same tree, we have two cases: (1) For $q(v_{17}, v_{18})$, as* $\mathsf{LCA}(v_{17}, v_{18}) = X(v_5)$ *and pos$(v_5)[2] = 0$, we compute* $\mathsf{Dist}(v_{17}, v_{18})$ *using Eq. 5. (2) For $q(v_{23}, v_{24})$, where* $\mathsf{LCA}(v_{23}, v_{24}) = X(v_7)$ *and pos$(v_7)[2] = 1$, we need to use both Eq. 1 and Eq. 5 to find the minimum value, i.e.,* $\mathsf{Dist}(v_{23}, v_{24}) = \min\{4+4, 2+2, 3+3, 6+6, 6+6\} = 4$.

E. WCF in Directed Road Networks

To handle directed road networks, the main difference is that we need to process the in-edges and the out-edges, respectively. For $v \in V_H$, $\mathcal{L}_H(v)$ includes in-labels $\mathcal{L}_{in}(v)$ and out-labels $\mathcal{L}_{out}(v)$. For $v \in V_L$, $\mathcal{L}_L(v)$ maintains two distance arrays $dis_{out}(v)$ and $dis_{in}(v)$, where $dis_{out}(v)$ records forward distances from v to all its ancestors and $dis_{in}(v)$ saves corresponding reverse distances.

For the query processing, we answer H-H queries in the same way as the directed PLL. For an H-L query $q(s,t)$, where $s \in V_L$ and $t \in V_H$, there are two cases. If t is a border vertex, we directly return $\mathsf{Dist}(s,t)$ from $dis_{out}(s)$. Otherwise, the distance is computed by $dis_{out}(s)$, $\mathcal{L}_{in}(t)$, and $\mathcal{L}_{out}(u)$, where $u \in VB_i$. For an L-L query $q(s,t)$, there are also two cases. If s and t are in the same tree, we compute the distance with $dis_{out}(s)$ and $dis_{in}(t)$. Otherwise, we push the in-labels and out-labels of border vertices to s and t. Then we compute $\mathsf{Dist}(s,t)$ in the same way as the directed PLL.

VI. Temporal Locality-Aware Partitioning

We observe that the query frequency distributions of most vertices exhibit a temporal locality. That is, adjacent time slots have similar distributions. Since WCF has demonstrated the capability of reducing the workload query cost, if the query frequency distribution is more fine-grained, the cost can be further reduced. To exploit the temporal locality, we divide one day into a sequence of time slots, i.e., $T = \langle t_1, \ldots, t_{|T|} \rangle$. In experiments, each time slot is set to 15 minutes, so we have 96 time slots. We study a time interval partitioning problem (TIP) that partitions one day into a sequence of K time intervals, $T = \langle I_1, \ldots, I_K \rangle$, where each interval I_k is a sequence of time slots. Instead of building an index based on the query frequency distribution of the whole day, we construct an index for each time interval. Let $f_{i,k}$ be the query frequency of v_i in time interval I_k and $\mathsf{cost}(v_i, I_k)$ be the query cost of v_i with the index of time interval I_k, the workload query cost in Eq. 3 can be reformed as follows.

$$\mathsf{cost}(Q) = \sum\nolimits_{I_k \in T} \sum\nolimits_{v_i \in V_Q} \mathsf{cost}(v_i, I_k) \cdot f_{i,k} \tag{8}$$

If each time interval contains only one time slot, the cost can be mostly reduced since each time interval has a most fine-grained query frequency distribution. However, this is not applicable in practice since we have to maintain indexes for 96 time intervals. To better trade off between the workload query cost and the space consumption, we develop a reinforcement learning based time interval partitioning (RL-TIP) algorithm that chooses appropriate time slots for partitioning.

A. Greedy Time Interval Partitioning

For better understanding, we first introduce the greedy time interval partitioning (GTP) algorithm. GTP sequentially scans the time slots and computes an indicator value Δ for each time slot, compares it with a threshold δ to decide whether to partition at the time slot. Let Q_{t_j} be the workload of time slot t_j. Initially, we construct an index for the first time slot t_1 based on Q_{t_1}. The indicator value for each time slot t_j is computed as follows.

$$\Delta_j = \frac{|\mathsf{cost}(Q_{t_j}) - \mathsf{cost}(Q_{t^*})|}{\mathsf{cost}(Q_{t^*})} \tag{9}$$

where t^* is the time slot based on whom the most recent label index is built, and $\mathsf{cost}(Q_{t_j})$ and $\mathsf{cost}(Q_{t^*})$ are the query costs of Q_{t_j} and Q_{t^*} with the most recent label index. Eq. 9 implies that if the costs of Q_{t_j} and Q_{t^*} are far different, t_j and t^* should be placed in different time intervals.

If $\Delta_j \geq \delta$, a time interval $[t^*, t_j)$ is formed, and a new label index is constructed based on Q_{t_j}. Otherwise, we process the next time slot. The procedure is repeated until all time slots are scanned.

B. Time Interval Partitioning as a MDP

The GTP algorithm is a handcraft method that heavily relies on the chosen δ. To achieve better partitioning, we model the TIP problem as a *markov decision process* (MDP) [26] and employ the reinforcement learning technique to solve the problem. The key elements are as follows.

State. A state is a five-element tuple $s = (t_j, \rho^*, \rho_j, D_j^*, C)$ used for decision making. Element t_j is the current time slot.

Elements ρ^* and ρ_j are the query frequencies of all vertices at t^* and t_j. Element D_j^* is the Jensen-Shannon divergence [27] between the query frequency distributions of t^* and t_j. Element C is the number of time intervals that have already obtained. Note that when $j = |T|$, the state is the terminal state.

Action. An action $a \in \{0, 1\}$ is an indicator that decides whether to partition. When $a = 1$, we partition at time slot t_j, and a time interval $[t^*, t_{j-1})$ is formed. Moreover, a new label index is constructed based on the frequency distribution of t_j. When $a = 0$, we do not partition.

Reward. A reward denotes the quality of an action and is defined as $r_j = -\sum_{v_i \in V} f_{i,j} \cdot \mathsf{cost}^*(v_i)$, where $f_{i,j}$ is the query frequency of v_i at t_j, and $\mathsf{cost}^*(v_i)$ is the query cost of v_i based on the index built at t^*. Higher reward indicates better action.

Transition. A transition is a four-element tuple (s, a, s', r) indicating that we take an action a under the state s, receive a reward r, and move to the next state s'.

The target of solving MDP is to learn an action-value function for decision making. The action-value function takes the state information as input and outputs the value of actions, and then select the action with the maximum value to maximize the cumulative reward, which is computed as $\sum_{i=0}^{|T|-j} \lambda^k \cdot r_{j+i}$, where λ is the discount factor for future rewards.

C. Reinforcement Learning based Time Interval Partitioning

DQN [28] is a typical reinforcement learning method that uses a neural network to estimate the action-value function $Q(s, a; \theta)$. DQN contains two neural networks. One is the behavior-net $Q(s, a; \theta)$ that estimates the action-value function. $Q(s, a; \theta)$ takes a state as an input and output actions with Q-value. The action with the maximum Q-value is the best choice to take. The other is the target-net $\hat{Q}(s, a; \theta^-)$ that helps train the behavior-net. Besides, DQN maintains a memory replay array that is used to store the recently obtained transitions.

We proceed to introduce RL-TIP that trains the DQN to estimate the action-value function $Q(s, a; \theta)$. The RL-TIP algorithm takes a query workload as input and outputs the trained behavior-net $Q(s, a; \theta)$. First, the algorithm initializes the capacity of replay memory M. Then, it initializes the behavior-net $Q(s, a; \theta)$ with random weights θ and initializes the target-net $\hat{Q}(s, a; \theta^-)$ with weights $\theta^- = \theta$. Next, the behavior-net is trained for m_e episodes. In each episode, it extracts a part of the workload for training and initializes the first state s_1. For time slot t_j and state s_{t_j}, it uses the behavior-net to choose an action a_{t_j} using the ϵ-greedy policy (i.e., choose $a_{t_j} = \arg\max_a Q(s_{t_j}, a; \theta)$ with probability $1 - \epsilon$ and choose an random action with probability ϵ). Then, it receives a reward r_{t_j} and arrive the next state $s_{t_{j+1}}$. The transition $(s_{t_j}, a_{t_j}, s_{t_{j+1}}, r_{t_j})$ is stored in the replay-memory M, and a batch of transitions is sampled from replay memory M to train the network with stochastic gradient descent method. The loss function is defined as follows.

$$L(\theta) = [y_t - Q(s_t, a_t; \theta)]^2, \tag{10}$$

TABLE II
ROAD NETWORKS

Road Networks	# of Vertices	# of Edges
Haikou (HK)	10,191	27,020
Manhattan (MH)	17,084	42,650
Chengdu (CD)	72,745	190,996
NewYork (NY)	264,346	730,100
Florida (FL)	1,070,376	2,712,798
California (CA)	1,890,815	4,657,742

where y_t is the target value that is computed as follows.

$$\begin{cases} y_t = r_t & \text{terminal state} \\ y_t = r_t + \lambda \cdot \max\limits_{a'} \hat{Q}(s_{t+1}, a'; \theta^-) & \text{otherwise} \end{cases} \tag{11}$$

The parameters of target-net θ^- are periodically synchronized with those of the behavior-net θ.

As DQN only has a few layers, it takes a little time to compute the Q-value. So the time cost of RL-TIP is similar to that of GTP.

VII. EXPERIMENTS

A. Experimental Settings

All algorithms are implemented in C++ and compiled with GNU G++ 9.3.0, except that RL-TIP and BnB are implemented in Python 3. All experiments are conducted on a machine with an AMD Ryzen 7 PRO 4750G CPU and 32GB main memory running Linux (Ubuntu 20.04 LTS, 64bit).

Datasets. We conduct experiments in six real-world road networks [29], [30] with statistics shown in Table II. The query workloads are obtained as follows: (1) Real datasets: we extract query workloads for 30 consecutive days from real-world historical taxi order statistics [12], [31] by matching the start and end points to vertices in the networks for HK, MH, CD, and NY. We use real-world query datasets from the 1st day to the 20th day to train and validate the GraphWaveNet model [19], and predict the query frequency distribution. (2) Synthetic datasets: to study the scalability in large scale road networks, we generate 10 days of query workloads for FL and CA. The query frequency distributions and query distance distributions are generated by obeying those of real-world workloads. Specifically, to derive the query frequency of each vertex, we increment f_s and f_t when a query $q(s, t)$ is issued.

Performance metrics. (1) Average query time: we report the average query time of 1 million queries, and half of the query data is used for warm-up before testing. (2) Index construction time: we record the CPU clock time for index construction in main memory. (3) Index size: we use 32-bit integers to represent values in indexes and store the indexes in binary file format, which is consistent with previous studies.

B. Self-evaluation

Parameter study. We conduct parameter studies on all datasets with wPLL-$\beta = 0.1$, wH2H-$\gamma = 0.1$, wH2H-$\eta = 30$, WCF-$\beta = 0.1$, and WCF-$\omega_{max} = 30$ by default. Due to the space limitation, we only present the results for Manhattan.

TABLE III
WCF VS. WCF-VARIANT

Metric	Approach	HK	MH	CD	NY
Index Construction Time (ms)	WCF	19.9	54.3	506.3	1745.0
	WCF-Variant	49.13	126.9	1278.7	5008.2
Index Size (MB)	WCF	2.3	5.4	15.1	68.5
	WCF-Variant	14.2	37.9	125.2	571.5
Average Query Time (μs)	WCF	4.9	7.9	12.8	39.5
	WCF-Variant	1.2	1.6	2.2	2.3

From Figs. 9(a)–(c) where the x-axis denotes the parameter values scaled to $[0, 1]$, we observe that: (1) The index size and the index construction time for WCF increase as ω_{max} increases, while the average query time first decreases and then increases. The reason is that ω_{max} affects the scale of the core-forest structure and especially the sparsity degree of the core, which in turn influences the performance. So the overall label size increases while the label size of high frequency vertices decreases, which results in decreased average query time. For the parameters β and γ, the trends and the reasons for them are similar to those for ω_{max}. (2) For η, the index size of wH2H decreases first and then exceeds that of $\eta = 1$ with appropriate extra time overhead, which demonstrates the effectiveness of our block partitioning technique. As the results of ω_{max} and η are similar on other datasets, WCF-ω_{max} $= 30$ and wH2H-η $= 30$ are set as relatively fixed values on all road networks.

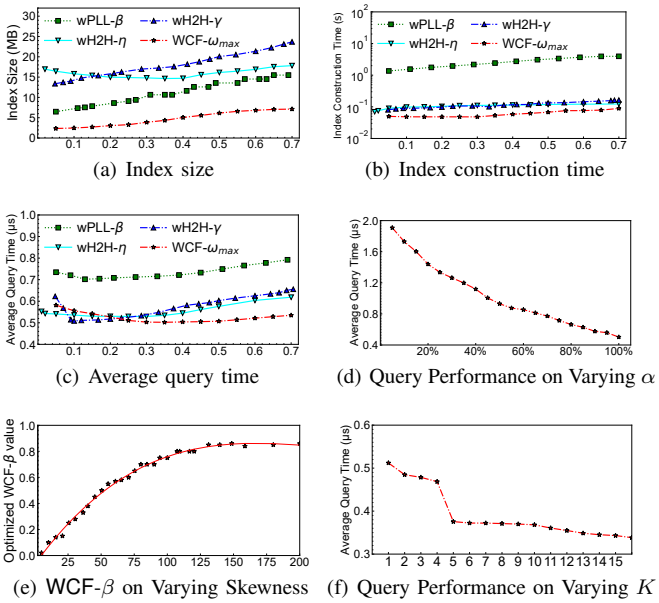
Parameters chosen based on skewness. For WCF-β, we select 30 query workloads with various skewness [32] from real-world datasets. For each workload, we tune WCF-β and record the value when WCF has the best query performance. From Fig. 9(e), we find that WCF-β is positively correlated to the skewness of workloads. As the fitting curve in red shows, given a query workload, we can set an approximately optimized value according to its skewness. Similarly, wH2H-γ and wPLL-β are set in a similar way.

Effect of α. Fig. 9(d) shows the average query time of WCF on varying α value. We can see that WCF has best performance for $\alpha = 100\%$. Therefore, we set $\alpha = 100\%$ by default, which avoids the hyper-parameter tuning and the overhead of vertex sorting by query frequency.

WCF vs. WCF-Variant. The comparison results between WCF and WCF-Variant are shown in Table III. It indicates that WCF-Variant consumes 8 times less query time for L-L queries, while consumes 2.5 times more index construction time and 7.5 times more space than WCF on average.

C. Exploiting Spatial Skew

Query time. (1) On real-world workloads. We first use the predicted query frequency distribution to construct indexes for all 5 methods. Then we test the average query time on the real-world query workloads from the 21st to the 30th day, denoted by $Q_1, Q_2, ..., Q_{10}$. As shown in Figs. 10(a)–(d), we make the following observations: 1) By exploiting the spatial skew, wPLL, wH2H, and WCF are much faster than PLL as they are workload-aware. The average query time improvement ratios are 12%, 32.8%, and 35.3%, respectively.

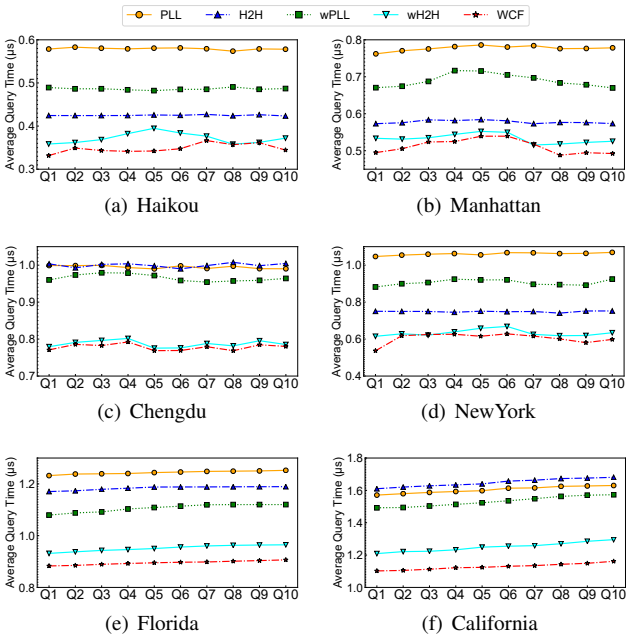


(a) Index size (b) Index construction time
(c) Average query time (d) Query Performance on Varying α
(e) WCF-β on Varying Skewness (f) Query Performance on Varying K

Fig. 9. Analysis of Parameters

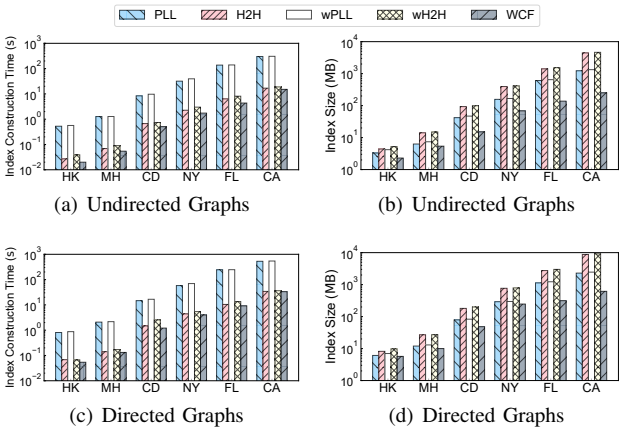
2) Compared with H2H, the improvement ratios of wH2H and WCF are 14.3% and 17.7%, respectively. PLL and wPLL are slower than H2H on some datasets because H2H preserves more distance information in the index than PLL does. (2) Varying query distributions. To test the performance for the cases when the queries do not follow the distribution in the predicted query workload, we generate two query workloads Q_e (equal) and Q_r (reverse) where $V_{Q_e} = V_{Q_r} = V_Q$. In Q_e, the query frequency of each vertex is equal. In Q_r, we reverse the query frequencies of all vertices, e.g., the vertex with most query frequency in Q becomes the vertex with smallest query frequency while keeping the value unchanged. Then we test the average query time of Q, Q_e, and Q_r with the index constructed by the predicted query frequency. As Fig. 13(a) shows, query time of Q_e and Q_r is increased by 34.86% and 45.45% on average compared to that of Q. (3) Different query types. We generate 1 million queries for each type of queries (H-H, H-L, and L-L). We record the average query time of each type. As Fig. 13(a) shows, the experimental results are consistent with our analysis. (4) On synthetic workloads. Figs. 10(e)–(f) shows the average query time on 10 days of workloads for FL and CA, denoted by $Q_1, Q_2, ..., Q_{10}$. WCF has the lowest query times and exhibits the best scalability.

Index construction time. Table IV shows the speedup ratio of parallel index construction using τ over single threaded index construction for WCF. The speedup ratio first increases and then decreases slightly because additional overhead is incurred with the increase of threads. Therefore, we set $\tau = 5$ for index construction. Fig. 11(a) compares the index construction times of all 5 methods in the 6 road networks. WCF is clearly best on all datasets, and decreases the index construction time by 95.4% and 22.7% on average compared with PLL and



(a) Haikou (b) Manhattan (c) Chengdu (d) NewYork (e) Florida (f) California

Fig. 10. Comparison on Average Query Time



(a) Undirected Graphs (b) Undirected Graphs (c) Directed Graphs (d) Directed Graphs

Fig. 11. Index Construction Time and Index Size

TABLE IV
PARALLEL INDEX CONSTRUCTION SPEEDUPS

Networks	$\tau = 3$	$\tau = 5$	$\tau = 10$	$\tau = 15$	$\tau = 20$
Haikou	18%	25%	12%	10%	8%
Manhattan	30%	38%	25%	20%	15%
Chengdu	30%	40%	45%	40%	40%
NewYork	38%	53%	60%	57%	54%
Florida	30%	40%	45%	40%	40%
California	38%	53%	60%	57%	54%

H2H. The efficiency of WCF comes from the hierarchical structure and parallel computation. wPLL and wH2H consume more time than PLL and H2H, respectively, since the order of wPLL and the tree decomposition of wH2H are suboptimal.

Index size. As Fig. 11(b) shows, WCF has the smallest index size on all datesets, which is 53.7% and 76.7% smaller than those of PLL and H2H on average. The reasons are: (1) Compared to PLL, WCF constructs labels on a small overlay graph, which reduces the index size considerably; (2) Compared to H2H, WCF avoids storing long distance arrays by transforming a single tree decomposition to a forest with small tree heights. Moreover, the index sizes of wPLL versus PLL and H2H versus wH2H are comparable, with the workload-aware indexes consuming slightly more space.

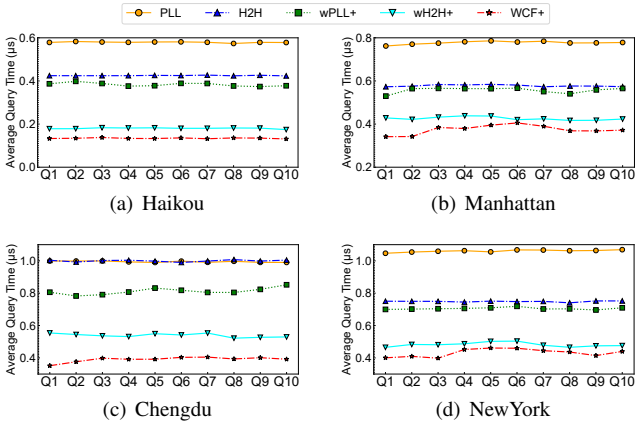
D. Exploiting Both Spatial Skew and Temporal locality

To evaluate the performance of RL-TIP, we implement wPLL+, wH2H+, and WCF+ where wPLL+ is an adapted version of wPLL with applying RL-TIP, so as wH2H+ and WCF+. DQN of RL-TIP is a fully connected neural network with two hidden layers of size 64 and 128. We use the Adam optimizer with learning rate 0.001. The batch size is 128, $\epsilon = 0.95$, $\lambda = 0.9$, and the synchronization period is set as 300 training batches.

Parameter Chosen on K. Fig. 9(f) represents the average query time for different K values. With the increase of K, the query time first decreases sharply. When K exceeds 5, the downward trend slows. To balance the index construction overhead and query performance, we set $K = 5$ by default.

Query time. We set each time slot to be 15 minutes and partition one day into 5 time intervals with applying RL-TIP. We construct indexes for each interval. We study the average query time for the real-world query workloads in the corresponding intervals from the 21st day to the 30th day and have the following observations: (1) In Fig. 12, we see that WCF and wH2H outperform the other competitors substantially on all datasets. The reason is that after partitioning, our proposed methods are able to cope with high frequency vertices in each time interval more effectively with fine-grained frequency distributions. In addition, the skew in the query workloads in each time interval increases after partitioning, and our methods perform better on more skewed data. Compared with PLL, the average improvement ratios of wPLL+, wH2H+, and WCF+ are 28.9%, 53.8%, and 62.3%, respectively. Compared with H2H, the average improvement ratios of these methods are 9.4%, 41.4%, and 51.4%, respectively. (2) The average query time of wPLL+, wH2H+, and WCF+ over those without applying RL-TIP is reduced by 19.6%, 31.7%, and 41.7%, respectively. The results offer evidence that considering both spatial skew and temporal locality is more effective than only considering spatial skew.

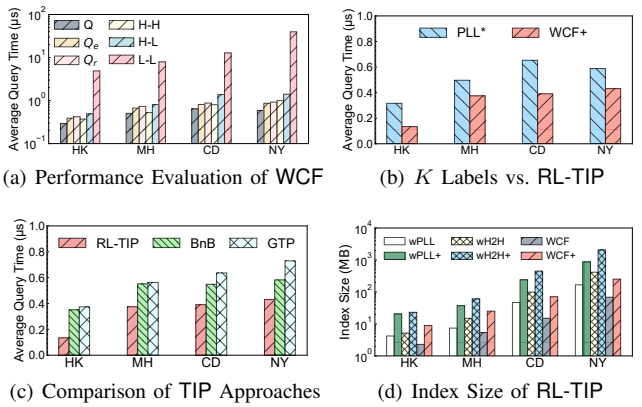
Improvement by RL-TIP. To further demonstrate the effectiveness of RL-TIP under the same indexing overhead, we generate label sets using significantly different ordering schemes {betwenness, degree, random, query frequency, wPLL}. At query stage, we check all K sets of 2-hop labels and choose the label set with smallest $|L(s)|+|L(t)|$ to answer $q(s,t)$. We call this method PLL*. From Fig. 13(b), we can see WCF+ performs best on all datasets. The average query improvement ratio compared with PLL* is 37.3%.



(a) Haikou (b) Manhattan

(c) Chengdu (d) NewYork

Fig. 12. Comparison of Average Query Time with RL-TIP



(a) Performance Evaluation of WCF (b) K Labels vs. RL-TIP

(c) Comparison of TIP Approaches (d) Index Size of RL-TIP

Fig. 13. Performance and Effectiveness Evaluation

Comparison of TIP approaches. We study the effectiveness of three time interval partitioning methods RL-TIP, GTP, and BnB [33], where BnB is a state-of-the-art algorithm for change detection. Fig. 13(c) shows the comparison of average query time. RL-TIP performs the best, which offers evidence that applying reinforcement learning to solve TIP is effective.

Index Overhead. As Fig. 13(d) shows, the index size of wPLL+, wH2H+, and WCF+ on average are 5 times, 4.5 times, and 4.2 times larger than that of the base versions, respectively. The index construction time is about the same as that of base version with using parallel computation.

E. Performance on Directed Road Networks

We study the performance of our proposed methods on directed networks. As the query regularity is similar to what we have seen for undirected networks, we only show index construction time and size in Figs. 11(c)–(d). We observe that both the space and time overheads are about two times higher than for undirected networks. It is because all algorithms traverse the in-edges and out-edges during run time and record in-distances and out-distances for indexes.

VIII. Related Work

Shortest path distance problem. The computation of shortest path distances has been studied for decades. Dijkstra's algorithm [34] is the classical algorithm but has long running times when two argument vertices are far away. Hence many studies focus on improving query time by indexing the underlying road-network graph, such as hierarchy-based and hop-based solutions. HH [35] establish hierarchical structures among roads to reduce the search scope. CH [3] are similar but introduce a shortcuts when contracting vertices.

Hop-based labeling. 2-hop labeling is first proposed by Cohen et al. [6] whose key aspect is how to choose hops to satisfy the set cover property. Subsequent studies focus on reducing the index size and indexing time. Takuya et al. [8] propose a hop pushing method with pruning, which improves scalability. Akiba et al. [36] extract highways from road networks to reduce the preprocessing time. Li et al. [20] provide a comprehensive comparison of the 2-hop based techniques. Several studies [9], [37], [38] use tree decomposition during indexing. Lakhotia et al. [39] exploit trees to parallelize Pruned Landmark Labeling. ParDiSP [40] partitions the road network into components and supports both shortest distance and path queries. Storandt [41] proposes a region-aware route planning framework by extracting subgraphs for user's region-of-interest. IS-Label [7] organizes graphs into layers and forms a hierarchical structure that obeys the vertex dependence. However, WCF applies RelaxedMinDegree to extract a core-forest structure and computes labels in a top-down manner, which is much faster than the bottom-up manner of IS-Label. More importantly, the core-forest structure can facilitate parallel computation and decrease the index size substantially. WCF answers queries with 2-hop labels, which is much faster than the bi-Dijkstra search in IS-Label.

IX. Conclusion and Future Work

We provide a new solution that utilizes the skew in query workloads to accelerate shortest path distance querying in road networks. First, we retrofit the existing PLL and H2H methods as workload-aware hop labeling methods named wPLL and wH2H, respectively. Based on these, we propose a workload-aware indexing approach called WCF. Next, we propose a reinforcement learning based solution called RT-TIP to enable our workload-aware indexes to adapt to dynamic changes in query distributions. The experimental study employs 6 widely-used datasets with corresponding real-world query statistics and offers evidence that the proposed wPLL, w-H2H, and WCF are capable of achieving 62% query processing speed-ups on average compared to competitors. In addition, when compared to PLL and H2H, WCF improves the index size by about 54% and 77%, and the index construction time by about 95% and 23%. The experimental results also offer evidence of the effectiveness of GTP and RL-TIP.

It is of great interest to study the workload query processing problem in time dependent road networks. However, given the complexity of this problem, we plan to study it in the future.

REFERENCES

[1] A. Goldberg and C. Harrelson, "Computing the shortest path: A* search meets graph theory," Tech. Rep., July 2004.

[2] P. Sanders and D. Schultes, "Engineering highway hierarchies," vol. 17, 09 2006, pp. 804–816.

[3] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *WEA*, vol. 5038, 2008, pp. 319–333.

[4] 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 *SIGMOD*, 2013, pp. 857–868.

[5] H. Bast, S. Funke, and D. Matijevic, "Ultrafast shortest-path queries via transit nodes," in *The Shortest Path Problem*, vol. 74, 2006, pp. 175–192.

[6] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," in *SODA*, 2002, pp. 937–946.

[7] A. W. Fu, H. Wu, J. Cheng, and R. C. Wong, "IS-LABEL: an independent-set based labeling scheme for point-to-point distance querying," *PVLDB*, vol. 6, pp. 457–468, 2013.

[8] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *SIGMOD*, 2013, pp. 349–360.

[9] D. Ouyang, L. Qin, L. Chang, X. Lin, Y. Zhang, and Q. Zhu, "When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks," in *SIGMOD*, 2018, pp. 709–724.

[10] W. Li, M. Qiao, L. Qin, Y. Zhang, L. Chang, and X. Lin, "Scaling distance labeling on small-world networks," in *SIGMOD*, 2019, pp. 1060–1077.

[11] R. Jin, Z. Peng, W. Wu, F. F. Dragan, G. Agrawal, and B. Ren, "Pruned landmark labeling meets vertex centric computation: A surprisingly happy marriage!" *CoRR*, vol. abs/1906.12018, 2019.

[12] [Online]. Available: https://gaia.didichuxing.com

[13] Y. Yano, T. Akiba, Y. Iwata, and Y. Yoshida, "Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths," in *CIKM*, 2013, pp. 1601–1606.

[14] I. Abraham, D. Delling, A. V. Goldberg, and R. F. F. Werneck, "Hierarchical hub labelings for shortest paths," in *ESA*, vol. 7501, 2012, pp. 24–35.

[15] B. Zheng, J. Wan, Y. Gao, Y. Ma, K. Huang, X. Zhou, and C. S. Jensen, "Workload-aware shortest path distance querying in road networks," *Technical Report*, https://bolongzheng.com/public/workload_SP.pdf.

[16] 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*, vol. 7, pp. 1203–1214, 2014.

[17] B. Zheng, Q. Hu, L. Ming, J. Hu, L. Chen, K. Zheng, and C. S. Jensen, "Soup: Spatial-temporal demand forecasting and competitive supply," *TKDE*, https://doi.org/10.1109/TKDE.2021.3110778.

[18] S. Guo, Y. Lin, H. Wan, X. Li, and G. Cong, "Learning dynamics and heterogeneity of spatial-temporal graph data for traffic forecasting," *TKDE*, 2021.

[19] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling," in *IJCAI*, 2019, pp. 1907–1913.

[20] Y. Li, L. H. U, M. L. Yiu, and N. M. Kou, "An experimental study on hub labeling based shortest path algorithms," *PVLDB*, vol. 11, pp. 445–457, 2017.

[21] D. Delling, A. Goldberg, T. Pajor, and R. Werneck, "Robust distance queries on massive networks," in *Microsoft Research*, 09 2014, pp. 321–333.

[22] M. A. Bender and M. Farach-Colton, "The LCA problem revisited," in *LATIN*, vol. 1776, 2000, pp. 88–94.

[23] H. L. Bodlaender, "Treewidth: Characterizations, applications, and computations," in *WG*, vol. 4271, 2006, pp. 1–14.

[24] M. Zhang, L. Li, W. Hua, and X. Zhou, "Efficient 2-hop labeling maintenance in dynamic small-world networks," in *ICDE*, 2021, pp. 133–144.

[25] M. Zhang, L. Li, W. Hua, R. Mao, P. Chao, and X. Zhou, "Dynamic hub labeling for road networks," in *ICDE*, 2021, pp. 336–347.

[26] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, ser. Wiley Series in Probability and Statistics, 1994.

[27] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing*, 2001.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[29] [Online]. Available: https://www.openstreetmap.org/

[30] [Online]. Available: http://www.diag.uniroma1.it/~challenge9/

[31] [Online]. Available: https://www1.nyc.gov/site/tlc/about/data.page

[32] Wikipedia, "Skewness," https://en.wikipedia.org/wiki/Skewness.

[33] B. Hooi and C. Faloutsos, "Branch and border: Partition-based change detection in multivariate time series," in *SDM*, 2019, pp. 504–512.

[34] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[35] P. Sanders and D. Schultes, "Highway hierarchies hasten exact shortest path queries," in *ESA*, vol. 3669, 2005, pp. 568–579.

[36] T. Akiba, Y. Iwata, K. Kawarabayashi, and Y. Kawata, "Fast shortest-path distance queries on road networks by pruned highway labeling," in *ALENEX*, 2014, pp. 147–154.

[37] T. Akiba, C. Sommer, and K. Kawarabayashi, "Shortest-path queries for complex networks: exploiting low tree-width outside the core," in *EDBT*, 2012, pp. 144–155.

[38] F. Wei, "TEDI: efficient shortest path query answering on graphs," in *SIGMOD*, 2010, pp. 99–110.

[39] K. Lakhotia, R. Kannan, Q. Dong, and V. K. Prasanna, "Planting trees for scalable and efficient canonical hub labeling," *PVLDB*, vol. 13, pp. 492–505, 2019.

[40] T. Chondrogiannis and J. Gamper, "Pardisp: A partition-based framework for distance and shortest path queries on road networks," in *MDM*, 2016, pp. 242–251.

[41] S. Storandt, "Region-aware route planning," in *W2GIS*, vol. 10819, 2018, pp. 101–117.