FAHL: An Efficient Labeling Index for Flow-Aware Shortest Path Querying in Road Networks

Tangpeng Dan*, Xiao Pan†, Bolong Zheng‡, Xiaofeng Meng*
*Renmin University of China, Beijing, China
†Shijiazhuang Tiedao University, Shijiazhuang, China
‡Huazhong University of Science and Technology, Wuhan, China
*{tangpengdan,xfmeng}@ruc.edu.cn, †smallpx@stdu.edu.cn, ‡bolongzheng@hust.edu.cn

***Abstract*—As a fundamental operation of location-based services, shortest path querying is widely adopted in real-time applications. Regrettably, most prior works overlook the impact of traffic-flow on shortest path querying. Taking traffic-flow into account is essential for finding a more convenient path through the Flow-Aware Shortest Path Querying (FSPQ). FSPQ faces the following challenges: (1) *index restriction*, existing indexes are only constructed by the relative spatial distance, if we leverage the traffic-flow to build the index, we can reduce the index size and improve its query efficiency. (2) *maintenance latency*, the traffic-flow and edges' weights undergo high-frequency changes with different traffic conditions, meaning that our index must be able to support high-frequency updates. To end this, we propose a novel Flow-Aware Hierarchical Labeling Index (FAHL) in this paper. In the index construction aspect, we propose a degree-flow joint ordering method to obtain the joint vertex ordering, and then build the index on it. In this way, FAHL can not only perceive both spatial distance and traffic-flow information but also reduce the index overhead during the query. In the index maintenance aspect, we propose Improved Structure Update (ISU) and Index Label Update (ILU) algorithms to support the index updating when high-frequency flow\weight changes. Moreover, a flow priority shortest path search algorithm with pruning query bounds is proposed to speed up the query processing. Extensive experiments demonstrate that our proposed method achieves *33.1%* speedup on average for the flow-aware shortest path querying compared to the state-of-the-art methods.**

I. Introduction

With the proliferation of navigation services (e.g., Google Maps[1] and Amap[2]) and smartphones, substantial amounts of spatial data have been generated. This data has fueled advancements in real-time route planning [1], [2], spatial privacy protection [3], [4], and other location-based applications [5], [6], significantly improving users' daily travel efficiency. At the core of these applications lies the fundamental operation of shortest path querying (SPQ), which has garnered attention from both academia and industry.

In the context of SPQ, traditional methods return the optimal planning path with the shortest distance or minimum traveling time on road networks [7], [8]. As city transportation systems evolve and user demands diversify, various road networks, such as large-scale networks [9], [10], dynamic networks [11], [12], and time-dependent networks [13], [14], have been proposed to make our data measurement close to the real-life environments. Compared with traditional static road networks, these complex networks introduce dynamic changes or travel time constraints to vertices and edges, making them more reflective of real-world conditions. Despite these advancements, current SPQ methods overlook the effects of traffic-flow, which makes it challenging to obtain the most convenient path in real-world scenarios.

TABLE I: An example traffic-flow list

Traffic-flow of each vertex at arrival time										
v_i	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	Q_u	Q_d
$Flow$	5	2	4	8	15	24	20	12	10	10

Fig. 1: A Shortest Path Querying with Traffic-flow

For example, Fig. 1 displays a real navigation map, and Table I gives the traffic-flow of each vertex. Traffic-flow on each vertex represents the number of vehicles passing through the vertex when a user arrives. Suppose a user near "Joy City" wishes to go to the "Capital Museum" and set the traffic-flow for each vertex at query time as 10. Traditional path-finding methods might suggest the red path $P_1 = \{Q_u, v_4, v_5, v_6, v_7, Q_d\}$ as the optimal route due to its shortest distance of 41. However, in the real world, each vertex has a different traffic-flow at different arrival times, making the green path $P_2 = \{Q_u, v_1, v_2, v_3, v_8, Q_d\}$ to be a potentially better choice with a traffic-flow of 43, compared to a traffic-flow of P_1 is 87. Therefore, flow-aware shortest path querying (FSPQ) emerges as the times require.

Corresponding author: Xiaofeng Meng (xfmeng@ruc.edu.cn).
[1] https://www.google.com/maps
[2] https://www.amap.com/

Existing navigation services (e.g., Google Maps and Amap) utilize real-time traffic-flow to improve their performance, however, they primarily consider the traffic-flow at the time of the query when planning routes. During navigation, the system can only alert users to incoming traffic congestion but cannot prevent it. In contrast, our FSPQ considers all dynamic updates (including traffic-flow and edge weights) from the query location to the destination of navigation, thereby preventing navigation errors and overcoming the limitations of existing applications. In this way, FSPQ is able to recommend paths that are more reflective of real-life conditions and help users avoid traffic congestion to reach their destinations more quickly. In addition, FSPQ can also support downstream tasks such as ridesharing recommendation [15], [16] and estimated time of arrival [17], [18].

Due to vehicles in one vertex can reach any other connected vertices, traffic-flow exhibits transitivity between different vertices, leading to mutual influence. Fortunately, existing technologies like deep learning methods [19], [20] are able to learn this spatial temporal correlation and give an accurate predicted traffic-flow. As this line of studies is orthogonal to our work, we adopt a well-recognized model PDFormer [20] to obtain an accurate traffic-flow. Based on it, we mainly focus on how to efficiently answer FSPQ in this work.

Motivation. By the observation from the FSPQ in real-world querying, we find that the FSPQ exhibits the following challenge.

1) *Index Restriction*. Existing SPQ indexes are only constructed by the relative spatial distance. Meaning that these methods cannot leverage the traffic-flow information. If we build the index by considering both spatial distance and traffic-flow, we can place the vertices with lower flow near the root, resulting in smaller label entries and index size. Thus, we can improve the query efficiency of the entire index.
2) *Maintenance Latency*. The traffic-flow and edges' weights undergo high-frequency changes due to different traffic conditions. Meaning that when flow\weight changes, the index must have the ability to complete update operations within a short time. Otherwise, the latency will cause error queries. The status indicates that we have to develop efficient index maintenance algorithms to support the index updating.

Therefore, if we can efficiently predict an accurate traffic-flow and improve the index of obtaining the spatial distance, the efficiency of FSPQ will be improved.

To address these challenges, we propose a novel Flow-Aware Hierarchical Labeling Index (FAHL) in this paper. FAHL fully utilizes spatial and temporal information. Specifically, we design a degree-flow joint ordering method to obtain the vertex ordering in FRN based on spatial distance and traffic-flow two dimensions and then construct the labeling index with tree decomposition and joint ordering. Leveraging accurate traffic-flow, FAHL determines the flow order of each vertex and computes label values with both degree and flow order. By fusing spatial distance and traffic-flow, FAHL places the vertices with lower traffic-flow near the root, resulting in smaller label entries. In this way, we reduce the overall index size and enhance the query efficiency. To maintain the index when flow\weight updating, we propose Improved Structure Update (ISU) and Index Label Update (ILU) algorithms with FAHL in FRN, and prove theoretically that they can help us to reduce redundant calculations. Finally, an efficient Flow Priority Shortest Path Search Algorithm (FPSPS) with pruning query bounds is proposed to further improve the query performance. Our experiments show that our FAHL achieves 33.1% speedup on average for FSPQ when compared with the state-of-the-art methods (details please refer to Section VI). In summary, the main contributions of this paper are the following:

- We propose a novel index FAHL over FRN, which can efficiently obtain the FSPQ result by exploiting the spatial distance and traffic-flow.
- To overcome the challenge of index inefficiency, we propose a degree-flow joint ordering method to obtain the vertex ordering in FRN, which supports the construction of FAHL. Based on it, FAHL can perceive both spatial and temporal information.
- To overcome the challenge of maintenance latency, we propose ISU and ILU algorithms to support the index structure and index label update respectively. Based on the updating algorithms, FAHL is able to answer FSPQ under high-frequency flow\weight changes. Besides, an efficient FPSPS algorithm with pruning query bounds is designed to enable further query processing speedups.
- We conduct extensive experiments on real-world datasets. The result demonstrates that our proposed method achieves the best performance including the state-of-the-art methods.

Section II provides preliminaries and formal definitions of our paper, and in Section III, the details of FAHL are described, including the index structure, and the index construction processes. Then, in Section IV, we present ISU and ILU two algorithms to support the index structure and label updating respectively. In Section V, we will propose an efficient FPSPS algorithm with pruning query bounds to finish the final optimal query. Finally, we demonstrate that FAHL markedly outperforms the state-of-the-art techniques in Section VI.

II. PRELIMINARIES

In this section, we proceed to give the formal definition of the FRN and the problem definition. We also list the frequently used notations in Table II.

Definition 1: ***Flow-Aware Road Networks (FRN)****. We model a FRN as a weighted graph $G_f = (V, E, F_v, W_e)$ with the recorded time slice T, where:*

1) *V represents a set of $n = |V|$ vertices (i.e., road segments) within the FRN.*
2) *E is an edge set, each of which represents the connectivity of the corresponding vertex pair.*
3) *$F_v = (fl^{t-T+1}, \ldots, fl^{t-1}, X^t) \in \mathbb{R}^{T\times n}$ is the traffic-flow (i.e., number of vehicles) for all vertices from the*

TABLE II: Notations

Notation	Description
$G_f = (V, E, F_v, W_e)$	Flow-aware road network G_f
$D(v_i)$	The degree of v_i in FRN
$\rho(v_i, v_j)$	A set of path between v_i and v_j
$PDis_{G_f}(v_i, v_j)$	Path distance
$TF_{G_f}{}^t(v_i, v_j)$	Path traffic-flow
η_u	User constrained parameter
$\hat{P}$	Predicted traffic-flow
$FSP_{G_f}{}^t(v_i, v_j)$	Flow-aware shortest path
$FSD_{G_f}{}^t(v_i, v_j)$	Flow-aware shortest distance
$Q = \langle Q_u, D_u, t_q \rangle$	Shortest distance query at time t_q
$\chi(v_i)$	Tree node of T_{G_f}
ϖ_T	The treewidth of T_{G_f}
h_T	The treeheight of T_{G_f}
$\ell(v_i)$	A label set for v_i

current t to the past T time slice. The graph signals $fl^t = (fl^t_{v_1}, fl^t_{v_2}, \dots, fl^t_{v_n}) \in \mathbb{R}^n$ and $fl^t_{v_i} \in F_v$ denotes the traffic-flow at t and v_i respectively.

4) *W_e represents a set of edges's weight. For any edge $(v_i, v_j) \in E$, we have their weight $w_{v_i,v_j} \in \mathbb{N}^*$ which describes the spatial distance.*

Compared with existing road networks, our FRN has the following key distinctions: 1) we take road segments as vertices instead of the original intersection nodes to obtain more accurate traffic-flow; 2) spatial distance and traffic-flow are independent attributes in G_f rather than internal factors that cause the weights (e.g., travel time) to change. Thus, it avoids the redundant consideration of related features.

Definition 2: ***Adjacency Matrix and Vertex Degree***. *The adjacency matrix of FRN is represented by $A_{G_f} \in \mathbb{R}^{n\times n}$, where n is the number of vertices. $Adj_{ij} \in A_{G_f}$ represents the row-i, column-j element of A_{G_f}. If we have $v_i, v_j \in V, (v_i, v_j) \in E$, Adj_{ij} is set to 1, otherwise, Adj_{ij} is equal to 0. For vertex v_i, we adopt $D(v_i) = \sum_{j=1}^{n} Adj_{ij}$ to denote the vertex degree.*

Definition 3: ***Path Distance and Traffic-Flow***. *For any vertices v_i and v_j in G_f, $P_a(v_i, v_j) = \{v_i, \dots, v_k, v_{k+1}, \dots, v_j\}$ is a sequence of connected vertices. Let $v_i = v_1$, $v_j = v_m$, if $(v_k, v_{k+1}) \in E, 1 \le k < m$, we use $\rho(v_i, v_j) = \{P_a \,|P_a(v_i, v_j)\}$ to denote the set of the all path between v_i and v_j. The sum of the weights for each edge in $\rho(v_i, v_j)$ is the path distance, denoted as $PDis_{G_f}(v_i, v_j)$, $SPDis(v_i, v_j) = \min\{PDis_{G_f}(v_i, v_j)\}$ is the shortest distance. Path traffic-flow $TF_{G_f}{}^t(v_i, v_j)$ is the sum of the vertices' flow which are contained of the vertices in path, $TF_{G_f}{}^t(v_i, v_j) = \sum_{v_k \in \rho(v_i,v_j)} X^t_{v_k}$. In our work, we adopt PDFormer to obtain the predicted flow $\hat{P}$.*

To measure the road capacity and different updated quantities of flow and weight. We propose the following definition.

Definition 4: ***Capacity-based flow and update ratio***. *We define a capacity-based flow to replace the predicted traffic-flow, denoted by $\hat{C}_f = W_c \cdot \hat{P} + (1 - W_c) \cdot \hat{R}$, where $\hat{P}$ and $\hat{R}$ are traffic-flow and road capacity respectively, $\hat{R} = \hat{P} \Big/ N_l$, N_l is the predicted lane numbers, and W_c represents the ratio of $\hat{P}$ and $\hat{R}$. We adopt $\lambda = T(\hat{P}^*) \Big/ T(w^*)$ to denote the update ratio, where $T(\hat{P}^*)$ and $T(w^*)$ represent the quantities of traffic-flow changes and edge weight changes, respectively. We also adopt PDFormer to estimate the road capacity.*

Definition 5: ***Flow-Aware Shortest Path (FSP)***. *To combine the distance and traffic-flow simultaneously, we propose a simple yet effective equation to compute the shortest flow-aware distance $FSD_{G_f}{}^t(v_i, v_j)$ of the path as follows,*

$$FSD_{G_f}{}^t(v_i, v_j) = \min\{\alpha \cdot PDis_{G_f}{}' + (1-\alpha) \cdot TF_{G_f}{}^{t\prime}\} \quad (1)$$

$$PDis_{G_f}{}' = \frac{(PDis_{G_f}(v_i, v_j) - PDis_{G_f}(v_i, v_j)_{\min})}{(PDis_{G_f}(v_i, v_j)_{\max} - PDis_{G_f}(v_i, v_j)_{\min})} \quad (2)$$

$$TF_{G_f}{}^{t\prime} = \frac{(TF_{G_f}{}^t(v_i, v_j) - TF_{G_f}{}^t(v_i, v_j)_{\min})}{(TF_{G_f}{}^t(v_i, v_j)_{\max} - TF_{G_f}{}^t(v_i, v_j)_{\min})} \quad (3)$$

where $PDis_{G_f}{}'$ and $TF_{G_f}{}^{t\prime}$ are the normalized distance and traffic-flow respectively, $\alpha \in (0, 1)$. We adopt the min-max normalization [21] in this definition. The parameter α is adopted to balance the effects of the path distance and traffic-flow. Final, we have $FSP_{G_f}{}^t(v_i, v_j)$ is the FSP which has the shortest flow-aware distance.

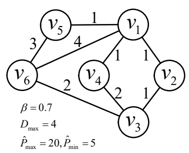
We are able to get $PDis_{G_f}(v_i, v_j)_{\min} = SPDis(v_i, v_j)$ on Def. 3. However, the corresponding distance of $PDis_{G_f}(v_i, v_j)_{\max}$ may be very large since $PDis_{G_f}(v_i, v_j)$ can even contain **all vertices** of FRN. If we take this distance as $PDis_{G_f}(v_i, v_j)_{\max}$, it will cause most paths' $PDis_{G_f}{}'$ to remain a low value. In addition, paths with overly long distances contradict the goal of the flow-aware shortest path. Thus, in FSPQ, the value of $PDis_{G_f}(v_i, v_j)_{\max}$ should not exceed $SPDis(v_i, v_j)$ too much. To solve this problem, we propose the maximum constrained path distance by giving a user constrained parameter η_u. η_u will not cause us to miss the potential FSP. This is because it only affects the normalization function, when traffic-flow is fixed, candidate paths with longer distances will certainly not be the FSP.

The maximum constrained path distance can be computed as $MCPDis(v_i, v_j) = \eta_u \cdot SPDis(v_i, v_j)$. We take $MCPDis(v_i, v_j)$ as the maximum distance while we obtain the FSP. The paths with distance $PDis_{G_f}(v_i, v_j) > MCPDis(v_i, v_j)$ will not be considered in FSPQ.

Flow-Aware Shortest Path Querying (FSPQ) Problem: The FSPQ considers both spatial distance and traffic-flow simultaneously. Given G_f, query $Q = \langle Q_u, D_u, t_q \rangle$, where Q_u and D_u are query location and destination respectively, t_q is the query time step. Our goal is to propose an efficient query framework to find the FSP in FRN, i.e., $FSP_{G_f}{}^t(Q_u, D_u)$.

III. Flow-Aware Hierarchical Labeling Index

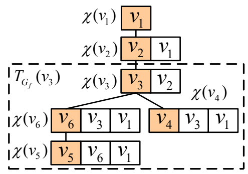
We proceed to present the Flow-Aware Hierarchical Labeling Index (FAHL) that extends H2H [11], [22] to support FSPQ efficiently. Like H2H imposes a vertex ordering and maps the entire road network to a tree structure, we also map the G_f into a tree structure T_{G_f} and enable it to answer FSPQ.



(a) Example of G_f

	$\widehat{D}(v_i)$	$\hat{P}(v_i)$	$\widehat{P}(v_i)$	$\varphi(v_i)$
v_1	1	20	1	1
v_2	0.5	5	0	0.15
v_3	0.75	7	0.2	0.32
v_4	0.5	9	0.33	0.34
v_5	0.5	12	0.47	0.48
v_6	0.75	9	0.27	0.42

(b) Calculation of $\varphi(v_i)$



(c) Flow – Aware Tree Decomposition

$\ell(v_i)$	$Position(v_i)$	$Distance(v_i)$	$\ell(v_i)$	$Position(v_i)$	$Distance(v_i)$
v_1	1	0	v_4	1,2,4	2,2,1,0
v_2	1,2	1,0	v_5	1,2,5	3,4,3,2,0
v_3	1,2	1,0	v_6	1,2,4	2,3,4,0

(d) Flow – Aware Hierarchical Labeling Index

Fig. 2: The Structure of FAHL

When we adopt a tree decomposition to map the entire FRN G_f into a tree structure T_{G_f}, we observe that the label entries of the vertices near the root of T_{G_f} are smaller than others. Since traffic-flow is carried by the vertices of G_f and a vertex with lower traffic-flow may have a greater impact on the result of FSPQ, if we place the vertices with lower traffic-flow near the root, these vertices will have smaller label entries. In this way, we will spend less time when we obtain the FSPQ involving these vertices since vertices near the root are more likely to become the LCA node in T_{G_f} and we obtain FSPQ on it (will be discussed in Section III-C). Meanwhile, we are able to reduce the treewidth ϖ_T of T_{G_f} and improve the query efficiency in this situation.

Although both H2H and our FAHL adopt the tree decomposition, the problems that these two indices aim to solve are different, which means they have different structures. To enable the index can perceive the traffic-flow, we propose a joint ordering method. Moreover, we place the vertices with lower traffic-flow near the root of the tree. Besides, FAHL records the flow in each node. Referring to the H2H, its T_{G_f} is only built on the ordering of all vertices' topological degree, and it only records the spatial information. To achieve FAHL, the urgent task is to introduce this flow information into the vertex ordering. In this way, we can make full use of the tree decomposition and flow information to improve the efficiency of FSPQ. In this section, we will first introduce the FAHL index structure and then provide the index's construction algorithm.

A. The FAHL Index Structure

Based on the above intuition, we propose a degree-flow joint ordering to sort the vertex ordering and use it to compute the flow-aware T_{G_f} with the corresponding labeling index. Before that, we will first introduce the tree decomposition in FRN.

Definition 6: ***Tree Decomposition in FRN****. We assume T_{G_f} is the tree decomposition of a flow-aware road network $G_f = (V, E, F_v, W_e)$. $v_{T_{G_f}}$ denotes the set of nodes in T_{G_f}*[3]*. $\chi(v_i) \in T_{G_f}, v_i \in V$ represents each node of T_{G_f} which contains a subset of V, and the following properties should be satisfied:*

1) *$\cup\chi(v_i) = V$;*
2) *For any edge $(v_i, v_j) \in E$, there must exists a $\chi(v')$ which satisfies $v_i, v_j \in \chi(v')$;*
3) *For any vertex $v_i \in V$, a connected subtree of tree decomposition T_{G_f} is denoted by the set $\{\chi(v_i) \,|v_i \in \chi(v')\}$.*

T_{G_f} has two basic attributes, the tree width ϖ_T and tree height h_T. ϖ_T is one less than the maximum size of all nodes, it can be computed by $\varpi_T = \max\{|\chi(v_i)|\} - 1$. $|\chi(v_i)|$ represents the number of vertices in $\chi(v_i)$. The tree height h_T is the maximum distance from the leaf nodes to the root. Existing works have been proven that the smaller ϖ_T and h_T, the higher efficiency of corresponding T_{G_f}. For any node $\chi(v_i) \in T_{G_f}$, if there exists a vertex $v_j \in \chi(v_i)\backslash\{v_i\}$, we call $\chi(v_j)$ is an ancestor of $\chi(v_i) \in T_{G_f}$. Let $(\chi(m_1), \chi(m_2), \ldots, \chi(m_k))$ as the path from the root of T_{G_f} to $\chi(v_i)$, where $\chi(m_1)$ is the root and $\chi(m_k)$ is the $\chi(v_i)$, we can adopt $\chi(v_i)_{anc} = (m_1, m_2, \ldots, m_k)$ to represent the ancestor array of $\chi(v_i)_{anc}$, $\chi(v_i)^k_{anc}$ is the k-th element in it.

The traditional vertex ordering method of tree decomposition is to compute the vertex's degree. If we adopt such a method to build T_{G_f}, we will compute the degree of v_i and generate $\chi(v_i)$ in ascending order of their degree to ensure that the nodes with high degrees are near to the root of T_{G_f}. By selecting the nodes with high degree first, the fewest edges are needed to add in each step of the T_{G_f}, yielding a tree with small width ϖ_T and height h_T. Vertices' degree can represent a spatial correlation, we plan to introduce traffic information while inheriting this correlation. In this situation, FAHL can simultaneously leverage both spatial information and traffic-flow. To place the vertices with low traffic-flow near the root and speed up the query, we propose a degree-flow joint ordering method in Def. 7:

Definition 7: ***Degree-Flow Joint Ordering Method****. Let $\varphi(v_i)$ represent the degree-flow joint importance of the vertex, our degree-flow joint ordering method adopts $\varphi(v_i)$ to sort the vertex in ascending order, where $\varphi(v_i)$ is computed as follows,*

$$\varphi(v_i) = \beta \cdot \widehat{P}(v_i) + (1 - \beta) \cdot \widehat{D}(v_i) \tag{4}$$

where $\widehat{D}(v_i) = {}^{D(v_i)}/_{D_{\max}}$, $D(v_i)$ is the degree when we computing the vertex v_i in construction processing of T_{G_f}. $D_{\max}$ is the maximum degree of all vertex. The normalized predicted traffic-flow of vertex v_i is $\widehat{P}(v_i) = (\hat{P}_{total}(v_i) - \hat{P}_{\min}) \big/ (\hat{P}_{\max} - \hat{P}_{\min})$, $\hat{P}_{\max}$ and $\hat{P}_{\min}$ are the maximum and minimum traffic-flow of all vertices respectively.

Example 1: Given an example G_f as shown in Fig. 2(a), we can compute all relevant variables in Fig. 2(b) based on the degree-flow joint ordering method. First of all, we obtain the D_{max} by graph topology and $\hat{P}_{max} = 20$, $\hat{P}_{min} = 5$ according to the prediction traffic-flow $\hat{P}(v_i)$. Due to the parameter β=0.7, we have the degree-flow joint importance $\varphi(v_i)$ for each vertex in G_f and can sort all vertices in ascending order as $\{v_2, v_3, v_4, v_6, v_5, v_1\}$. v_1 is selected as the root node of T_{G_f} since it has the degree-flow joint importance. Then, we will generate $\chi(v_i)$ in ascending order of the joint importance.

[3]For simplicity, we adopt each vertex $v_i \in V$ of G_f as a **vertex** and to each $\chi(v_i) \in T_{G_f}$ in tree decomposition as a **node** in our paper.

Example 2: Combining the degree-flow joint ordering and tree decomposition, our flow-aware tree decomposition is shown in Fig. 2(c). There are six nodes in T_{G_f} corresponding to six vertices in G_f. For node $\chi(v_3)$, it contains two vertices $\{v_3, v_2\}$. For edge $(v_4, v_3) \in E$, node $\chi(v_4) = \{v_4, v_3, v_1\}$ guarantees the second property of Def. 6. $T_{G_f}(v_3)$ is a subtree of T_{G_f}. It contains nodes $\chi(v_3)$, $\chi(v_4)$, $\chi(v_4)$ and $\chi(v_5)$, $\chi(v_3)$ is the root of this subtree.

Based on Def. 6 and Def. 7, we can finally generate the FAHL, the definition is as follows,

Definition 8: ***Flow-Aware Hierarchical Labeling Index (FAHL).*** *The FAHL is defined on the flow-aware tree decomposition. For each node $\chi(v_i) \in T_{G_f}$, suppose we have $\chi(v_i) = \{n_1, n_2, \ldots, n_p\}$ and $\chi(v_i)_{anc} = (m_1, m_2, \ldots, m_k)$, $\chi(v_i)$ is the subset of the $\chi(v_i)_{anc}$. The FAHL is denoted by $\ell(v_i) = \{Position(v_i), Distance(v_i)\,|v_i \in \chi(v_i)\}$, where $Position(v_i)$ and $Distance(v_i)$ are arrays that store the relative position and road distance between a vertex to its ancestors respectively.*

- *$Position(v_i)$: we let $Position(v_i)$ to represent the position array of $\chi(v_i)$, where $Position(v_i) = (pv_1, pv_2, \ldots, pv_p)$ and $pv_i, (1 \leq i < p+1)$ is the position of n_i in $\chi(v_i)_{anc}$, i.e., $\chi(v_i)^{pv_i}_{anc} = n_i$. For simplicity, we arrange values in $Position(v_i)$ as increasing order, and adopt $\chi(v_i)^j_{pos}$ to denote the j-th element in $Position(v_i)$ while $1 \leq j \leq p$.*
- *$Distance(v_i)$: we let $Distance(v_i)$ to denote the distance array of $\chi(v_i)$, and define it as $Distance(v_i) = ((SPDis(v_i, m_1), SPDis(v_i, m_2), \ldots, SPDis(v_i, m_k))$. In a nutshell, the distance array contains all shortest distance from vertex v_i to its ancestors in $\chi(v_i)_{anc}$. Same as the position array, we use $\chi(v_i)^j_{dis}$ to denote the j-th element in $Distance(v_i)$.*

Example 3: Fig. 2(d) shows an example of proposed FAHL for the given G_f which adopts the flow-aware tree decomposition. For the node $\chi(v_5) = \{v_5, v_6, v_1\}$, we have its ancestor array $\chi(v_5)_{anc} = \{v_1, v_2, v_3, v_6, v_5\}$, and we can obtain the $Position(v_5)$ as $Position(v_5) = (5, 2, 1)$. Because v_5, v_6 and v_1 are the 5-th, 2-nd and 1-st elements in $\chi(v_5)_{anc}$ respectively. According to the sorted ascending ordering sequence, we have $Position(v_5) = (1, 2, 5)$. For $Distance(v_5)$, we can directly compute the distance between v_5 and its ancestors. In this way, we have $Distance(v_5) = (SPDis(v_5, v_1), SPDis(v_5, v_2), \ldots, SPDis(v_5, v_5)) = (3, 4, 3, 2, 0)$.

B. The FAHL Index Construction

In this section, we will give the index construction processes with the algorithm.

The FAHL Index Construction Algorithm. With the help of Def. 6-8, we are now ready to propose the index construction algorithm. The detailed pseudocode is shown in Alg. 1. When we get the flow-aware road network G_f, we will first initialize the T_{G_f} and obtain the predicted traffic-flow of each vertex (Lines 1 to 2). Suppose t_{start} and t_{end} are the first and last prediction timesteps respectively, t is the time interval, and $t_q \in [t_{start}, t_{end}]$ is the query time. We will predict the traffic-flow every t minutes during $[t_{start}, t_{end}]$ and build the index on t_{start}. Then, the algorithm will compute the joint ordering $\varphi(v)$ and iteratively eliminate the vertex v with the smallest $\varphi(v)$ (Lines 4 to 5). We initialize $\chi(v)$ for each vertex $v \in G_f$ (Lines 6 to 7), since each vertex will be a node in T_{G_f}. In line 8, we maintain the order $\varphi(v)$ for all vertices. Then in Lines 9 to 12, we assign the parent node for each node in T_{G_f} to construct the tree structure. Then we generate each node $\chi(v)$ (Lines 13 to 16) and compute the corresponding $\chi(v)_{pos}$ and $\chi(v)_{dis}$ in it (Lines 17 to 19). Finally, the position array and distance array for all vertices in G_f are obtained as the FAHL index. The time complexity of Alg. 1 is $O(n \cdot log(n) + \varpi_T{}^2 \cdot n)$, and the space complexity of this algorithm is $O(n \cdot \varpi_T)$, where ϖ_T is the treewidth of tree decomposition T_{G_f}.

Algorithm 1: *Index Construction Algorithm*

Input: A Flow-aware Road Network T_{G_f}
Output: The FAHL index
1 $T_{G_f} \leftarrow \emptyset$;
2 $\hat{P}_{total} \leftarrow$ obtain the predicted traffic-flow;
3 **for** $i = 1$ *to* $n = |V|$ **do**
4   Compute the joint ordering $\varphi(v)$ by Def. 7;
5   $v \leftarrow$ the node in T_{G_f} with biggest $\varphi(v)$;
6   $\chi(v) \leftarrow$ initialize $\chi(v)$ for all vertices;
7   Generate $\chi(v)$ in T_{G_f};
8   Maintain the $\varphi(v)$ for each vertex;
9 **for** *all* $v \in G_f$ **do**
10   **if** $|\chi(v)| > 1$ **then**
11     $m \leftarrow$ the vertex in $\chi(v) \backslash \{v\}$ smallest $\varphi(v)$;
12   Set the $\chi(m)$ as the parent node of $\chi(v)$ in T_{G_f};
13 **for** *all* $v \in G_f$ **do**
14   Sort vertices in $\chi(v)$ in ascending order of $\varphi(v)$;
15   **if** $1 \leq i \leq |\chi(v)|$ *and* $v_i \in \chi(v)_{anc}$ **then**
16   Add v_i into the $\chi(v)$ and it is the i-th vertex of $\chi(v)$;
17 **for** *all* $\chi(v) \in T_{G_f}$ **do**
18   Generate the $\chi(v)_{anc}$;
19   $\chi(v) \leftarrow$ compute the $\chi(v)_{pos}$ and $\chi(v)_{dis}$;
20 Return the FAHL index;

Algorithm 2: *Shortest Spatial Distance Querying*

Input: T_{G_f}, the FAHL and query pair (Q_u, D_u)
Output: the shortest spatial distance $SPDis(Q_u, D_u)$
$L \leftarrow Lca(Q_u, D_u)$;
$SPDis(Q_u, D_u) \leftarrow +\infty$;
$k \leftarrow 0$, obtain the position array L_{pos} of node L in T_{G_f};
for $k \in L_{pos}$ **do**
  Retrieve distance arrays $\chi(Q_u)^k_{dis}$ and $\chi(D_u)^k_{dis}$;
  $SPDis(Q_u, D_u) \leftarrow \min\{\chi(Q_u)^k_{dis} + \chi(D_u)^k_{dis}\}$;
Return $SPDis(Q_u, D_u)$;

C. Shortest Spatial Distance Querying

The shortest spatial distance between any vertex pair (v_i, v_j) where $v_i \in \chi(v_i)$ and $v_j \in \chi(v_j)$ dependents on the Lowest Common Ancestor (LCA) of $\chi(v_i)$ and $\chi(v_j)$, denotes as $Lca(v_i, v_j)$ in flow-aware tree decomposition T_{G_f}. $SPDis(v_i, v_j)$ goes through at least one vertex in $Lca(v_i, v_j)$. By the help of the $Lca(v_i, v_j)$, the $SPDis(v_i, v_j)$ can be computed quickly as follows:

$$SPDis(Q_u, D_u) = \min_{k \in L_{pos}} \{\chi(Q_u)^k_{dis} + \chi(D_u)^k_{dis}\} \quad (5)$$

where Q_u and D_u are query vertices, $L = Lca(Q_u, D_u)$ is the lowest common ancestor node in T_{G_f}, L_{pos} represents L's position array. $\chi(Q_u)^k_{dis}$ ($\chi(D_u)^k_{dis}$) is the k-th element in the corresponding $Distance(Q_u)$ ($Distance(D_u)$), the detailed algorithm is shown in Alg. 2 with $O(\varpi_T)$ time complexity. Based on Eq. 5, we can compute the shortest distance by FAHL directly rather than traverse the vertices. Meanwhile, benefiting from the degree-flow joint ordering method, the vertices near the root have lower traffic-flow than other vertices, thus we can more easily obtain the FSPQ when we compute the path in corresponding LCA. These advantages improve the entire query efficiency.

Example 4: We take the T_{G_f} of Fig. 2 as a running example of the shortest spatial distance querying. Given shortest distance query $SPDis(v_6, v_4)$ of v_6 and v_4, we need to first find the $Lca(v_6, v_4) = \chi(v_3)$. After that, we obtain the position array of $\chi(v_3)$ by $Position(v_i) = \{1, 2\}$. At last, we have $SPDis(v_6, v_4) = \min\{2 + 2, 2 + 3\} = 4$.

IV. Index Maintenance in Flow-aware Road Networks

FRN is changing frequently. On the one hand, each vertex's traffic-flow will change as time goes on, which results in the update of degree-flow joint ordering. Thus, we need first to update the T_{G_f}, and then update the relevant label entries for the structure changes since the vertices' ancestor array is changed. On the other hand, the edge weights may change frequently by the influence of different road conditions, while we need to update the label entries in FAHL to ensure that the index contains the correct distance. Thus, in this section, we aim to devise efficient index maintenance algorithms to handle the index updates in the following aspects: (1) the update of FAHL index structure, caused by the vertex's flow change; (2) the update of FAHL recorded value, caused by the edges' weights change.

A. FAHL Index Structure Update

Given a former FRN G_f. If the traffic-flow of v_i is updated to $\hat{P}(v_i)^*$, and we adopt $G_f{}^*$ to represent the updated road network after the flow changes. Based on the index construction algorithm, we know that the index relies on the degree-flow joint ordering. If the change of flow influences the vertices' ordering, we need to reconstruct the relevant subtree of changed vertices. In other words, for those vertices that have not been affected, we don't need to update them in any way. We have:

Lemma 1: Given G_f, $v_i \in G_f$. $\hat{P}(v_i)^*$ is an updated traffic-flow of v_i from $\hat{P}(v_i)$, if v_i satisfies the following conditions, we will not update the existing index structure: (1) v_i is the root node of tree decomposition T_{G_f}, and $\hat{P}(v_i)^* > \hat{P}(v_i)$; (2) v_i is not the root node, and we have $\varphi(v_n) \leq \varphi(v_i)^* \leq \varphi(v_m)$. $\varphi(v_i)^*$ is a recomputed degree-flow joint ordering by $\hat{P}(v_i)^*$, v_n and v_m are the corresponding vertices adjacent to v_i in the original non-updated ordering sequence; (3) v_i is located in the first place of the original non-updated ordering sequence, and $\hat{P}(v_i)^* \leq \hat{P}(v_i)$.

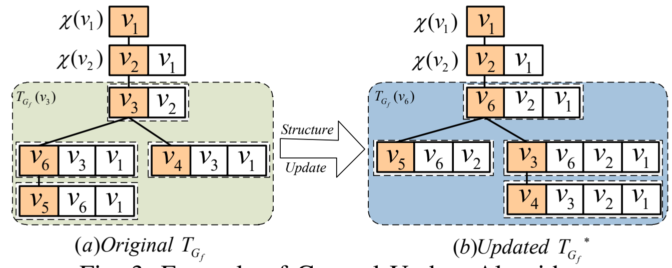


Fig. 3: Example of General Update Algorithm

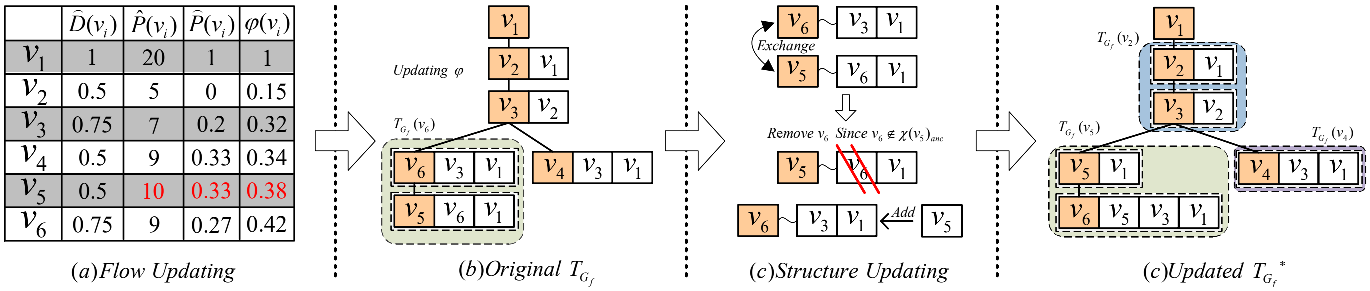
Proof: Case (1) is direct, if v_i is the root node, and traffic-flow increases, v_i also has the maximum φ and it will be selected as the root node too. Thus, we do not need to update the structure. For case (2), we prove it by contradiction. If we need to update the index structure in this situation, v_i's ordering must change, and the ascending order sequence is updated. Put it briefly, when we update the $\varphi(v_i)^*$ by $\hat{P}(v_i)^*$, v_i is not located in the original place of updated order sequence and we have $\varphi(v_i)^* > \varphi(v_m)$ or $\varphi(v_i)^* < \varphi(v_n)$ which conflict with the $\varphi(v_n) \leq \varphi(v_i)^* \leq \varphi(v_m)$. Therefore, case (2) is proved. Case (3) is similar to case (1), v_i is located in the first place and its traffic-flow decreases, so the ordering of all vertices is unchanged and we do not update the tree structure. Hence, the lemma is proved. ■

General Structure Update Algorithm (GSU). Based on Lemma 1, it is obvious that whether the index structure needs to be updated depends on whether the degree-flow joint ordering sequence changes due to the traffic changes. In this way, we can divide the entire index structure update into two steps. For v_i, in the first step, we recompute the updated joint ordering $\varphi(v_i)$ and sort all vertices with new ordering to get an updated sequence. Then in the second step, we get the k-th order in the sequence of v_i, keep the vertices' corresponding node in T_{G_f} which the order before k-th, and reconstruct the tree structure from v_i to other vertices. If $\varphi(v_i)$ does not influence the original root node's ordering, we keep the root node in the updated index, otherwise, we reconstruct the entire index. We adopt Alg. 1 to finish the reconstruction processes.

Example 5: Take the G_f and T_{G_f} in Fig. 2 as an example. Fig. 3 shows the structure update processes. We have the original order sequence as $\{v_2, v_3, v_4, v_6, v_5, v_1\}$. Assume that v_6's traffic-flow is changing. Based on the general structure update algorithm, we have to recompute the $\varphi(v_i)$ and obtain the updated sequence $\{v_2, v_6, v_3, v_4, v_5, v_1\}$. Since v_6's ordering changed from 4-th to 2-th, we keep $\chi(v_1)$ and $\chi(v_2)$ into the index and reconstruct the tree structure from v_6 with order 2-th. We have subtree $T_{G_f}(v_6)$ as shown in Fig. 3(b) instead of the $T_{G_f}(v_3)$ as shown in Fig. 3(a).

Although the GSU algorithm can help us update the index structure in any situation, however, it still involves lots of **redundant calculation** when reconstructing the changed subtree. Thus, we aim to further improve the maintenance performance by avoiding unnecessary calculations now.

Based on the observation of the GSU algorithm, we find that the smallest reconstruction unit is the entire subtree of the corresponding vertex whose ordering is changed. This is too coarse since if the changed vertex is located at the beginning



	$\widehat{D}(v_i)$	$\widehat{P}(v_i)$	$\widehat{P}(v_i)$	$\varphi(v_i)$
v_1	1	20	1	1
v_2	0.5	5	0	0.15
v_3	0.75	7	0.2	0.32
v_4	0.5	9	0.33	0.34
v_5	0.5	10	0.33	0.38
v_6	0.75	9	0.27	0.42

Fig. 4: Example of Improved Structure Update Algorithm

Algorithm 3: *Improved Structure Update Algorithm*

```
   Input: The original FAHL index, updated flow P̂(v_t)*
   Output: The updated FAHL index
1  Sub_φ ← ∅, v_m ← ∅;
2  Compute updated joint ordering φ(v_t)* by P̂(v_t)*;
3  Update the ordering sequence in ascending order by φ;
4  for all v_i ∈ T_{G_f} do
5      if φ(v_i) < φ(v_t) and φ(v_t)* < φ(v_i) then
6          Sub_φ ← Add the vertex v_i into the subsequence;
7  v_m ← find the original root of subtree in Sub_φ;
8  Exchange χ(v_m) and χ(v_t);
9  Remove unfit vertices in χ(v_m), χ(v_t) by χ(v_m)_anc, χ(v_t)_anc;
10 for all χ(v_i) ∈ T_{G_f} do
11     for all v_i ∈ T_{G_f} do
12         χ(v_i) ← Add v_i into the χ(v_i) by Def. 6;
13 Return the updated FAHL index;
```

of the original sequence, we almost need to reconstruct the entire tree. In addition, parts of the existing index can be reused. From the index update in Example 5, $\chi(v_4)$ also contains v_2 and v_1 in the updated $T_{G_f}{}^*$. The index structure is highly correlated with the vertex ordering, and each vertex v_i is located in its parents' subtree $T_{G_f}(v_i)_{anc}$. Based on it, when $\varphi(v_i)$ changes, we only need to update the v_i with influenced ranked vertices, since the order of subsequences remains relatively constant. We have:

Lemma 2: Given the tree decomposition T_{G_f}, $T_{G_f}(v_m) \in T_{G_f}$ is a subtree of T_{G_f} and v_m is the root node in this subtree. For vertex $v_t \in T_{G_f}(v_m)$, assume that the traffic-flow of v_t changes, and $\varphi(v_t)^* < \varphi(v_m)$. We need to update the index structure in this situation. We first exchange the position between $\chi(v_t)$ with $\chi(v_m)$. Then, we remove the vertices recorded in $\chi(v_t)$ ($\chi(v_m)$) which are not in $\chi(v_t)_{anc}$ ($\chi(v_m)_{anc}$). Finally, We need to add a series of missing edges according to the second property of Def. 6. In this way, we do not need to reconstruct the entire $T_{G_f}(v_m)$.

Proof: We prove it by contradiction. Assume that we need to reconstruct the entire $T_{G_f}(v_m)$, there must be other parts of the structure that need to be updated. From this, we can infer that there must exist a vertex v_q, and its joint ordering changed. Indicating that $\varphi(v_q)^* > \varphi(v_{q+1})$ or $\varphi(v_q)^* < \varphi(v_{q-1})$. Thus, v_q's traffic-flow also changes. However, only v_t' flow has changed. In this condition, v_q and v_t are the same vertex or v_q does not exist. The lemma is proved. ∎

Improved Structure Update Algorithm (ISU). By Lemma 2, we can finally propose the Improved Structure Update Algorithm (ISU) in Alg. 3. First of all, we initialize the ordering subsequence Sub_φ and the influenced subtree node v_m (Line 1). After this, we compute the updated joint ordering $\varphi(v_t)^*$ and obtain the updated ordering sequence (Lines 2 to 3). Then we create the ordering subsequence which is affected by v_t's flow change (Lines 4 to 6). Next, we update the index structure by exchanging $\chi(v_t)$ and $\chi(v_m)$, remove the unfit vertices by verifying their ancestor sets (Lines 8 to 9). Finally, we traverse all nodes and all vertices in T_{G_f} to preserve the missing edges by Def. 6 and obtain the complete updated FAHL index (Lines 10 to 13). The time complexity of Alg. 3 is $O(n \cdot (\varpi_T + log(n)))$, ϖ_T is the treewidth. Compared with the GSU algorithm, ISU avoids lots of redundant calculations, making frequent updates possible in FAHL.

Example 6: We also take the G_f and T_{G_f} in Fig. 2 as an example. Fig. 4 shows the update processes. When the traffic-flow of v_5 changes to 10, we recompute the $\widehat{P}(v_5) = 0.33$ and compute $\varphi(v_t)^* = 0.38$. Then, we obtain the updated ordering sequence $\{v_2, v_3, v_4, v_5, v_6, v_1\}$. v_1 is also the root node of T_{G_f}. Since the order of v_5 and v_6 changed, we can obtain the influenced subsequence as $\{v_5, v_6\}$. In the index updating processes, we first exchange $\chi(v_5)$ and $\chi(v_6)$. Due to $v_6 \notin \chi(v_5)_{anc}$, we need to remove v_6 in the updated $\chi(v_5)$ node. After this, we check each node by Def. 6 to preserve the missing edges, and we add v_5 into v_6 for (v_5, v_6) to get the final updated FAHL index.

B. FAHL Index Label Update

The road network in real applications is never static, in addition to the flow changes for vertices, the edge weight may change frequently due to the random traffic status. In this section, we mainly focus on the label updated by edge weight change and assume that the flow remains constant when the edge weight changes. Our goal is to propose an efficient label update algorithm to improve the query performance of FAHL.

Given a FRN G_f, edge $e = (v_i, v_j)$, we adopt $w_{v_i,v_j}{}^*$ to denote the updated weight of e. Based on the Def. 6, Def. 7, and Def. 8, it is obvious that the entire FAHL index structure remains unchanged during the edge weight update, which means we only need to maintain the corresponding label value in each tree nodes of the index. We have:

Lemma 3: Given FRN G_f and the corresponding FAHL index, suppose that a weight w_{v_i,v_j} of (v_i, v_j) changes, we need to update the label value of $\chi(v_t)$ in the index when such nodes meet with the following conditions: (1) for any $\chi(v_t)$, $v_i, v_j \in \chi(v_t)$; (2) there is another node $\chi(v_p)$ expect $\chi(v_t)$,

Algorithm 4: *Index Label Update Algorithm*

Input: The original FAHL index, updated edge weight $w_{v_i,v_j}{}^*$
Output: The updated FAHL index

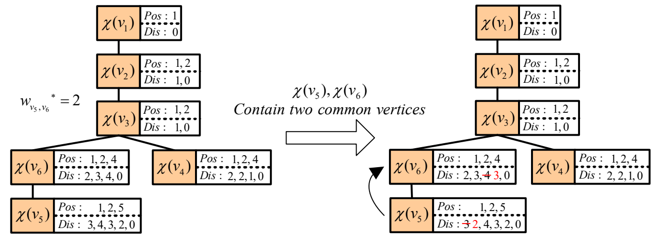
1 $w_{v_i,v_j} \leftarrow w_{v_i,v_j}{}^*$;
2 **if** $\varphi(v_i) > \varphi(v_j)$ **then**
3 Exchange the position of (v_i, v_j) ;
4 Initialize queue with $T_\chi \leftarrow \emptyset$;
5 $T_\chi \leftarrow T_\chi.push(\chi(v_i))$;
6 **while** $T_\chi \notin \emptyset$ **do**
7 $\chi(v_t) \leftarrow T_\chi.pop()$;
8 **for** $Label(v_t) \in \chi(v_t)$ **do**
9 $Label(v_t)^* \leftarrow$ update the label by w_{v_i,v_j};
10 **if** $Label(v_t)^* \neq Label(v_t)$ **then**
11 $Label(v_t) \leftarrow Label(v_t)^*$;
12 **for** $\chi(v_p) \in T_{G_f}$, $\chi(v_p)$ *and* $\chi(v_t)$ *contain more than two common vertices* **do**
13 **if** $\chi(v_p) \notin T_\chi$ **then**
14 $T_\chi \leftarrow T_\chi.push(\chi(v_p))$;
15 Return the updated FAHL index;

if $\chi(v_p)$ and $\chi(v_t)$ contain two or more common vertices, the corresponding label in $\chi(v_p)$ is also need to be updated.

Proof: For case (1), if $v_i, v_j \in \chi(v_t)$, $SPDis(v_i, v_j)$ must be recorded in the label for distance arrays of $\chi(v_t)$. Thus, when the weight of (v_i, v_j) changes, $SPDis(v_i, v_j)$ also changes, we need to update the corresponding weight value of the original label. For case (2), assume that $v_i, v_j \notin \chi(v_t)$, the label value in $\chi(v_t)$ changed, however, the label in all $\chi(v_p)$ expect $\chi(v_t)$ which contain more than two common vertices with $\chi(v_t)$ do not change when w_{v_i,v_j} changes. Based on Def. 6, there must exist a node χ_{v_i,v_j} which contains the (v_i, v_j). For any spatial shortest path $SPDis(v_m, v_n), v_m, v_n \in \chi(v_t)$ which contains edge (v_i, v_j). In this way, there must exist a subpath $SPDis(v_a, v_b) \subseteq SPDis(v_m, v_n)$ satisfies that $(v_i, v_j) \in SPDis(v_a, v_b)$ and the intersection set between the vertex in $SPDis(v_a, v_b)$ with $\chi(v_t)$ is $\{v_i, v_j\}$. Since $SPDis(v_a, v_b)$ is the subpath of $SPDis(v_m, v_n)$, when w_{v_i,v_j} changes, $SPDis(v_m, v_n)$ will also change, in this situation, $\chi(v_t)$ needs to be updated. Based on the assumption, the label in $\chi(v_p)$ and $\chi(v_t)$ must be unchanged. It contradicts the assumption. Therefore, we have proved this lemma. ∎

According to Lemma 3, when the weights change, we first update the corresponding label in the FAHL index, if the label remains unchanged, it means that the index is not affected by such edge. Otherwise, we need to update the label of tree nodes following the Lemma 3 until there is no change.

Index Label Update Algorithm (ILU). Following the Lemma 3, the index label update algorithm is shown in the Alg. 4. At the beginning, we update the edge weight w_{v_i,v_j} by $w_{v_i,v_j}{}^*$ (Line 1). To ensure that $v_j \in \chi(v_i)$, we need to exchange the position of (v_i, v_j) if $\varphi(v_i) > \varphi(v_j)$ (Lines 2 to 3). Next, we initialize queue T_χ to record the changed nodes χ and push $\chi(v_i)$ into it as the first node (Lines 4 to 5). We then iteratively get the node $\chi(v_t)$ of T_χ, and update the label in it (Lines 6 to 9). If the updated labels are different from the existing ones, we then update the label in other relevant by adding them into the T_χ (Lines 10 to 14). The update processing finishes when there are no changes occur the index, T_χ is empty and we return the final updated FAHL index (Line 15). The time complexity of Alg. 4 is $O(n \cdot (n + \varpi_T))$, ϖ_T is the treewidth.



(a)Original Label (b)Updated Label

Fig. 5: Example of Index Label Update Algorithm

Example 7: We take the G_f and FAHL in Fig. 2 as the example. The update processes can be found in Fig. 5. When edge (v_5, v_6)'s weight changes to $w_{v_5,v_6}{}^* = 2$, we first locate it in $\chi(v_5)$ and update the label of (v_5, v_6) directly in it. After this, we start to check other nodes' labels in T_{G_f} which contains more than two common vertices with $\chi(v_5)$. Since $\chi(v_5) \cap \chi(v_6) = \{v_6, v_1\}$, $\chi(v_6)$ satisfies the above condition. Thus, we update the label of the 3-rd records in $\chi(v_6)$ from 4 to 3, because $SPDis(v_6, v_1) = w_{v_6,v_5} + w_{v_6,v_1}$. Due to there is no other weight changes, the update processes are finished.

V. Flow Priority Shortest Path Search Algorithm

We are proceeding to present how to further improve the query efficiency by utilizing the predicted traffic-flow. Therefore, we propose a Flow Priority Shortest Path Search Algorithm (FPSPS). FPSPS mainly processes two kinds of data: 1) the precise predicted traffic-flow at different future times from the model; and 2) the distance between the query vertices from the FAHL. Next, we will first describe our pruning method with query bounds, and then we give the details of our FPSPS algorithm.

A. Pruning Method with Query Bounds

According to the previous sections, we have gained the path distance and traffic-flow of any vertex pair (v_i, v_j) which can support the FSPQ through the Eq. 1. To further improve the query speeds, we should find the lower and upper bounds of the $PDis_{G_f}{}'$ and $TF_{G_f}{}^{t'}$ respectively. Actually, with the help of the maximum constrained path distance, we have already restricted the distance of $PDis_{G_f}{}'$. Intuitively, we can obtain the query bounds of FSPQ through the path traffic-flow, if we prune the candidate path between v_i and v_j which is out of the bounds, the query speed can be improved.

Given query (v_i, v_j). When we obtain the FSPQ, we first compute the distance and traffic-flow of path $\rho(v_i, v_j)$ and then adopt Eq. 1 to obtain the final result. In the first step, if traffic-flow $TF_{G_f}{}^t(v_i, v_j)$ satisfies the follow lemma, we continue the processes. Otherwise, we can prune this path safely[4].

Lemma 4: Given query (v_i, v_j), the maximum flow $TF_{G_f\max}^{\ t}$, the minimum flow $TF_{G_f\min}^{\ t}$ and user constrained parameter η_u. We have the lower bound $LB_{TF} =$

[4]For simplicity, we adopt $TF_{G_f}{}^t$ to represent the $TF_{G_f}{}^t(v_i, v_j)$ in the Lemma 4.

$TF_{G_f}{}^t_{\min} - (TF_{G_f}{}^t_{\max} - TF_{G_f}{}^t_{\min}) \cdot (\alpha \cdot \eta_u)/(\eta_u - 1)(1-\alpha)$, and the upper bound $UB_{TF} = TF_{G_f}{}^t_{\min} + (TF_{G_f}{}^t_{\max} - TF_{G_f}{}^t_{\min}) \cdot (\eta_u - 1 - \alpha \cdot \eta_u)/(\eta_u - 1)(1-\alpha)$. We will prune every $\rho(v_i, v_j)$ if its traffic-flow $TF_{G_f}{}^t < LB_{TF}$ or $TF_{G_f}{}^t > UB_{TF}$ since this $\rho(v_i, v_j)$ can never become our goal.

Proof: Based on Def. 5 and the maximum constrained path distance, we can overwrite $\alpha \cdot PDis_{G_f}{}'$ in Eq. 1 as:

$$\alpha \cdot PDis_{G_f}{}' = \alpha \cdot \frac{PDis_{G_f}(v_i, v_j) - SPDis(v_i, v_j)}{(\eta_u - 1) SPDis(v_i, v_j)}$$

By scaling the equation, we have:

$$\alpha \cdot PDis_{G_f}{}' \leq \frac{\alpha}{(\eta_u - 1) SPDis(v_i, v_j)} \cdot PDis_{G_f}(v_i, v_j)$$

Due to $PDis_{G_f}(v_i, v_j)_{\max} = \eta_u \cdot SPDis(v_i, v_j)$ and the shortest flow-aware distance is located from 0 to 1, we can conclude the follow inequation as:

$$0 \leq \frac{\alpha \cdot \eta_u}{(\eta_u - 1)} + (1-\alpha) \cdot \frac{TF_{G_f}{}^t - TF_{G_f \min}}{TF_{G_f \max} - TF_{G_f \min}} \leq 1$$

By inequality transformation, we have the range of the traffic-flow $TF_{G_f}{}^t$ as:

$$TF_{G_f}{}^t \geq TF_{G_f}{}^t_{\min} - (TF_{G_f}{}^t_{\max} - TF_{G_f}{}^t_{\min}) \cdot \frac{\alpha \cdot \eta_u}{(\eta_u - 1)(1-\alpha)}$$

$$TF_{G_f}{}^t \leq TF_{G_f}{}^t_{\min} + (TF_{G_f}{}^t_{\max} - TF_{G_f}{}^t_{\min}) \cdot \frac{(\eta_u - 1 - \alpha \cdot \eta_u)}{(\eta_u - 1)(1-\alpha)}$$

In this way, we have obtained the upper and lower bounds of predicted flow $TF_{G_f}{}^t$. ■

Due to Eq. 1, our FPSPS algorithm can be divided into two stages: (1) in the first stage, we first compute the traffic-flow $TF_{G_f}{}^t(Q_u, D_u)$ of the given query $Q = \langle Q_u, D_u, t \rangle$, and then compute the spatial distance $PDis_{G_f}(Q_u, D_u)$; (2) in the second stage, we maintain the shortest flow-aware distance $FSD_{G_f}{}^t(Q_u, D_u)$ as Eq. 1 until there is no candidate path to be computed. Therefore, when we compute the $TF_{G_f}{}^t(Q_u, D_u)$ in the first, we can adopt the query bounds of traffic-flow to speedup the query efficiency. This is because when we prune this candidate path, we do not need to continue computing its distance which saves our space and time costs.

B. Flow Priority Shortest Path Search

Flow Priority Shortest Path Search Algorithm (FPSPS). With Eq. 1, Eq. 5, and Lemma 4, we are now ready to design the FPSPS algorithm. The pseudocode of the algorithm is shown in Alg. 5. We first set $FSD_{G_f}{}^t(Q_u, D_u)$ to help us gain the best optimal result (Line 1). Recall that FAHL stores the distance in label entries, we can generate a candidate path set which contains all relevant paths in the LCA node of $\chi(Q_u)$ and $\chi(D_u)$ based on Eq. 5 and obtain the flow value of each vertex by $\hat{P}_{total}$ (Lines 2 to 4). In this way, we can avoid traversing all the connectivity paths of (Q_u, D_u) to save more time costs. After that, we compute the shortest distance $SPDis_{G_f}(Q_u, D_u)$ by Alg. 2 which can help us to save the time cost since we do not need to enumerate all $\rho(Q_u, D_u)$. Then, with the help of Lemma 4, we can obtain the query bounds UB_{TF} and LB_{TF} easily (Lines 5 to 6).

In the previous steps, we have gained all necessary values including the shortest distance and query bounds. Now, we start to search the flow-aware shortest distance of each candidate $\rho \in Path_c$ (Lines 8 to 16). If the flow value of ρ is outside the query bounds, we can prune it directly, otherwise, we continue to compute its flow-aware distance until we obtain the shortest flow-aware distance $FSD_{G_f}{}^t(Q_u, D_u)$. Finally, we retrieve the shortest flow-aware path $FSP_{G_f}{}^t(Q_u, D_u)$ with $FSD_{G_f}{}^t(Q_u, D_u)$ (Line 17). The time complexity of FPSPS is $O(\varpi_T \cdot log(|\rho|))$, ϖ_T is the treewidth and $|\rho|$ is the number of candidate set.

Algorithm 5: *Flow Priority Search Algorithm*

Input: Query $Q = \langle Q_u, D_u, t \rangle$, the FAHL and predicted $\hat{P}_{total}$
Output: the flow-aware shortest path $FSP_{G_f}{}^t(Q_u, D_u)$

```
1  FSD_{G_f}^t(Q_u, D_u) ← +∞;
2  Obtain the LCA node L of χ(Q_u) and χ(Q_u)
3  Path_c ← generate candidate set by L, ρ(Q_u, D_u) and Eq. 5;
4  Obtain P̂_t(v_i) for each vertex at t;
5  Compute the shortest distance SPDis_{G_f}(Q_u, D_u);
6  Obtain LB_{TF} and UB_{TF} by Lemma 4;
7  SPDis(Q_u, D_u) ← +∞;
8  while ρ ∈ Path_c and Path_c ∉ ∅ do
9      Compute the traffic-flow TF_{G_f}^t of ρ(Q_u, D_u);
10     if TF_{G_f}^t > UB_{TF} or TF_{G_f}^t < LB_{TF} then
11         Prune ρ(Q_u, D_u);
12         Break;
13     else
14         Compute the flow-aware distance FAD on Eq. 1;
15         if FAD ≤ FSD_{G_f}^t(Q_u, D_u) then
16             FSD_{G_f}^t(Q_u, D_u) ← FAD;
17 Retrieve FSP_{G_f}^t(Q_u, D_u) with FSD_{G_f}^t(Q_u, D_u);
18 Return FSP_{G_f}^t(Q_u, D_u);
```

TABLE III: The Statistics of Different Datasets

Dataset	Vertices	Edges	Description	Default Records
BRN	28,342	38,577	Beijing	4,761,456
NYC	264,364	733,846	New York	44,413,152
BAY	321,270	800,172	Bay Area	53,973,360
COL	435,666	1,057,066	Colorado	73,191,888

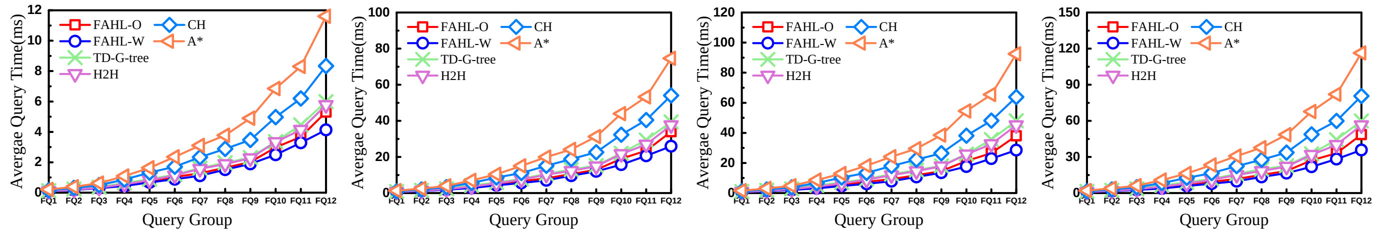
VI. Experiments

All algorithms in this section are implemented with C++. Besides, we evaluate all the methods on a Linux server with 2 Intel Xeon Gold 5122 CPU @ 3.60GHz CPUs and 128G running memory.

Adopted datasets. To fully evaluate our proposed FAHL, we adopt BRN, NYC, BAY, and COL four real datasets in this section. The first dataset comes from T-drive[5], and other three datasets are widely recognized public datasets that come from the DIMACS[6] Challenge. All four datasets represent a real-world city or state, BRN represents Beijing City, NYC represents New York City, BAY represents the Bay Area of

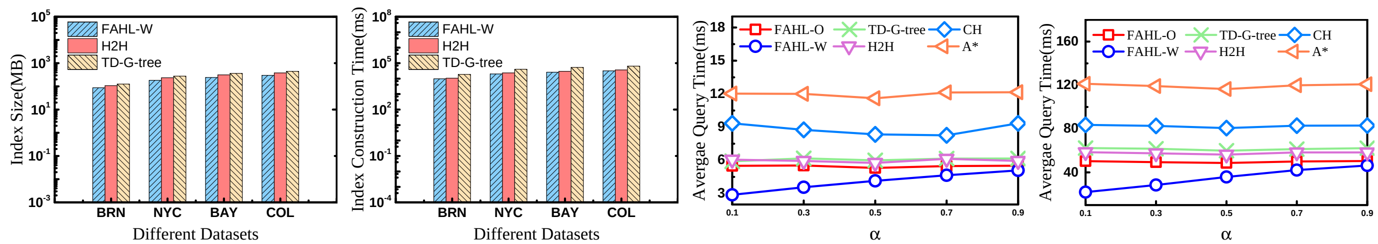
[5] https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/

[6] http://www.diag.uniroma1.it/challenge9/download.shtml



(a) BRN Dataset (b) NYC Dataset (c) BAY Dataset (d) COL Dataset

Fig. 6: Performance Evaluation on Different Query Groups



(a) Index Size (b) Index Construction Time (c) BRN with FQ_{12} (d) COL with FQ_{12}

Fig. 7: Performance Evaluation on Index Construction and Parameter α

San Francisco, and COL represents Colorado. As shown in Table III, we evaluate the proposed methods with vertices varying from 28K to 435K. To obtain FRNs, we adopt a well-recognized pre-trained model PDFormer [20] to obtain the traffic-flow for all datasets in 7 days. We set the default time interval for all datasets to 60 minutes, thus, each vertex can record 7×24=168 timesteps during 7 days. Finally, we have the total records varying from 4M to 73M.

Compared methods. We compare the efficiency of our FAHL index, maintenance algorithms, and pruning technology with four state-of-the-art methods, which can deal with the shortest path querying in flow-aware road networks. These methods are the following: (1) A^* [23], it is a heuristics straightforward algorithm without index. Thus, it does not need to spend time on index construction; (2) TD-G-tree is a tree index method with graph partition which can support the querying in time-dependent road networks, we implemented it on [13] and improved it by adding shortcuts on [24]; (3) CH [25], [26], it is a label index with contraction hierarchy; (4) H2H [22], [27], it is a label index based on tree decomposition with series efficient index updating algorithms. Meanwhile, we use FAHL-O to represent our method which does not take the prune bounds, and use FAHL-W to denote the method with the prune bounds. Methods denoted with + indicate that we have replaced the predicted $\hat{P}$ with $\hat{C}_f$.

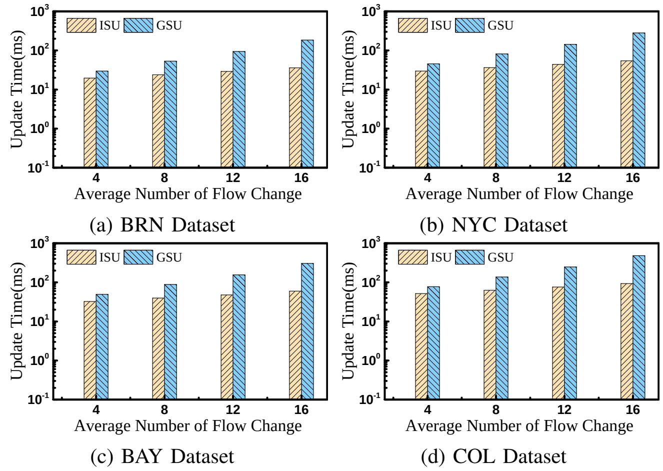
Experiment metrics. Based on the previously compared methods, we evaluate the performance of different methods in different aspects: (1) if a method belongs to the straightforward algorithm, we only evaluate its query time; (2) otherwise, we test its index size, indexing time, query time, and update time.

Experiment settings. We give the parameters and query settings of the experiments. For the parameters, the default correlation coefficient is set to $\alpha = \beta = 0.5$ for FSPQ and FAHL respectively. Besides, we set the user constrained parameter $\eta_u = 3$ for all datasets. For the index structure updating, we set the average number of changed weights as 4 as the default. To evaluate the performance with GSU and ISU algorithms, we give the average number of flow changes as $\{4, 8, 12, 16\}$. For the shortest path query, we randomly generate twelve groups of queries $FQ_i, i = 1, \ldots, 12$ for $Q = \langle Q_u, D_u, t\rangle$ as follows: let $FD_{\min} = {}^1\!/_4$ to represent the minimum flow-aware distance of the generated queries between Q_u and D_u, $FD_{\max}$ denotes the maximum distance. We set $m = (FD_{\min}/FD_{\max})^{1/12}$. Then, we generate 1,000 query Q of $FQ_i, i = 1, \ldots, 12$ at different time with the distance of (Q_u, D_u) falls into the range of $(FD_{\min} \cdot m^{i-1}, FD_{\min} \cdot m^i]$. The training epochs of PDFormer is 200.

A. Evaluation on Different Query Groups

Fig. 6 shows the query performance between different methods based on our generated query groups from FQ_1 to FQ_{12} in all datasets. The query performance is mainly related to the size and complexity of the datasets, the average query time for COL and BAY is obviously larger than NYC and BRN since COL and BAY have more total records, and all methods need to spend more time to obtain the path distance and path traffic-flow of the candidate ρ. Besides, with the increase of distance in FQ_i, the query time of all methods tends to increase. This is because, with the increase of query distance, more vertices and edges will be involved in the shortest path. In other words, more paths will be involved in the candidate set of the shortest path. Hence, in this situation, more time is needed to finish the path search.

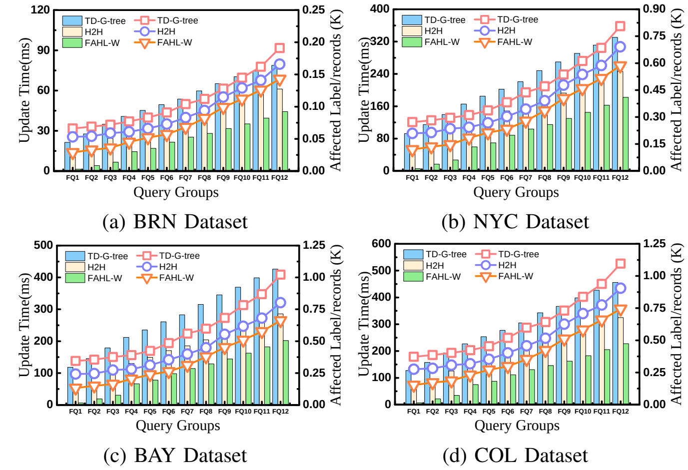
Next, we will discuss each method in detail. Due to A^* does not rely on index structure, it needs more time to traverse all possible vertices over spatial and temporal dimensions to search the result, so it has the worst performance for all datasets. Compared with CH, the query performance of TD-G-tree and H2H is close, and the performance of TD-G-tree is slightly weaker than that of H2H. This is mainly because TD-



(a) BRN Dataset (b) NYC Dataset

(c) BAY Dataset (d) COL Dataset

Fig. 8: Evaluation on Index Structure Updating



(a) BRN Dataset (b) NYC Dataset

(c) BAY Dataset (d) COL Dataset

Fig. 9: Evaluation on Index Label Updating

G-tree needs to traverse the tree structure to obtain the results, while H2H only needs to search the labeling list. Thus, TD-G-tree costs more time than H2H. Due to both G^*-tree and H2H do not specifically optimize for the traffic-flow, they can only obtain the optimal path through an iterative approach. So, it is obvious that their performances are worse than FAHL (FAHL-W and FAHL-O). FAHL adopts degree-flow joint ordering to build the tree decomposition which leads the vertices near the root to have a lower flow value. Therefore, FAHL is able to reduce the index overhead during the query processing. Without the prune bounds, such method can still guarantee the query efficiency of FAHL-O. FAHL-W achieves the best performance among all methods and is 33.1% faster than the best state-of-the-art H2H on average of all datasets. This is mainly because the FAHL-W further adopts a pruning method with the query bounds. Therefore, FAHL-W is more suitable for FSPQ in flow-aware road networks than other methods.

B. Evaluation on Index Construction and Parameter α

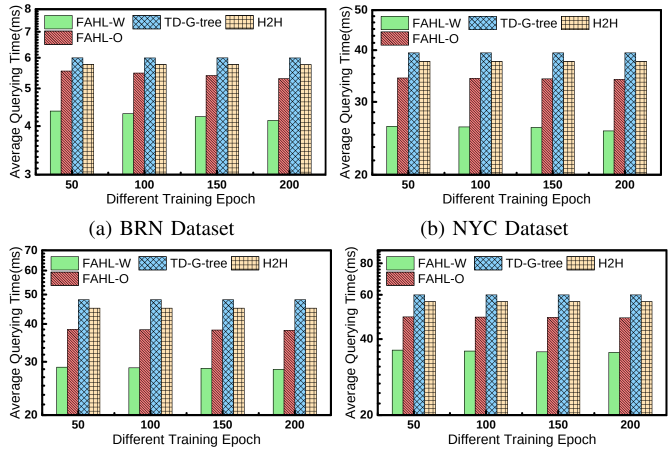
Fig. 7a and Fig. 7b show the result of index size and construction time. For FAHL-W and H2H, both of them need to build the labeling for each vertex in FRN, hence, their index size and construction time are largely determined by the number of vertices and edges in datasets. Even though both of them adopt tree decomposition to construct the index, there are still differences in the scale of the index. By adopting the degree-flow joint ordering method, FAHL-W can obtain a more suitable tree decomposition structure than H2H, resulting in FAHL-W having less index size and construction time than H2H. Furthermore, the timesteps and total records only have a little impact on the index size, this is because neither of the three methods can perceive the traffic-flow, they just build an index on the distance, and our FAHL-W is the only one method that utilizes the traffic-flow information.

Then, we evaluate the parameter α in BRN and COL datasets with FQ_{12}. Parameter α is used to measure the importance between the spatial distance and traffic-flow in FSPQ, Fig. 7c and Fig. 7d show the results. When α is small, the flow-aware distance is mainly influenced by the traffic-flow, otherwise, the flow-aware distance is mainly influenced by the spatial distance. However, FAHL-O, H2H, G^*-tree, CH, and A^* do not adopt the spatiotemporal specificity pruning methods, thus, the average query time of the above methods is almost invariant with different α. For FAHL-W, with the help of the pruning method, it has a better performance when the flow-aware distance is mainly influenced by traffic-flow. In other words, when α is small, we can prune more unnecessary candidates path by the lower and upper bounds of $TF_{G_f}{}^t$. Therefore, with the increase of the parameter α, the average query time of FAHL-W also increases. These results indicate that our proposed method is more suitable for FRN than other methods, especially when α is small.

C. Evaluation on Index Maintenance

The efficiency of index maintenance is a core criterion for evaluating the index's overall quality. In this section, we first evaluate the performance of index structure updating and then on label updating. Based on the previous query performance, we compare the maintenance of our proposed FAHL-W with H2H and TD-G-tree on all datasets.

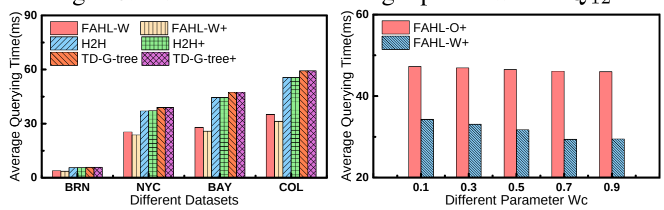
We first discuss the index structure update. Since TD-G-tree, H2H, and CH do not involve traffic-flow updates, we evaluate our GSU and ISU with FAHL here. As mentioned in Section IV-A, we have two index structure update algorithms GSU and ISU. As shown in Fig. 8, with the increase of an average number of flow changes, both GSU and ISU require more time to finish the structure update. Meanwhile, with the increase of data mount, the update time of both algorithms also increases. This is because more vertices' ordering may be influenced by the flow change. In this way, we need to check more nodes to ensure that we can obtain a correct structure update. While GSU is a general update method, for any vertex's traffic-flow change, GSU can handle the structure update by reconstructing the corresponding subtree. It is obvious that GSU involves lots of redundant calculations. For ISU, it only needs to exchange the relevant $\chi(v_i)$ and then check the entries in it to finish the update processes. In addition, compared with GSU, with the increase of the average number of flow changes, the update time of ISU increases more slowly, proving that ISU has



(a) BRN Dataset (b) NYC Dataset

(c) BAY Dataset (d) COL Dataset

Fig. 10: Evaluation on Training Epochs with FQ_{12}



(a) Varying dataset with $W_c = 0.5$ (b) Varying different W_c

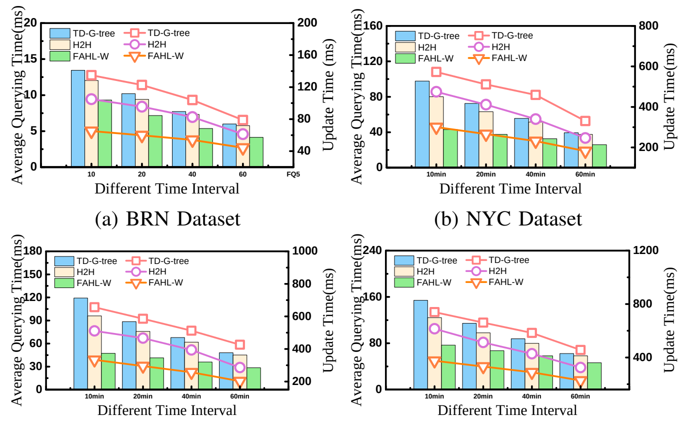
Fig. 11: Evaluation on Capacity-based Flow with FQ_{12}

stronger generalization ability. Therefore, ISU has a better performance than GSU.

We discuss the index label updating now. FAHL-W adopts the ILU algorithm to finish the label updating. For H2H and FAHL-W, we count the number of affected labels. For TD-G-tree, we count the number of updated records in the index. As shown in Fig. 9, with the growth of the distance in FQ_i, both update time and affected label/records increase. This is because the query distance increases from FQ_1 to FQ_9, the longer the path that needs to be searched, the more vertices may enter the shortest path set due to updates, resulting in an increased index updating number. TD-G-tree performs the worst among all methods, possibly because of its tree structure. Although H2H and FAHL-W also use tree decomposition to construct the index, however, in the label updating processes, the index structure is fixed and only updates the values in the label. In TD-G-tree, we need to update both the tree structure and its records simultaneously. Both H2H and FAHL-W need to update their labels, as FAHL-W adopts the ILU algorithm proposed in Section IV-B, hence, FAHL-W only needs to update the entries related to the original changed edges which avoids duplicate calculations. In this way, our FAHL-W outperforms H2H, and the result indicates that our proposed method is the most suitable method for FSPQ.

D. *Evaluation on Training Epochs and Capacity-based Flow*

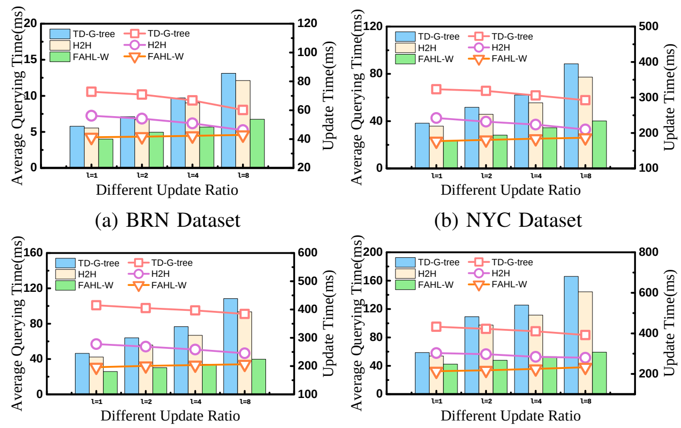
In this subsection, we first evaluate how the traffic-flow accuracy will affect the query performance. Since the learning of deep learning models is determined by epochs, so the larger the epoch, the higher the accuracy of the traffic-flow. Fig. 10 shows the result. The impact of estimation accuracy on TD-G-tree and H2H is negligible, this is mainly because both of them do not build on traffic information. It is sufficient to retrieve the relevant flow information during the query, and the accuracy of the predictions does not affect their index structure. For FAHL, higher estimation accuracy leads to better query performance and shorter query times. This is because FAHL's construction relies on traffic-flow. Lower estimation accuracy will lead us to obtain a wrong vertex ordering, resulting in more time to query the results. However, even at lower estimation accuracy, FAHL still outperforms existing methods. The results indicate the effectiveness of our index.



(a) BRN Dataset (b) NYC Dataset

(c) BAY Dataset (d) COL Dataset

Fig. 12: Evaluation on Time Intervals with FQ_{12}



(a) BRN Dataset (b) NYC Dataset

(c) BAY Dataset (d) COL Dataset

Fig. 13: Evaluation on Update Ratios with FQ_{12}

We are now evaluating how the capacity-based flow influences the query performance. Fig. 11 shows the results. As the dataset size increases, the query times for all methods also increase. Specifically, FAHL-W+ has better performance than FAHL-W, it indicates that capacity-based flow is better for us to identify the important vertices in the road network compared to adopting a single traffic-flow. However, $\hat{C}_f$ has almost no impact on H2H and TD-G-tree, this is because we only replace the previous predicted traffic-flow with $\hat{C}_f$, since both of H2H and TD-G-tree unable to perceive the flow information. When varying different W_c, we can find that, with the of increases W_c, the query time for FAHL-W+ decreases significantly faster than that of FAHL-O+. Because FAHL-W+ employs FPSPS, making the pruning effect more pronounced when the ratio of traffic-flow is higher in $\hat{C}_f$, and it achieves the best performance on $W_c = 0.7$.

E. Evaluation on Time Intervals and Update Ratios

In this subsection, we first evaluate the impact of different time intervals on query and update performance. The results are shown in Fig. 12. As the time interval increases, all methods' query and update times decrease. This is because shorter time intervals lead to higher frequencies of changes, requiring more time to complete the updates and queries. However, the time increase for FAHL is the smallest when the interval decreases, indicating that FAHL is the most suitable method for FSPQ compared to other methods.

Then, we evaluate on different update ratios of 10,000 updates with FQ_{12} in all datasets. Fig. 13 shows the results of the query and update times on different update ratios. As the update rate λ increases, the query time for all methods also increases, with H2H and TD-G-tree showing a significantly higher rate of increase compared to FAHL-W. This is because they incorporate traffic information during index construction. So, when locating the LCA node of query vertices, their LCA nodes may have higher traffic-flow. They need spend more time to traverse other possible nodes in a tree structure to compute FSPQ. As for index update time, H2H and TD-G-tree only consider the weight changes. Thus, their update times are decreasing with the reduction of edge weight updates. However, our FAHL adopts ILU and ISU algorithms to reduce redundant computations. Therefore, under different update ratios, FAHL still outperforms other algorithms.

VII. Related Work

Straightforward shortest path methods. These methods include famous Dijkstra [7] and A^* [23] algorithms, which can obtain the shortest path over a road network without any extra data. Relatively, we need to spend more time on them in large-scale datasets. To overcome this limitation, IER [28] and REAL [29] are proposed to extend Dijkstra and A^* respectively. IER will compute the distance between the candidate vertices to the search area to achieve a kind of fast verification. Besides, REAL obtains a better result by combining A^* with the reach-based approach. BOA^* [30] and EBA^* [31] are recent Bi-objective search algorithms by extending the A^* method. BOA^* can efficiently obtain the target path by introducing a consistent heuristic function. While EBA^* introduces early pruning to reduce the computational overhead of pruning and filtering during the path search process. Obviously, we can easily extend this kind of method to support the FSPQ. However, they are not optimized with traffic-flow, which leads them to search for flow only through exhaustive methods, requiring more time to obtain correct results when traffic changes during the query process, making it difficult to support online queries.

Index-based shortest path methods. These methods can be further divided into tree index and label index methods.

For tree index, G-tree [32], G^*-tree [24] and TD-G-tree [13] are representative methods. To construct the tree structure, G-tree [32] will first adopt graph partition technology [33] to generate a series subgraph of the entire graph, and then build the index based on the hierarchy of the subgraph and store the distance matrix in each node. Thanks to the graph partition technology, the tree index methods can easily support the large graph query. G^*-tree [24] and TD-G-tree [13] are proposed to extend G-tree. While G^*-tree takes shortcuts to reduce the time cost when the queries are located between leaf nodes in different subtrees of the index, and TD-G-tree stores the time weight functions to support the time-dependent road networks. To prevent traffic congestion caused by routing algorithms that are not aware of their results' influence on the traffic-flow, the SBTC framework with RR-index [34] has been proposed to manage the impact of routing results and enhance future route planning. RR-index adopts a binary tree structure to index route records in different time slices. However, as for the index update phase, this kind of method exhibits a bottleneck in efficiency since it first needs to traverse the tree structure and then update the changed entries in the node. Hence, they are not suitable for the FSPQ.

For label index, 2-hop labeling [35], CH [25], [26], [36]–[39], HL [40]–[42] and H2H [22], [27], [43]–[47] are almost the best efficient path find methods. These methods compute the label to store the distance between each vertex to its important vertices. For example, CH [37], [38] will first remove unimportant vertices and then build a shortcut to record the distance, if we adopt CH to answer FSPQ, the high-frequency changes will require CH to spend more memory and running time than obtain a single shortest path query since it may change the internal ordering set of CH. HL [40] builds the label with hub vertices which are filtered by a set of rules, however, it is hard to support the index in FSPQ since the flow and weight change will influence the selection of such "hub", resulting in more time to rebuild the index. H2H [22] builds the label through the tree decomposition, it is noteworthy that the success of H2H inspires us to revisit the tree decomposition for FSPQ. However, H2H only takes degree ordering to build the index and does not optimize for the traffic-flow information, which leads to H2H having a much larger index size than ours. Therefore, these methods are hard to get great results in flow-aware road networks since they do not consider the traffic information during the index construction. To address these problems, our FAHL reduces the size and query overhead of the index by fully leveraging the traffic-flow information.

Traffic-flow prediction. ARIMA [48] is a representative of statistical models, it requires complete and detailed training data. However, due to the complexity of the traffic environment, traffic data often has missing parts and contains noise. GNNs [19], [49] are widely used to capture the spatial temporal dependency. However, they are hard to handle this problem in long-range prediction. Transformer [50] and its variants like PDFormer [20] demonstrate better performance than the above methods in long-range prediction. This is because it decouples the feature into dynamic long-range spatial dependencies, and the time delay of traffic propagation. Inaccurate predictions may lead to an unsatisfactory vertex ordering of FAHL, thereby affecting the effectiveness of FSPQ. Therefore, we choose PDFormer in our work.

Routing techniques. In recent years, routing techniques have been proposed to find the optimal route in complex environments. SOR and SRH [51] are proposed to alleviate the congestion caused by excessive users following the original route algorithm recommendations. They have a good performance on current and anticipated future traffic statuses respectively. However, since these methods do not rely on an index, they may experience greater query overhead when faced with large-scale, high-frequency update scenarios. GRO [52] is proposed to achieve the global optimization route when both traffic conditions and routing evaluation are interdependent. However, repeatedly evaluating routes and traffic conditions in FSPQ will lead to redundant computations. Skyline pathfinding methods [53], [54] are widely applied to multi-dimensional optimal path query problems. For instance, [53] utilizes tree decomposition to hierarchically assign approximation ratios, thereby circumventing the skyline path search, [54] proposes a KTree to extend the skyline path into the keyword search. However, solving FSPQ with high-frequency updates by skyline methods incurs a substantial computational expense.

VIII. Conclusion

To efficiently answer the shortest path querying problem in flow-aware road networks, we propose a novel FAHL in this paper. In the index construction processes, we propose a degree-flow joint ordering method to obtain the vertex ordering in FRN based on spatial distance and traffic-flow two dimensions. With the help of joint ordering, FAHL reduces both index size and query overhead during the FSPQ. In the index maintenance processes, we propose Improved Structure Update (ISU) and Index Label Update (ILU) algorithms to support the index updating during the high-frequency of traffic-flow and edge weight. We also give theoretical proof to prove that the proposed algorithm can help us to reduce redundant calculations. In the querying processes, a flow priority shortest path search algorithm is proposed to find the final query result. We further propose a pruning method with query bounds on traffic-flow to improve the query performance. An extensive empirical study on four large datasets demonstrates that FAHL has better performance than the existing methods. In the future, we plan to extend our work to manage the FSPQ in constrained flow-aware road networks.

References

[1] B. Zheng, L. Bi, J. Cao, H. Chai, J. Fang, L. Chen, Y. Gao, X. Zhou, and C. S. Jensen, "Speaknav: Voice-based route description language understanding for template-driven path search," *Proceedings of the VLDB Endowment*, vol. 14, no. 12, pp. 3056–3068, 2021.

[2] T. Dan, X. Pan, B. Zheng, and X. Meng, "Double hierarchical labeling shortest distance querying in time-dependent road networks," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 2077–2089.

[3] X. Pan, J. Xu, and X. Meng, "Protecting location privacy against location-dependent attacks in mobile services," *IEEE Transactions on Knowledge & Data Engineering*, vol. 24, no. 08, pp. 1506–1519, 2012.

[4] X. Sun, Q. Ye, H. Hu, Y. Wang, K. Huang, T. Wo, and J. Xu, "Synthesizing realistic trajectory data with differential privacy," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[5] B. Zheng, K. Zheng, C. S. Jensen, N. Q. V. Hung, H. Su, G. Li, and X. Zhou, "Answering why-not group spatial keyword queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 1, pp. 26–39, 2018.

[6] B. Zheng, C. Huang, C. S. Jensen, L. Chen, N. Q. V. Hung, G. Liu, G. Li, and K. Zheng, "Online trichromatic pickup and delivery scheduling in spatial crowdsourcing," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 973–984.

[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, 2022, pp. 287–290.

[8] C. Sommer, "Shortest-path queries in static networks," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 1–31, 2014.

[9] Y. Wang, Q. Wang, H. Koehler, and Y. Lin, "Query-by-sketch: Scaling shortest path graph queries on very large networks," in *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, 2021, pp. 1946–1958.

[10] T. Dan, C. Luo, Y. Li, Z. Guan, and X. Meng, "Lg-tree: An efficient labeled index for shortest distance search on massive road networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 12, pp. 23 721–23 735, 2022.

[11] M. Zhang, L. Li, and X. Zhou, "An experimental evaluation and guideline for path finding in weighted dynamic network," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 2127–2140, 2021.

[12] M. Zhang, L. Li, W. Hua, and X. Zhou, "Efficient 2-hop labeling maintenance in dynamic small-world networks," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 133–144.

[13] Y. Wang, G. Li, and N. Tang, "Querying shortest paths on time dependent road networks," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1249–1261, 2019.

[14] J. Li, C. Ni, D. He, L. Li, X. Xia, and X. Zhou, "Efficient knn query for moving objects on time-dependent road networks," *The VLDB Journal*, vol. 32, no. 3, pp. 575–594, 2023.

[15] S. Zhang, C. Markos, and J. James, "Autonomous vehicle intelligent system: Joint ride-sharing and parcel delivery strategy," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 10, pp. 18 466–18 477, 2022.

[16] T. Wang, H. Luo, Z. Bao, and L. Duan, "Dynamic ridesharing with minimal regret: Towards an enhanced engagement among three stakeholders," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3712–3726, 2022.

[17] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire *et al.*, "Eta prediction with graph neural networks in google maps," in *Proceedings of the 30th ACM international conference on information & knowledge management*, 2021, pp. 3767–3776.

[18] Z. Chen, X. Xiao, Y.-J. Gong, J. Fang, N. Ma, H. Chai, and Z. Cao, "Interpreting trajectories from multiple views: A hierarchical self-attention network for estimating the time of arrival," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 2771–2779.

[19] S. Lan, Y. Ma, W. Huang, W. Wang, H. Yang, and P. Li, "Dstagnn: Dynamic spatial-temporal aware graph neural network for traffic flow forecasting," in *International conference on machine learning*. PMLR, 2022, pp. 11 906–11 917.

[20] J. Jiang, C. Han, W. X. Zhao, and J. Wang, "Pdformer: Propagation delay-aware dynamic long-range transformer for traffic flow prediction," *arXiv preprint arXiv:2301.07945*, 2023.

[21] S. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *arXiv preprint arXiv:1503.06462*, 2015.

[22] 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 *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 709–724.

[23] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[24] Z. Li, L. Chen, and Y. Wang, "g^*-tree: An efficient spatial index on road networks," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 268–279.

[25] X. Chen, Y. Peng, S. Wang, and J. X. Yu, "Dlcr: efficient indexing for label-constrained reachability queries on large dynamic graphs," *Proceedings of the VLDB Endowment*, vol. 15, no. 8, pp. 1645–1657, 2022.

[26] Y. Peng, J. X. Yu, and S. Wang, "Pspc: efficient parallel shortest path counting on large-scale graphs," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 896–908.
[27] J. Zhang, L. Yuan, W. Li, L. Qin, Y. Zhang, and W. Zhang, "Label-constrained shortest path query processing on road networks," *The VLDB Journal*, pp. 1–25, 2023.
[28] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *Proceedings 2003 VLDB Conference*. Elsevier, 2003, pp. 802–813.
[29] A. V. Goldberg, H. Kaplan, and R. F. Werneck, "Reach for a*: Shortest path algorithms with preprocessing." in *The shortest path problem*. Citeseer, 2006, pp. 93–139.
[30] C. Hernández, W. Yeoh, J. A. Baier, H. Zhang, L. Suazo, S. Koenig, and O. Salzman, "Simple and efficient bi-objective search algorithms via fast dominance checks," *Artificial intelligence*, vol. 314, p. 103807, 2023.
[31] L. Mandow and J.-L. Pérez de la Cruz, "Improving bi-objective shortest path search with early pruning," *ECAI 2023*, pp. 1680–1687, 2023.
[32] R. Zhong, G. Li, K.-L. Tan, L. Zhou, and Z. Gong, "G-tree: An efficient and scalable index for spatial search on road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2175–2189, 2015.
[33] G. Karypis and V. Kumar, "Analysis of multilevel graph partitioning," in *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, 1995, pp. 29–es.
[34] Z. Xu, L. Li, M. Zhang, Y. Xu, and X. Zhou, "Managing the future: Route planning influence evaluation in transportation systems," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 4558–4572.
[35] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *SIAM Journal on Computing*, vol. 32, no. 5, pp. 1338–1355, 2003.
[36] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7*. Springer, 2008, pp. 319–333.
[37] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 349–360.
[38] A. W.-C. Fu, H. Wu, J. Cheng, and R. C.-W. Wong, "Is-label: an independent-set based labeling scheme for point-to-point distance querying on large graphs," *Proceedings of the VLDB Endowment*, vol. 6, no. 6, p. 457, 2013.
[39] Z. Yu, X. Yu, N. Koudas, Y. Liu, Y. Li, Y. Chen, and D. Yang, "Distributed processing of k shortest path queries over dynamic road networks," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 665–679.
[40] Y. Zhang and J. X. Yu, "Hub labeling for shortest path counting," in *Proceedings of the 2020 ACM SIGMOD*, 2020, pp. 1813–1828.
[41] M. Zhang, L. Li, W. Hua, R. Mao, P. Chao, and X. Zhou, "Dynamic hub labeling for road networks," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 336–347.
[42] Q. Feng, Y. Peng, W. Zhang, Y. Zhang, and X. Lin, "Towards real-time counting shortest cycles on dynamic graphs: A hub labeling approach," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 512–524.
[43] 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," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 2372–2384.
[44] M. Yang, W. Li, W. Wang, D. Wen, and L. Qin, "Querying numeric-constrained shortest distances on road networks," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 2463–2475.
[45] Z. Liu, L. Li, M. Zhang, W. Hua, and X. Zhou, "Fhl-cube: multi-constraint shortest path querying with flexible combination of constraints," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 3112–3125, 2022.
[46] Y.-X. Qiu, D. Wen, L. Qin, W. Li, R.-H. Li, Z. Ying *et al.*, "Efficient shortest path counting on large road networks," *Proceedings of the VLDB Endowment*, 2022.
[47] B. Zheng, Y. Ma, J. Wan, Y. Gao, K. Huang, X. Zhou, and C. S. Jensen, "Reinforcement learning based tree decomposition for distance querying in road networks," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 1678–1690.
[48] B. M. Williams and L. A. Hoel, "Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results," *Journal of transportation engineering*, vol. 129, no. 6, pp. 664–672, 2003.
[49] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 922–929.
[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[51] L. Wang, R. C.-W. Wong, and C. S. Jensen, "Congestion-mitigating spatiotemporal routing in road networks," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 4586–4599.
[52] Y. Xu, L. Li, M. Zhang, Z. Xu, and X. Zhou, "Global routing optimization in road networks," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 2524–2537.
[53] Z. Liu, L. Li, M. Zhang, W. Hua, and X. Zhou, "Approximate skyline index for constrained shortest pathfinding with theoretical guarantee," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 4222–4235.
[54] D. Wu, Z. Zhang, C. S. Jensen, and K. Lu, "Efficient skyline keyword-based tree retrieval on attributed graphs," *IEEE Transactions on Knowledge and Data Engineering*, 2024.