Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 2050489, 18 pages
https://doi.org/10.1155/2021/2050489

Research Article

Tree Index Nearest Neighbor Search of Moving Objects along a Road Network

Wei Jiang,[1] **Fangliang Wei**,[1] **Guanyu Li**,[1] **Mei Bai**,[1] **Yongqiang Ren**,[1] **and Jingmin An**[1,2]

[1]*College of Information Science and Technology, Dalian Maritime University, Dalian 116026, China*
[2]*Faculty of Computer and Software, Dalian Neusoft University of Information, Dalian 116026, China*

Correspondence should be addressed to Guanyu Li; rabitlee@163.com

Received 23 June 2021; Revised 22 August 2021; Accepted 31 August 2021; Published 29 September 2021

Academic Editor: Pengfei Wang

With the widespread application of location-based service (LBS) technology in the urban Internet of Things, urban transportation has become a research hotspot. One key issue of urban transportation is the nearest neighbor search of moving objects along a road network. The fast-updating operations of moving objects along a road network suppress the query response time of urban services. Thus, a tree-indexed searching method is proposed to quickly find the answers to user-defined queries on frequently updating road networks. First, a novel index structure, called the double tree-hash index, is designed to reorganize the corresponding relationships of moving objects and road networks. Second, an index-enhanced search algorithm is proposed to quickly find the k-nearest neighbors of moving objects along the road network. Finally, an experiment shows that compared with state-of-the-art algorithms, our algorithm shows a significant improvement in search efficiency on frequently updating road networks.

1. Introduction

The rapid development of mobile communication and spatial positioning technology has extensively promoted the rise of location-based services (LBS). The LBS promotes the vigorous development of user-centric applications on urban transportation, and there has been widespread research in user recommendation systems and road network searching. One key issue of urban transportation is the nearest neighbor search of moving objects along a road network. However, the fast-updating operations of moving objects along a road network hinder the query response time of urban services. Currently, it is still difficult to quickly find the answers to user-defined queries on a frequently updating road network. Thus, an index-enhanced search algorithm is proposed to accelerate the response time to user-defined queries.

The response problem faced by user-defined queries on a road network is generalized as a k-nearest neighbor search problem (k-NN searching). Research on the k-NN problem in road networks has societal and commercial value [1]. For example, when someone uses the Meituan errand business, the system will send the order information to multiple salespersons nearest the user. Furthermore, k-NN technology promotes the improvement of mobile travel software, such as Didi and Uber, which frequently dispatches vehicles nearest to requesting users.

Figure 1 shows a simple example of a k-NN problem, which is a snapshot at time t. Assuming that $k = 4$, and there exists a user-defined query point q at time t, then, 4-NN searching can be conducted with different calculation strategies. Considering only the pairwise distance of a user and a moving object, then four moving objects (m_{12}, m_{13}, m_{11}, and m_8) are the closest to the query point q. However, taking into account the road intersection nodes and the moving objects' driving directions, the four moving objects (m_{13}, m_{11}, m_{12}, and m_{14}) respond to user q because the moving objects m_{12} and m_{14} can turn around on the road intersection nodes v_1 and v_3, respectively.

Existing research primarily focuses on the k-NN problem on road networks using Euclidean distance strategies

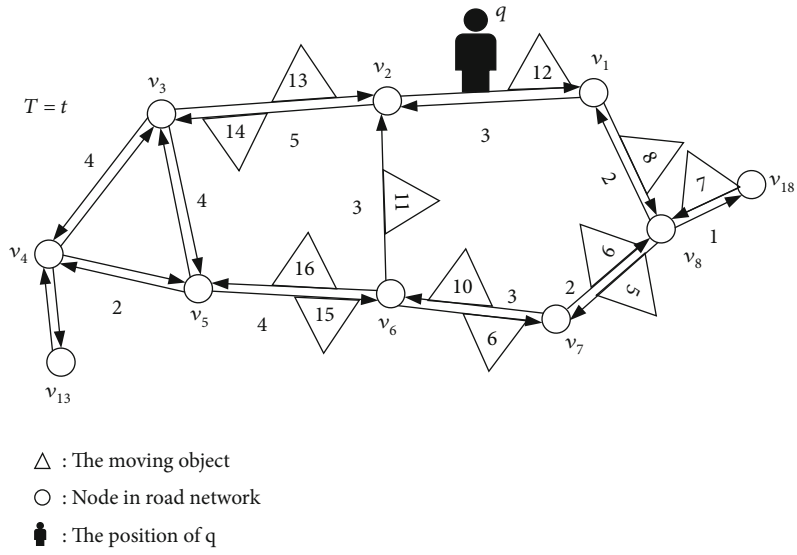


Figure 1: Segment of an urban road network.

for fixed objects [2–9] and Euclidean distance strategies for road-constrained moving objects [10–17].

The Euclidean distance strategies for fixed objects ignore the complexity of the road network, such as the nearest gas station on a highway, and prefer to calculate the shortest distance in a straight line in space, without considering the driving direction of vehicles and road intersection nodes. Road-constrained distance strategies more suitably address the complexity of transportation conditions and moving objects.

A preliminary work [17] provided an outstanding contribution to road-constrained distance strategies of moving objects. The work proposed a tree-based k-NN search algorithm. The method first constructs the balanced search-tree index, divides the whole road network into multiple partial road networks by hierarchical iteration, and then bounds the vertices to calculate the k-nearest objects. However, the strong search-tree index of the road network and moving objects results in a high cost to update the active records on the search-tree index.

A state-of-the-art work is the tree-decomposed k -NN search algorithm [18], which contains a tree-based decomposition strategy to establish the one-to-one relationship between each vertex and tree node. This method can quickly calculate the shortest distance between any two nodes; however, it needs to recalculate for updated inputs of k, resulting in increased time consumption thanks to redundant calculations.

Thus, we propose an index-enhanced search algorithm for the k-NN query problem of moving objects in a road network. The contributions of this paper are summarized as follows:

(1) A double tree-hash (DTH) index is designed to reorganize the weak relationships between the moving objects and the road network. The DTH index builds a weak bridge between tree and hash structures. The tree structure encapsulates the nodes, edges, and quantized relationships of intersubgraphs. The hash index records the moving objects. Through the weak bridge, the DTH index can realize the continuous updating-dynamic of moving objects in the road network

(2) An index-enhanced k-NN search algorithm is proposed to quickly find the k-nearest neighbors of moving objects on the road network. We call this the DMOK algorithm. Firstly, we establish an updating model to maintain the moving object updates while quickly and effectively expanding the positive nodes. Secondly, conditional distance rules are designed to reduce the number of expanded negative nodes. Finally, the k answers are responses to the user-defined requests and obtained using the DTH index

(3) Experiments with real data sets and artificial synthetic data sets verify the proposed algorithm's effectiveness

The structure of this paper is as follows. Section 2 reviews related work. Section 3 introduces the related definitions of the k-NN search problem for moving objects on a road network. A tree-hash index is presented in Section 4. Section 5 presents the index-enhanced k-NN search algorithm. Section 6 gives the experimental results and analysis. Conclusions are provided in Section 7.

2. Related Work

This section mainly summarizes the related research on the nearest neighbor search, including the related works on k-NN search problems for fixed objects (Section 2.1) and moving objects (Section 2.2) on a road network.

2.1. Related Works about k-NN Search Problem of Fixed Objects. The nearest neighbor search problem is one of the critical query types based on location services and was first proposed by Knuth in 1973 [19]. After ongoing research by many scholars, the theoretical knowledge has been enriched. The application field of the nearest neighbor

search has also been broadened, such that the nearest neighbor search owns a pivotal position in location-based services. The earliest studies on k-NN search problems mainly focused on fixed objects, and two classic k-NN search algorithms are IER [20] and INE [20].

Incremental Network Expansion (INE) adopted the greedy idea of Dijkstra's algorithm and expanded the answer by constantly asking the node closest to node q. The advantage of the INE algorithm is that the concept is clear, and the implementation is straightforward. However, its disadvantages are also apparent. For one, it needs to visit all the vertices close to vertex q. However, the number of these vertices may be much larger than k, and many of the visited nodes may be far away from query point q; thus, there is no need to visit them all.

In addition, the time and space complexities of the algorithm are high within large-scale data. For example, when using the priority queues to store the distance from other vertices to q, the algorithm's time complexity can reach $O(|W|\log|V|)$, where W is the number of edges and V is the number of vertices in a graph. Therefore, the INE algorithm has a good time and space complexity within small data sizes; it can easily make a preliminary judgment based on the input data size and then decide whether to choose the INE algorithm.

Kolahdouzan and Shahabi [21] proposed an improved algorithm, the Voronoi Network Nearest Neighbor (VN^3) algorithm, which answers k-NN queries by dividing the spatial network into multiple smaller Voronoi polygons based on the object-driven precalculating network distances. However, the performance of VN^3 depends on the density and distribution of the specific data set. As the data set becomes denser, the computational cost of calculating the distance each time becomes higher. Therefore, VN^3 is only suitable for sparse data sets.

The programming complexity of the Incremental Euclidean Restriction (IER) algorithm is higher than that of INE, and it needs to use data structures for preprocessing, such as R-tree. The advantage of the IER algorithm is that it avoids redundant nodes, especially nodes far away from query point q that should rarely be visited. The disadvantage of the IER algorithm is that it still needs to check the extreme cases of all nodes, and its worst time and space complexity is no less than that of the INR algorithm.

2.2. Related Works about the k-NN Search Problem of Moving Objects on a Road Network. Zhang et al. [22] proposed a keyword clustering query method (AKNN), which used user-specified query keywords to obtain text similarity. They designed a dual index structure (DG) generalized as a tree. The index structure saves the road network information and keeps the detailed knowledge of the network to establish an adjacency table. In literature [23], Huang et al. introduced the S-GRID algorithm to divide the spatial network into disjointed subnets and precalculated the shortest path of each pair within boundary nodes. In order to find k-nearest neighbors, network expansion is performed first within the subnet, and then, precalculated information is used to perform external expansion between boundary nodes. Samet et al. [10] proposed the DisBrw method, which precalculates the shortest path between all vertex pairs, and used a quadtree-based encoding for storing distance data. By storing the Euclidean distance as the shortest path distance boundary, they could find the k-nearest neighbors. Lee et al. [24] proposed a system framework called "ROAD" for spatial object searching on a road network. This method aimed at different object types effectively handled various location-related spatial queries, avoided the disadvantages that each node in the INE algorithm must be expanded, and directly skipped subnets with no obvious nodes of interest. Zhong et al. [25] proposed a balanced search-tree index (G-tree). G-tree uses a recursive segmentation method to divide the road network into corresponding subroad networks until the subnetwork is sufficiently small. This method can achieve an efficient search. Each node in G-tree is a subnet, and the leaf nodes in G-tree are equivalent to the nodes in a road network. If a leaf node and other external leaf nodes have directly connected edges, the leaf is the boundary of its upper network. Thus, G-tree calculates the shortest path of any two-leaf node boundaries on each divided network. In this paper, an assembly-based method is designed to calculate the minimum distance from the G-tree node to the query location and to reduce the complexity of the k-NN search problem.

Most studies on the underlying index structure employed the structures of R-tree [26] or other improved R-tree indices, such as $R*$-tree [27], FNR-tree [28], MON-tree [29], STCode-tree [30], and DBR$*$-tree [31]. Other works have used the general R-tree expansion, which converted two-dimensional space and one-dimensional time data into three-dimensional space data processing; examples include the 3DR-tree [32], RT-tree [33], and STR-tree [34]. Luo et al. [35] proposed a TOAIN algorithm. It used the SCOB index to optimize searching or update processing time and conducted workload analysis to configure SCOB indices to maximize throughput. Tianyang et al. [36] proposed a new algorithm for direction-aware KNN (DAKNN) searching of moving objects on a road network. This method used R-tree and a grid as the index structure, and the authors also proposed a novel local network extension method to quickly determine the direction of moving objects, reduce the communication costs, and simplify the calculation of the moving direction between moving objects and query points.

Two main issues should be considered: (1) road network environment representation and (2) moving object processing. Existing works mainly used data graphs to simulate the existing road network in terms of a road network environment [37]. $G = (V, E, W)$, where V is a node set and represents the connection between road segments, E is an edge set and illustrates the road segment in each road segment, and W means some additional information (road section cost, waiting time, etc.).

3. Problem Description

This paper focuses on quickly performing the k-NN search of moving objects on a road network environment and using the division strategy to improve the tree-mapping index

Table 1: Description of the symbols used in this paper.

Symbol	Meaning
K	Returns the number of moving objects
$\mathrm{Path}(v_1, v_2)$	Path from node v_1 to v_2
$\mathrm{Dist}(v_1, v_2)$	Distance from node v_1 to v_2
$\mathrm{SPath}(v_1, v_2)$	The shortest path from node v_1 to v_2
$\mathrm{SPDist}(v_1, v_2)$	The shortest path distance from node v_1 to v_2
M_N	Number of objects moved between two nodes
V_S	The starting node of the edge
V_E	End node of the edge
$\mathrm{AdSub}(v_i)$	Adjacency subgraph of v_i
$\mathrm{AcNode}(G_i)$	The active node of G_i
m.offset	The offset of the moving object from N_E
q.offset	Query the offset of point q from N_S
Cset	Candidate set of moving object results
R	k-NN query result set
$\mathrm{ShortChain}(v_i, v_j)$	The shortest chain from node v_i to v_j

mechanism for answering queries. Below, we introduce the definitions of the road network and associated concepts for the k-NN search on a road network environment. Table 1 lists and defines the main symbols used in this paper.

Definition 1 (road network). Given a directed weighted graph $G(V, E, W)$, where V is the road nodes and $V = \{v_1, v_2, \cdots, v_i, \cdots, v_n\}$. Each node v_i represents the intersection of roads on a road network, E is the set of directed edges on a road network, and $e_{1,2} = (v_1, v_2)$ means the edge from vertex v_1 to v_2. W represents the weights of the edge on the road network, in the paper, which indicates the distance of the edge $\mathrm{Dist}(v_1, v_2)$, formed as $w(v_1, v_2) = \mathrm{Dist}(v_1, v_2)$.

This paper studies the k-NN query problem of moving objects on a road network. Before introducing the definition of the k-NN query, some definitions of road networks and moving objects are given.

As shown in Figure 2, $\mathrm{Path}(v_6, v_8)$ represents the path from node v_6 to v_8 (for example, v_6-v_2-v_1-v_8 and v_6-v_7-v_8), and the Euclidean distance corresponding to the edge is $\mathrm{Dist}(v_6, v_8) = \mathrm{Dist}(v_6, v_2) + \mathrm{Dist}(v_2, v_1) + \mathrm{Dist}(v_1, v_8) = 8$ (or $\mathrm{Dist}(v_6, v_8) = \mathrm{Dist}(v_6, v_7) + \mathrm{Dist}(v_7, v_8) = 5$, etc.). On the road network, $\mathrm{SPath}(v_6, v_8)$ represents the shortest path from node v_6 to v_8. The shortest path of $\mathrm{SPath}(v_6, v_8)$ is v_6-v_7-v_8, which corresponds to the shortest distance $\mathrm{SPDist}(v_6, v_8) = 5$.

Definition 2 (moving object). On the road network, each moving object is expressed as $m = <(v_i, v_j), m.\mathrm{offset}, t>$; that is, at time t, the moving object m is on edge $<v_i, v_j>$, and the position offset from the vertex of v_j is m.offset.

As shown in Figure 2, the offset distance from m_{13} to the end node v_2 on the edge $e_{3,2}$ is $m_{13}.\mathrm{offset} = 2$, and $q.\mathrm{offset} = 1$ refers to the offset distance from the query object q to the start node v_2 on the edge $e_{1,2}$ which is 1.

Definition 3 (active node). A moving object m moves from node v_s to node v_e, and then, we call v_e a moving node of m. The active node of the subgraph G_i is denoted as $\mathrm{AcNode}(G_i)$.

For example, the moving object m_{13} moves from node v_3 to node v_2 on edge $< v_3, v_2 >$ in Figure 2. Therefore, v_2 is called an active node of m_{13}.

Definition 4 (moving object list). On a road network, a moving object is constantly moving, and its location information is regularly updated. The moving object list stores the positions of moving objects on the road network at a certain timestamp.

In Table 2, the moving object m_{11} exists on edge $E(v_6, v_2)$, and the distance from the terminal node v_2 is 1; meanwhile, the moving object m_{12} exists on edge $E(v_3, v_2)$, and the distance from the terminal node v_3 is 2. The number of moving objects with terminal node v_2 is 2. The moving object m_{14} exists on edge $E(v_2, v_3)$, and the distance from the terminal node v_3 is 2. Therefore, the number of moving objects with terminal node v_2 is 1.

4. Tree-Hash Index

This section introduces the double tree-hash (DTH) index proposed in this article, as well as the moving object update model (MOU-Model) based on the design of moving objects. We then introduce the processing method of moving object query on a road network, including the rapid solution of the road network distance and the fast processing of a query. Finally, a detailed introduction of the algorithm is given.

4.1. Division of Graphs. Due to the complexity of the road network, the calculation cost of the direct k-NN query is too high for our algorithm, and thus, we first divide graphs.

Definition 5 (k division of graph). Given a graph $G = (V, E, W)$, k division refers to dividing V into k subsets; that is, $V = V_1 \cup V_2 \cup \cdots \cup V_k$, and for any $i, j \in [1, k], i \neq j$, there is $V_i \cap V_j = \varphi$, $|V_i| = n/k$.

In the division of graphs, it is required that the number of edges E, where the associated vertices belong to different subsets, is minimized.

A tree hierarchy is established based on the graph division. The original graph is G, and graph G is divided iteratively as follows. Graph G is divided into $f = 2$ subgraphs G_1, G_2, G_3, and G_4; these subgraphs are regarded as subgraphs of G. The subgraph set of G is denoted by $C(G)$. G is the parent graph, and G_i^p represents the parent-child graph of subgraph G_i. Figure 3 shows the division process. For example, G_0 is divided into two subsubgraphs G_1 and

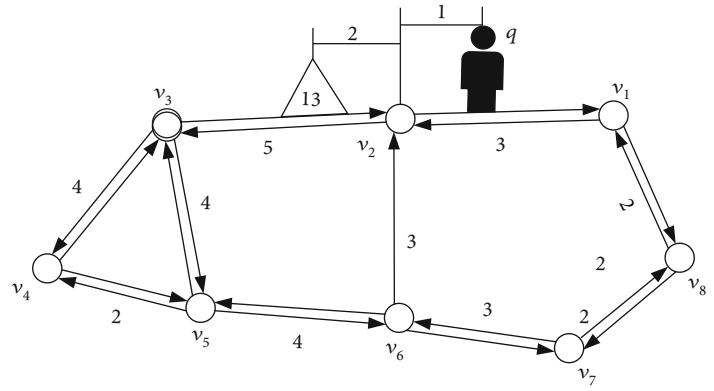


Figure 2: Moving objects under the road network with a query.

Table 2: Moving objects.

V_{end}	M	V_{start}	Offset	M_N
V_2	m_{11}, m_{13}	V_6, V_3	1,2	2
V_3	m_{14}	V_2	2	1

G_2, and then, $C(G_0) = (G_1, G_2)$ and $G_1^p = G_0$, as shown in Figure 3. Figure 4 is a subgraph of the road network.

Definition 6 (border). Given any one subgraph G_i, if node v_i ($v_i \in G_i$) is a boundary node, it needs to satisfy the following condition: there is a node v_j on the subgraph G_j ($i \neq j$), $v_i \in G_j$, such that $\langle v_i, v_j \rangle \in E$ is established. Thus, $B(G_i)$ is the set of boundary nodes of G_i.

4.2. DTH Index Construction. The double tree-hash (DTH) index is based on the characteristics of the road network structure and the characteristics of moving objects. By constructing a tree index structure, the shortest path of any two nodes in the road network can be quickly determined using the static and complex road network information; moreover, by constructing a distance hash table index structure, fast batch updates can be made for the frequently updated moving object information.

The divided subgraph information and the subgraph border nodes construct the road network tree structure, as shown in Figure 5.

Definition 7 (DisSet). The node distance set stores the global shortest distance between all nodes in each leaf subgraph and stores the global shortest distance between all border nodes in the nonleaf subgraphs with common ancestors. This is denoted as $\text{DisSet}(G_i) = \{S_i(v_i)\}$, $S_i(v_i) = \{v_j, \text{SPDist}(v_j, v_i), v_i, v_j \in G_i, i \neq j\}$. For the leaf subgraph, $S_i(v_i)$ stores the global shortest distance from other nodes in the subgraph to v_i; for nonleaf subgraphs, $S_i(v_i)$ stores the global shortest distance from other border nodes to border node v_i.

For example, for the leaf subgraph G_3 (Table 3), the global shortest distances between all nodes in the subgraph stored in the node distance set are stored in ascending order of $\text{DisSet}(G_3) = \{S_3(v_1), S_3(v_2), S_3(v_3), S_3(v_4), S_3(v_5), S_3(v_6), S_3(v_7), S_3(v_8)\}$, where $S_3(v_1) = \{(v_8, 2), (v_2, 3), (v_7, 4), (v_6, 4), (v_3, 8), (v_5, 10), (v_4, 12)\}$.

Tables 3 and 4 present the node distance sets of the leaf subgraph G_3 and the nonleaf subgraph G_2, respectively. The bold numbers in the tables are the distances between the border nodes and the other relevant border nodes.

Lemma 8 (see [17]). *Given a subgraph $G_i = (V_i, E_i, W_i)$, and vertices $v_i \in V_i$, $v_j \notin V_j$, the shortest distance from v_i to v_j must include a border node β of G_i. Then,*

$$SPDist(v_i, v_j) = SPDist(v_i, \beta) + SPDist(v_i, \beta). \quad (1)$$

Proof. Proof can be found in our technical report [38]. □

Corollary 9. *For the given two subgraphs G_i and G_j, the corresponding two nodes are $v_i \in G_i$, $v_j \in G_j$, and the two border nodes $\beta_i \in G_i$ and $\beta_j \in G_j$. Similarly,*

$$SPDist(v_j, v_i)SPDist(v_i, v_j) = SPDist(v_i, \beta_i) + SPDist(\beta_i, \beta_j) + SPDist(\beta_j, v_j). \quad (2)$$

Corollary 10. *For a given number of subgraphs G_i, G_j, and G_k, the corresponding nodes are $v_i \in G_i$, $v_j \in G_j$, and $v_k \in G_k$. G_k is the subgraph connecting G_i and G_j, and each subgraph corresponds to the border node $\beta_i \in G_i, \beta_j \in G_j$, $\beta_k \in G_k$. Similarly,*

$$SPDist(v_i, v_j) = SPDist(v_i, \beta_i) + SPDist(\beta_i, \beta_k) + SPDist(\beta_k, \beta_j) + SPDist(\beta_j, v_j). \quad (3)$$

Proof. According to Lemma 8 and Corollary 10, it can be proved. □

An example is shown in Figure 3, $v_{14} \in G_5, v_{31} \in G_6$, border nodes $v_{11} \in G_5, v_{10} \in G_6$, from Corollary 10, we

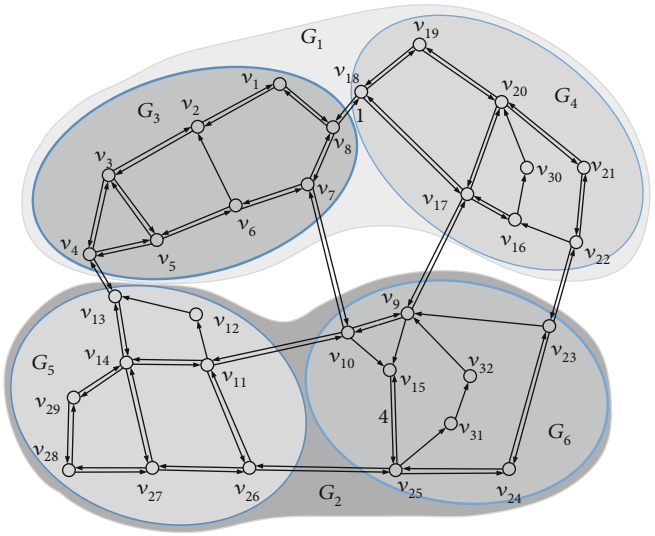


Figure 3: Subgraph division of road network.

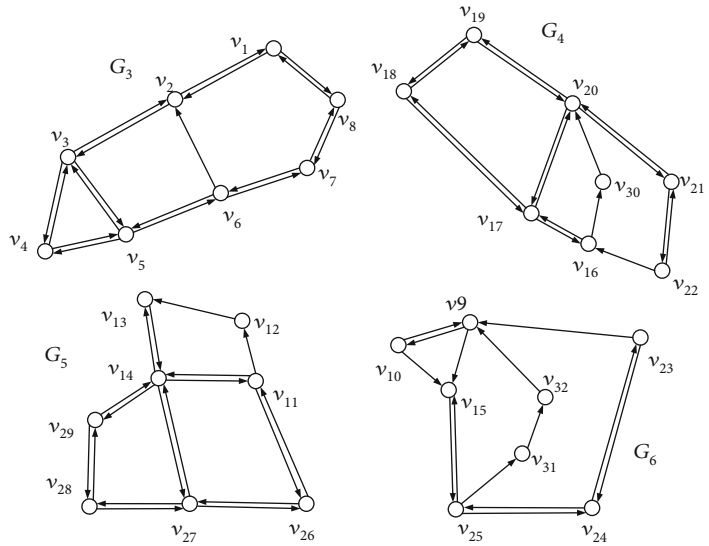


Figure 4: Subgraph of road network division.

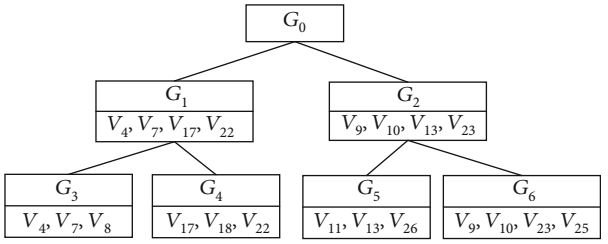


Figure 5: Tree structure diagram of the road network.

can get $\mathrm{SPDist}(v_{14}, v_{31}) = \mathrm{SPDist}(v_{14}, v_{11}) + \mathrm{SPDist}(v_{11}, v_{10}) + \mathrm{SPDist}(v_{10}, v_{31})$.

Definition 11 (activity-heap). Given a heap H, $v_i \in H$, if H satisfies $H = \{v_i \mid v_i \in B(G) \text{ or } v_i \in \mathrm{AcNode}(G)\}$, then H is called the active heap. This is denoted as H_{Ac}. If $H = \{v_i \mid v_i \notin B(G), v_i \notin \mathrm{AcNode}(G)\}$, then H is called an inactive heap and is denoted as H_{InAc}.

Corollary 12. *Assuming graph* $G = (V, E, W)$, $v_i \in G$, *then* $H_{Ac} \cup H_{InAc} = G(V)$.

Proof. According to Definition 11, it can be proved. Because $H_{\mathrm{Ac}} = \{v_i \mid v_i \in B(G) \text{ or } v_i \in \mathrm{AcNode}(G)\}$, $H_{\mathrm{InAc}} = \{v_i \mid v_i \notin B(G), v_i \notin \mathrm{AcNode}(G)\}$, $H_{\mathrm{Ac}} \cup H_{\mathrm{InAc}} = \{v_i \mid v_i \in B(G) \text{ or } v_i \in \mathrm{AcNode}(G)\} \cup \{v_i \mid v_i \notin B(G), v_i \notin \mathrm{AcNode}(G)\} = G(V)$. $H_{\mathrm{Ac}} \cup H_{\mathrm{InAc}} = G(V)$. □

4.3. DTH Index Process of Moving Objects. On the road network, the location information of moving objects (such as vehicles) is updated constantly. The V-tree index structure proposed in [17] needs to rerecord the states of the active vertices and update the recent global activities. In a busy road section, the position of moving vehicles changes

Table 3: S_3 node distance collection.

S_3	V_1	S_3	V_2		S_3	V_7	S_3	V_8
$\boldsymbol{V_8}$	**2**	V_1	3		$\boldsymbol{V_8}$	**2**	V_1	2
V_2	3	V_6	3		V_6	3	$\boldsymbol{V_7}$	**2**
$\boldsymbol{V_7}$	**4**	V_3	5	.....	V_1	4	V_2	5
V_6	6	$\boldsymbol{V_8}$	**5**		V_2	7	V_6	5
V_3	8	$\boldsymbol{V_7}$	**6**		V_5	7	V_5	9
V_5	10	V_5	7		$\boldsymbol{V_4}$	**9**	V_3	10
$\boldsymbol{V_4}$	**12**	$\boldsymbol{V_4}$	**9**		V_3	11	$\boldsymbol{V_4}$	**11**

Table 4: S_2 node distance collection.

S_2	V_{11}	S_2	V_{13}		S_2	V_{23}	S_2	V_{25}
$\boldsymbol{V_{10}}$	**5**	V_{11}	5		V_{25}	13	$\boldsymbol{V_9}$	**6**
$\boldsymbol{V_9}$	**6**	$\boldsymbol{V_{10}}$	**10**		$\boldsymbol{V_9}$	**19**	$\boldsymbol{V_{10}}$	**6**
$\boldsymbol{V_{13}}$	**6**	$\boldsymbol{V_9}$	**11**	.....	$\boldsymbol{V_{10}}$	**19**	V_{26}	10
V_{26}	7	V_{26}	12		V_{26}	23	$\boldsymbol{V_{11}}$	**11**
$\boldsymbol{V_{23}}$	**13**	$\boldsymbol{V_{23}}$	**18**		V_{11}	24	$\boldsymbol{V_{23}}$	**12**
V_{25}	16	V_{25}	18		$\boldsymbol{V_{13}}$	**30**	V_{13}	17

frequently in this environment. This update cost is very high, which is not conducive to query processing when moving objects change significantly. Thus, we propose a method based on updating moving objects. The road network structure diagram can be seen in Figure 6.

For example, in the bustling section of the city center, the number of moving vehicles is large, and the range of position changes is relatively large. The position of the moving object at $T = t$ and $T = t + 1$, and details of the moving objects are shown in Figure 7.

The existing k-NN algorithm only returns the k-nearest moving objects, regardless of their moving direction. After dividing the graph, according to the location relationship between the moving object and the query, it can be divided into the following situations.

As shown in Figure 8(a), the moving object is on the left side of the query, which is gradually approaching the query point along the road direction.

When $\mathrm{Dist}(v_2, v_1) - m_{12}.\mathrm{offset} \geq q.\mathrm{offset}$, the distance between the moving object and the query is

$$\mathrm{Dist}(m_{12}, q) = 2\mathrm{Dist}(v_2, v_1) - m_{12}.\mathrm{offset} - q.\mathrm{offset}. \quad (4)$$

Correspondingly, as shown in Figure 8(b), the moving object is on the right side of the query request point, that is, away along the road near the query point.

When $\mathrm{Dist}(v_2, v_1) - m_{12}.\mathrm{offset} < q.\mathrm{offset}$, it is necessary to recalculate the information of the nearest k moving objects. The distance between the moving object and the query is

$$\mathrm{Dist}(q, m_{12}) = \mathrm{Dist}(v_1, v_2) + m_{12}.\mathrm{offset} + q.\mathrm{offset}. \quad (5)$$

Table 5 lists the moving objects at time $T = t$. The gray entries denote nodes that are both a border node in the subgraph and an active node of the moving object.

For example, $G_4(2)$ means that at time t, there are two moving objects (m_{23} and m_{24}) in subgraph G_4, and they are moving from node v_9 to node v_{17}. v_{17} is both the border node of subgraph G_4 and the active node of the moving objects m_{23} and m_{24}.

Due to frequent updates of moving objects on the road network, we can establish the hash table as shown in Tables 6(a) and 6(b), which use V_E as the hash key and use $(V_S, M, \mathrm{Offset}, M_N)$ as the value; the mapping function is the $V_E \longrightarrow (V_S, M, \mathrm{Offset}, M_N)$.

For example, when key $= V_2$, value $= \{(V_6, m_{11}, 1, 2), (V_3, m_{13}, 2, 2)\}$, it means the moving object m_{11} is from V_6 to V_2, and $m_{11}.\mathrm{offset} = 1$, there are two moving objects on edge $<v_6, v_2>$.

5. Index-Enhanced k-NN Search Algorithm

This section mainly describes the proposed k-NN algorithm.

We present the calculation of the distance interval between two nodes and then the k-NN algorithm as a whole.

5.1. Node Expansion Strategy. Due to the complexity of the road network environment, calculating the global shortest distance between nodes requires an enormous time cost. This paper uses an inverted index structure to manage the distance calculated on the road network. Specifically, we sort the distance between two nodes from small to large and insert this information into the heap. When the distance between two nodes is an interval, we sort from small to large according to the upper bound of the interval. During subsequent calculations, we always process the first element in the heap H.

Given two nodes v_i and v_j, $v_i \in G_i, v_j \in G_j$, we quickly determine the global shortest distance between the two nodes.

Definition 13 (ShortChain). Given a subgraph $G_i = (V_i, E_i, W_i)$, for two nodes $v_i \in G_i$ and $v_j \in G_j$, the shortest chain of v_i, v_j refers to the DTH index. The path taken by the smallest common ancestor is denoted as $\mathrm{ShortChain}(v_i, v_j) = \{G_i \longrightarrow G_{i+1} \longrightarrow \cdots \longrightarrow G_j\}$, where G_i is the leaf subgraph to which v_i belongs and G_j is the leaf subgraph to which v_j belongs. The middle subgraph is the path of the divided subgraph (except for the common smallest ancestor) that G_i to G_j traverses.

For example, given two nodes v_{31} and v_2, the shortest chain from v_{31} to v_2 is $\mathrm{ShortChain}(v_{31}, v_2) = \{G_6 \longrightarrow G_2 \longrightarrow G_1 \longrightarrow G_3\}$, because $v_{31} \in G_6, v_2 \in G_3$, and $G_6 \neq G_3$, ${G_6}^p = G_2, {G_3}^p = G_1$, and $G_2 \neq G_1, {G_2}^p = G_0, {G_1}^p = G_0$, the node v_{31} and v_2 have the smallest common ancestor G_0. The shortest chain is shown in Figure 9.

Given two different nodes v_i and v_j, we have $v_i \in G_i, v_j \in G_j$, and $G_i \neq G_j$.

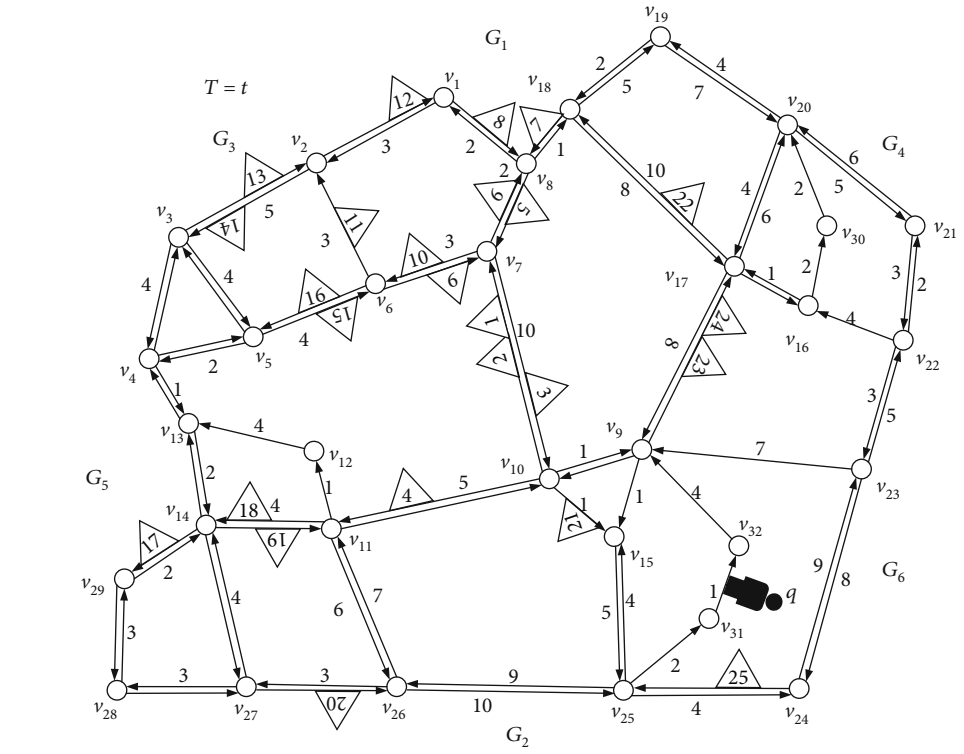
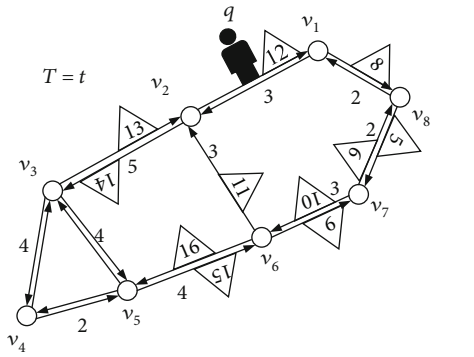
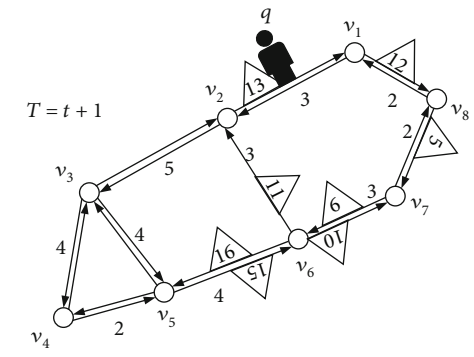


Figure 6: Road network structure diagram.

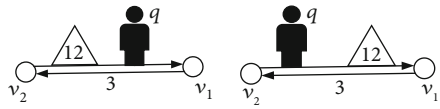


Moving object list ($T = t$)				
V_E	M	V_S	$Offset$	M_N
V_1	m_{12}	V_2	1	1
V_2	m_{11}, m_{13}	V_6, V_3	1,2	2
V_3	m_{14}	V_2	2	1
V_5	m_{16}	V_6	1	1
V_6	m_{10}, m_{15}	V_7, V_5	1, 1	2
V_7	m_5, m_9, m_1, m_2	V_8, V_6, V_{10}, V_{10}	1, 1.5, 2,4	4
V_8	m_7, m_8, m_6	$V1_8, V_1, V_7$	0.5, 0.5, 1	3
…	…	…	…	…



Moving object list ($T = t + 1$)				
V_E	M	V_S	$Offset$	M_N
V_1	m_{13}	V_2	2	1
V_2	m_{11}	V_6	1	1
V_4	m_{16}, m_{14}	V_6, V_5	1,1	2
V_6	m_6, m_{15}	V_7, V_5	2,1	2
V_7	m_5, m_{10}	V_8, V_6	1,2	2
V_8	m_{12}	V_1	1	1
…	…	…	…	…

Figure 7: Position of the moving objects at $T = t$ and $T = t + 1$.



(a) (b)

Figure 8: (a) M is on the left side of q; (b) M is on the right side of q.

Table 5: Moving objects at time $T = t$.

Moving object list ($T = t$)					
G_i	V_E	M	V_S	Offset	M_N
$G_3(14)$	V_1	m_{12}	V_2	1	1
	V_2	m_{11}, m_{13}	V_6, V_3	1, 2	2
	V_3	m_{14}	V_2	2	1
	V_5	m_{16}	V_6	1	1
	V_6	m_{10}, m_{15}	V_7, V_5	1, 1	2
	V_7	m_5, m_9, m_1, m_2	V_8, V_6, V_{10}, V_{10}	1, 1.5, 2, 4	4
	V_8	m_7, m_8, m_6	V_{18}, V_1, V_7	0.5, 0.5, 1	3
$G_4(3)$	V_{17}	m_{24}, m_{23}	V_9, V_9	2, 3	2
	V_{18}	m_{22}	V_{17}	2, 3	2
$G_5(5)$	V_{26}	m_{20}	V_{27}	1	1
	V_{29}	m_{17}	V_{14}	1	1
	V_{11}	m_4, m_{19}	V_{10}, V_{14}	2, 1.5	2
	V_{14}	m_{18}	V_{11}	2	1
$G_6(3)$	V_{10}	m_3	V_7	4	1
	V_{15}	m_{21}	V_{10}	0.3	1
	V_{25}	m_{25}	V_{24}	3	1

(1) We determine the leaf subgraph G_j where v_j is located and add the distances from all border nodes in G_j to v_j into the heap H, in ascending order

(2) We then process the first element v_{j-1} of the heap H and determine the next ShortChain subgraph as G_{j-1} according to the shortest chain ShortChain(v_j, v_i) sequence from v_j to v_i. Next, we calculate the respective border nodes of G_{j-1} to v_{j-1} and add them to the heap H in ascending order

(3) We end the expansion when the node expansion stop condition is met

When the node is expanded, the expansion is observed in heap H. When the following conditions are met for heap H, the node expansion ends:

(1) When the nodes in heap H are expanded in sequence

(2) When the nodes in the heap H have not been expanded, but the value of the first element of the heap H from the query point is greater than the set threshold or the maximum value in the candidate set

Table 6

(a) The hash table of moving objects ($T = t$)

Hash key (AcNode(V_i))	Value ($T = t$)			
V_E	M	V_S	Offset	M_N
V_1	m_{12}	V_2	1	1
V_2	m_{11}, m_{13}	V_6, V_3	1, 2	2
V_3	m_{14}	V_2	2	1
V_5	m_{16}	V_6	1	1
V_6	m_{10}, m_{15}	V_7, V_5	1, 1	2
V_7	m_5, m_9, m_1, m_2	V_8, V_6, V_{10}, V_{10}	1, 1.5, 2, 4	4
V_8	m_7, m_8, m_6	V_{18}, V_1, V_7	0.5, 0.5, 1	3
…	…	…	…	…

(b) The hash table of moving objects ($T = t + 1$)

Hash key (AcNode(V_i))	Value ($T = t + 1$)			
V_E	M	V_S	Offset	M_N
V_1	m_{13}	V_2	2	1
V_2	m_{11}	V_6	1	1
V_4	m_{16}, m_{14}	V_6, V_5	1, 1	2
V_6	m_6, m_{15}	V_7, V_5	2, 1	2
V_7	m_5, m_{10}	V_8, V_6	1, 2	2
V_8	m_{12}	V_1	1	1
…	…	…	…	…

Given two nodes, the calculation of the shortest chain is as shown in Algorithm 1.

5.2. DTH-Enhanced Nodewise Distance Calculation. This section mainly describes the algorithm for calculating the shortest distance between any two nodes in the road network environment based on the DTH index structure. This DTH-based algorithm to calculate the distance between two nodes (called DCDP) combines the advantages of the DTH index and can complete a search quickly. At the same time, we use the node expansion strategy to reduce unnecessary node expansion and network overhead.

Definition 15 (conditional distance). Given three different nodes v_i, v_j, and v_k, $v_i \in G_i, B(G_i) = \{v_k\}, B(G_j) = \{v_j\}$, and $G_i \neq G_j$. The distance from v_i through G_i's border node v_k to v_j in the subgraph G_j is called the distance from node v_i to v_j under the condition of passing node v_k. It is denoted as Dist$((v_i, v_j) \mid v_k)$.

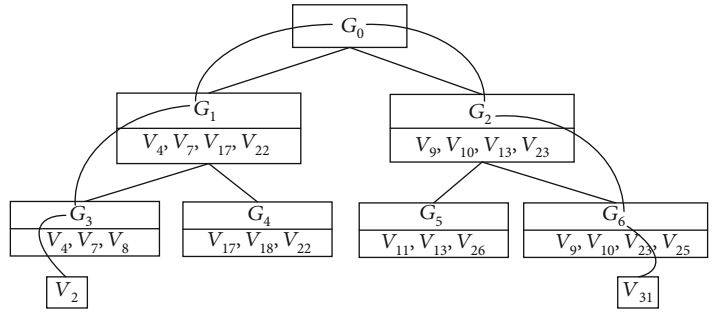


Figure 9: The shortest chain graph of nodes v_{31} to v_2.

Input: $v_i \in G_i, v_j \in G_j$
Output: **ShortChain**(v_i, v_j)
1 Initialize **ShortChain**$(v_i, v_j) = \{\varnothing\}$;
2 Make sure the subgraphs G_i, G_j where v_i, v_j are located;
3 while $G_i \neq G_j$ do
4 Add G_i, G_j to **ShortChain**(v_i, v_j), get **ShortChain**$(v_i, v_j) = \{G_i \longrightarrow G_j\}$;
5 Determine the parent node $G_i^p G_j^p$ of G_i, G_j;
6 $G_i = G_i^p$; $G_j = G_j^p$;
7 Output **ShortChain**(v_i, v_j)

Algorithm 1: The algorithm for calculating the shortest chain.

For example, for $v_{31} \in G_6$, we have $B(G_6) = \{v_{25}\}$ and $B(G_5) = \{v_{11}\}$. The distance from node v_{11} to node v_{31} through border node v_{25} is expressed as $\text{Dist}((v_{11}, v_{31}) \mid v_{25})$.

Lemma 16. *Given two nodes v_i and v_j, the shortest path must pass through the border nodes of the divided subgraphs corresponding to ShortChain(v_i, v_j), where multiple divided subgraphs can fit the same border node.*

In the above example, $\text{ShortChain}(v_{31}, v_2) = \{G_6 \longrightarrow G_2 \longrightarrow G_1 \longrightarrow G_3\}$. Then, the shortest path from v_{31} to v_2 is $v_{31}(G_6) \longrightarrow v_9(G_6, G_2) \longrightarrow v_7(G_1, G_3) \longrightarrow v_2(G_3)$.

We can directly obtain the distance between two nodes in the same subgraph through the node distance set, and thus, there is no need to calculate the shortest chain. For nodes in two different subgraphs, the process of calculating the shortest chain is as in Algorithm 1.

The process of calculating the global shortest distance between any two nodes is given below. According to the node distance set, the global shortest distance can be quickly obtained for two nodes in the same leaf subgraph (v_s, v_e):

(a) Use Algorithm 1 to calculate the shortest chain $\text{ShortChain}(v_e, v_s)$ between v_e and v_s; calculate the distance from the subgraph where v_s is located to the border node of the next chain subgraph according to the shortest chain sequence, and store it in heap H (whose elements are sorted from small to large)

(b) Take the first element V_{cur} of the heap H and expand according to the node expansion strategy

(c) Iterate successively until the next chain subgraph of the first element V_i of the first heap H is the same as the subgraph where v_e is located. Calculate $\text{Dist}(v_s, v_e)$, and temporarily store its value in $\text{Dis t}'(v_s, V_i)$

(d) Calculate other elements in heap H in turn. When $\text{Dist}(v_s, V_{cur}) < \text{Dist}'(v_s, V_i)$, $\text{SPDist}(v_s, v_e) = \text{Dist}(v_s, V_{cur})$; when $\text{Dist}(v_s, V_{cur}) \geq \text{Dist}'(v_s, V_i)$, terminate the algorithm

In Section 5.1, we introduce the algorithm used to calculate the shortest chain. Based on the DTH index and Algorithm 1, we propose the DCDP algorithm to calculate the shortest distance between any two nodes quickly. The $\text{SPDist}(v_{31}, v_2)$ solution process is shown in Figure 10. The specific description is given in Algorithm 2.

For example, the process of solving $\text{SPDist}(v_{31}, v_2)$ is as follows:

(1) According to Algorithm 1, the shortest chain from v_2 to v_{31} is $\text{ShortChain}(v_2, v_{31}) = \{G_3 \longrightarrow G_1 \longrightarrow G_2 \longrightarrow G_6\}$

(2) According to the order of the shortest chain $\text{ShortChain}(v_2, v_{31})$, expand using the node expansion strategy

Processing node	Border point of the next chain graph	Expansion of this node	Heap HS
$v_2 \in G_3$	$G_3(v_4, v_7, v_8)$	$SPDist(v_4, v_2) = 9$ $SPDist(v_7, v_2) = 6$ $SPDist(v_8, v_2) = 5$	$\boxed{v_8(5)}$ $v_7(6)$ $v_4(9)$
$v_8(5) \in G_3$	$G_1(v_4, v_7, v_{17}, v_{22})$	$Dist((v_4, v_2)\|v_8) = 16$ $Dist((v_7, v_2)\|v_8) = 7$ $Dist((v_{17}, v_2)\|v_8) = 16$ $Dist((v_{22}, v_2)\|v_8) = 18$	$\boxed{v_7(6)}$ $v_4(9)$ $v_{17}(16)$ $v_{22}(18)$
.....	.....	.....	.....
$v_{10}(16) \in G_6$	v_{31}	$Dist((v_{10}, v_2\|v_{31}) = 24$	$\boxed{v_{17}(16)}$ $v_9(17)$ $v_{22}(18)$ $v_{23}(28)$ $v_{25}(18)$ $v_{31} \to v_2(24)$
$v_{17}(16) \in G_1$	$G_2(v_9, v_{23})$	$Dist((v_9, v_2\|v_{17}) = 24$ $Dist((v_{23}, v_2\|v_{17}) = 26$	$\boxed{v_9(17)}$ $v_{22}(18)$ $v_{23}(26)$ $v_{25}(28)$ $v_{31} \to v_2(24)$
$v_9(17) \in G_6$	v_{31}	$Dist((v_{31}, v_2\|v_9) = 22$	$\boxed{v_{23}(23)}$ $v_9(24)$ $v_{25}(28)$ $v_{31} \to v_2(22)$
$v_{23}(23)$	$Dist(v_{23}, v_2) = 23 > Dist(v_{31}, v_2) = 22$		$v_{31} \to v_2(22)$

Figure 10: $\text{SPDist}(v_{31}, v_2)$ solution process.

Input: Given two nodes v_s, v_e, $\textbf{Dist}(v_s, v_e) = \infty$
Output: $\textbf{SPDist}(v_s, v_e)$
1 If v_s, v_e belong to the same subgraph, directly output $\textbf{SPDist}(v_s, v_e)$ according to the node distance set;
2 Determine the shortest chain $\textbf{ShortChain}(v_s, v_e)$;
3 $V' = \varnothing$; $V_{cur} = \varnothing$; Let $\textbf{Dist}(v_s, V') = \infty$;
4 Add v_e to heap H;
5 Make sure the subgraphs G_i, G_j where v_i, v_j are located;
6 **while** *H is not empty* **do**
7 Take V_{cur}, the first element of H;
8 **if** $\textbf{Dist}(v_s, V_{cur}) < \textbf{Dist}(v_s, V')$ **then**
9 break;
10 Get G_i's next chain graph G_j according to **ShortChain**;
11 **for** *every border point* V_i of G_j **do**
12 if $\textbf{Dist}(v_s, V_{cur}) < \textbf{Dist}(v_s, V_i)$ then
13 $\textbf{Dist}'(v_s, V_i) < \textbf{Dist}(v_s, V_{cur}) + Dist(v_s, V_i)$
14 **if** $\textbf{Dist}'(v_s, V_i) < \textbf{Dist}(v_s, V_i)$ **then**
15 $\textbf{Dist}(v_s, V_i) = \textbf{Dist}'(v_s, V_i)$
16 Add $V_i(\textbf{Dist}(v_s, V_i))$ to the heap H in descending order of value;
17 $V' = V_{cur}$;
18 $\textbf{SPDist}(v_s, v_e) = \textbf{Dist}(v_s, V_{Cur})$

Algorithm 2: DCDP algorithm().

(3) When the first element of heap H is v_9, calculate $\text{Dist}(v_{31}, v_2) = 22$

(4) When the expansion node is the first element v_{23} of heap H, $\text{Dist}(v_{23}, v_2) = 23 > \text{Dist}(v_{31}, v_2) = 22$, stop node expansion and terminate the algorithm

(5) Output $\text{SPDist}(v_{31}, v_2) = 22$

For any nodes v_i, v_j, v_k and subgraph G_j, G_j, the node distance set $\text{DisSet}(D_j)$, where $v_i \notin G_j, v_j \in G_j, v_k \in G_j$, and $\text{SPDist}(v_i, v_j)$ can be directly calculated by Algorithm 2. Through the node distance set $\text{DisSet}(D_j)$, we can obtain

Table 7: Distances to the border nodes between G_5 and G_6.

$B(G_5) \setminus B(G_6)$	$v_9(8)$	$v_{10}(8)$	$v_{23}(14)$	$v_{25}(2)$
v_{11}	6	5	24	11
v_{13}	12	11	30	17
v_{26}	13	12	23	10

the distance $\mathrm{SPDist}(v_j, v_k)$ between any two nodes in the subgraph. Then,

$$\mathrm{SPDist}(v_i, v_k) = \mathrm{SPDist}(v_i, v_j) + \mathrm{SPDist}(v_j, v_k). \quad (6)$$

5.3. Conditional Rules. To reduce the calculations required to obtain the distance between two nodes and perform node expansion, we propose a reduction strategy. First, through the expansion of the node, a distance interval is approximated, and the upper and lower bounds of the distance between the two nodes are determined. Then, in the expansion calculation, according to the upper and lower bounds determined above, we reduce the amount of calculation.

Definition 18 (distance interval). Given two nodes v_i and v_j ($v_i \in G_i, v_j \in G_j$) in different subgraphs, the maximum distance between node v_i and node v_j in subgraph G_j is called the upper bound of $\mathrm{Dist}(v_i, v_j)$ and is denoted as $\mathrm{upDist}(v_i, v_j)$. The minimum value of this distance is called the lower bound of $\mathrm{Dist}(v_i, v_j)$ and is represented as $\mathrm{lowDist}(v_i, v_m)$. The distance interval from node v_i to node v_j is $\mathrm{Dist}(v_i, v_j) = [\mathrm{lowDist}(v_i, v_m), \mathrm{upDist}(v_i, v_j)]$.

Lemma 19 (conditional distance interval). *Given nodes v_i and v_j, and $v_i \in G_i, v_j \in G_j$, and $v_k \in B(G_j)$, through node v_k, the conditional distance interval from node v_i to node v_j is as follows:*

$$\begin{aligned} lowDist\left((v_i, v_j) \mid v_k\right)_d &= \min\left\{Dist\left(v_i, B\left(G_j\right)\right)\right\} \\ &\quad + Dist\left(v_k, v_j\right), upDist\left((v_i, v_j) \mid v_k\right) \\ &= Dist\left(v_i, v_k\right) + Dist\left(v_k, v_j\right), \end{aligned} \quad (7)$$

and thus, $Dist((v_i, v_j) \mid v_k) = [lowDist((v_i, v_j) \mid v_k), upDist((v_i, v_j) \mid v_k)]$.

For example, to solve the distance interval between the node in G_5 through the border node v_{25} to v_{31}, that is, $\mathrm{Dist}((B(G_5), v_{31}) \mid v_{25})$, the distances to the border nodes between G_5 and G_6 are showed in Table 7.

$v_{31} \in G_6, v_{25} \in G_6$, according to Lemma 8, the distance from node to node v_{31} in subgraph G_5 must pass through the border node of G_5. $B(G_5) = \{v_{11}, v_{13}, v_{26}\}, B(G_6) = \{v_9, v_{10}, v_{23}, v_{25}\}, \mathrm{SPDist}(v_{25}, v_{31}) = 2$. The distances of the G_5 border node $B(G_5) = \{v_{11}, v_{13}, v_{26}\}$ to node v_{25} are $\mathrm{SPDist}(v_{11}, v_{25}) = 11, \mathrm{SPDist}(v_{13}, v_{25}) = 17$, and $\mathrm{SPDist}(v_{26}, v_{25}) = 10$. $\mathrm{SPDist}(v_{11}, v_9) = 6, \mathrm{SPDist}(v_{11}, v_{10}) = 5, \mathrm{SPDist}(v_{11}, v_{23}) = 24$, and $\mathrm{SPDist}(v_{11}, v_{25}) = 11$. Therefore, the distance of border nodes v_{11}, v_{13}, and v_{26} to v_{31} through v_{25} is

$$\begin{aligned} \mathrm{Dist}((v_{11}, v_{31}) \mid v_{25})_d &= \min\{\mathrm{Dist}(v_i, v_k)\} + \mathrm{Dist}(v_k, v_j) \\ &= \mathrm{SPDist}(v_{11}, v_{10}) + \mathrm{SPDist}(v_{25}, v_{31}) = 7, \\ \mathrm{Dist}((v_{11}, v_{31}) \mid v_{25})_u &= \mathrm{Dist}(v_i, v_k) + \mathrm{Dist}(v_k, v_j) \\ &= \mathrm{SPDist}(v_{11}, v_{25}) + \mathrm{SPDist}(v_{25}, v_{31}) = 13. \end{aligned} \quad (8)$$

$\mathrm{Dist}((v_{11}, v_{31}) \mid v_{25}) = [7, 13]$, expressed. The lower bound of the distance of v_{11} through node v_{25} to v_{31} is 7, and the upper bound is 13. Similarly, $\mathrm{Dist}((v_{13}, v_{31}) \mid v_{25}) = [13, 19]$, and $\mathrm{Dist}((v_{26}, v_{31}) \mid v_{25}) = (12)$.

Definition 20 (minimum value of subgraph). Given node v_i, $v_i \in G_i$, and subgraph G_j, the minimum distance between node v_i and $B(G_j)$ is called the minimum value of the subgraph and is denoted as $\mathrm{minDist}(v_i, G_j)$.

Corollary 21. *Given the subgraphs G_i and G_j, where $i \neq j$, then $minDist(G_i, G_j) = \min\{Dist(B(G_i), B(G_j)\}$.*

Proof. We know from Corollary 9 that the distance from node v_i outside the subgraph to the subgraph G_j must pass through the border node $B(G_j)$ of the subgraph G_j. Similarly, the distance between the node of the subgraph G_j and the subgraph G_i must pass through the border node $B(G_i)$ of the subgraph G_i. Therefore, $\mathrm{minDist}(G_i, G_j) = \min\{\mathrm{Dist}(B(G_i), B(G_j)\}$, which can be obtained from the node distance set. The proof is complete. □

For example, from Table 7, it can be seen that $\mathrm{minDist}(G_5, G_6) = \mathrm{Dist}(v_5, v_{10}\} = 5$; that is, the shortest distance from subgraph G_5 to subgraph G_6 is 5.

For distance interval values with the same starting node and ending node, the upper bound takes a smaller value, and the lower bound takes a more considerable value.

Theorem 22. *Given that the nodes $v_i \in G_i, v_j \in G_j$, and $v_k \in B(G_k)$, and $G_i^p = G_k^p$, and $Dist(v_k, v_j) = K$, for the first determined value node v_k of the heap H, if $lowDist((v_i, v_j) \mid v_k) = upDist((v_i, v_j) \mid v_k)$, that is, $Dist((v_i, v_j) \mid v_k)$ is a certain value, then $Dist((v_i, v_j) \mid v_k)$ is the shortest distance through node v_k. Moreover, node v_i does not need to be expanded in subsequent node expansion.*

Proof. Suppose that there are nodes v_i', v_j, and $v_k, v_i' \in G_i$, $v_j \in G_j$, and $v_k \in B(G_k)$. $\mathrm{Dist}((v_i', v_j) \mid v_k)$ is the smallest value because $\mathrm{lowDist}((v_i', v_j) \mid v_k) = \min\{\mathrm{Dist}(v_i', B(G_j))\} + \mathrm{Dist}(v_k, v_j)$. Only $\mathrm{Dist}(v_i', B(G_j))$ takes the minimum value, so $\mathrm{Dist}(v_i', v_k)$ takes the minimum value, because $\mathrm{Dist}((v_i, v_j) \mid v_k)$ takes the minimum value. Similarly, $\mathrm{Dist}(v_i, v_k)$

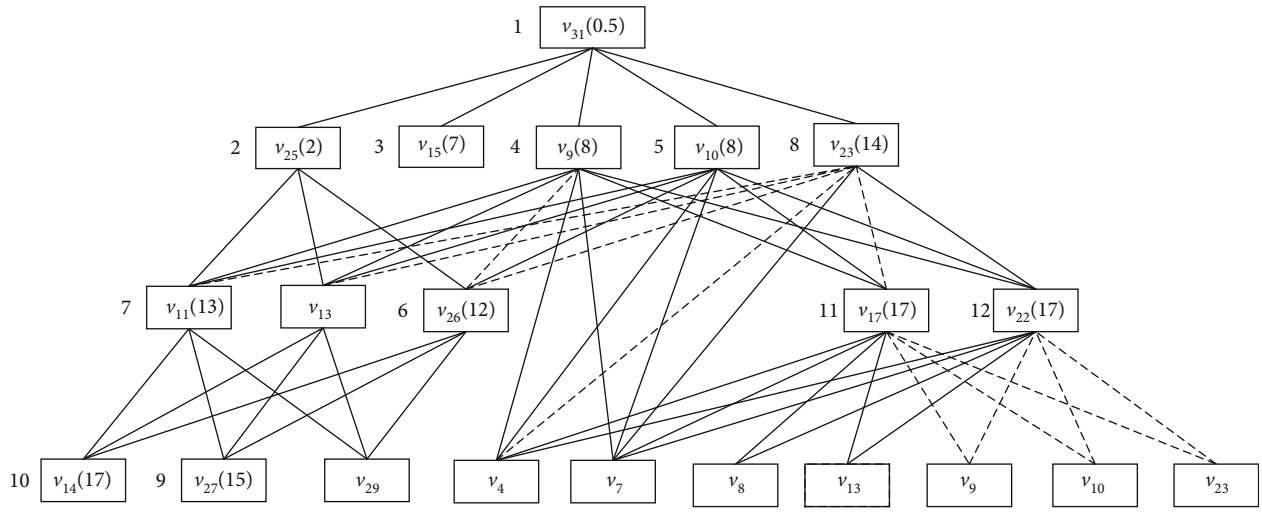


Figure 11: The expansion process of query $(q, 5)$.

takes the minimum value; that is, v_i makes $\mathrm{Dist}(v_i, B(G_j))$ take the minimum value. Since v_i and $v_i{}'$ are both in the same subgraph G_i, the uniqueness of the node shows that the assumption that there is node $v_i{}'$ in the G_i is wrong; that is, the value of $\mathrm{Dist}((v_i, v_j) \mid v_k)$ passes through the shortest distance of node v_k. Furthermore, to reduce the redundant calculations in node expansion, the minimum node does not need to be expanded repeatedly. □

Figure 11 shows the query process for $\mathrm{Query}(q, 5)$. The number, which lies on the left of the box, means the query processing steps. The solid lines represent the q expansion process using the distance interval method, and the dotted lines represent the calculation amount using the expansion method compared to the standard calculation method, which is not using the expansion method. There are 40 solid lines and 12 dotted lines; the calculation efficiency of the expansion method is increased by $12/(40 + 12) = 23\%$.

5.4. Tree-Based k-NN Search Algorithm

Definition 23 (adjacent subgraph). Take two subgraphs G_i and G_j, and $G_i \neq G_j$, $B(G_i) = \{v_i\}$, and $B(G_j) = \{v_j\}$. G_i and G_j have a common ancestor, i.e., $G_i{}^p = G_j{}^p$. We call G_i and G_j adjacent subgraphs. The adjacent subgraph of border node v_i is G_j and is denoted by $\mathrm{AdSub}(v_i) = \{G_j\}$.

For example, in Figure 6, v_{23} is the border node of G_6, and $v_{23} \in G_6$, $G_6{}^p = G_2$, and $G_5{}^p = G_2$. Therefore, G_5 and G_6 are adjacent subgraphs, and the adjacent subgraph of v_{23} is G_5. Furthermore, because $v_{23} \in G_2$, and $G_2{}^p = G_1{}^p = G_0$, G_1 is the v_{23} adjacency subgraph; that is, the adjacency subgraph of v_{23} is G_1. This is then denoted by $\mathrm{AdSub}(v_{23}) = \{G_5, G_1\}$.

Definition 24 (road network k-NN query). Given a directed weighted graph $G(V, E, W)$, M is the moving object set, and $M = \{m_1, m_2, m_3, \cdots, m_k\}$. The result of the query set is R. In the road network environment, the k-NN query at time t is expressed as $k\text{-NN} = (q, k, t)$, where k is a natural number and q is the query point. These variables are expressed as follows:

$$\begin{aligned} &|R| = k, \\ &R \subset M, \\ &\forall x \in R, \forall y \in M - R, \mathrm{SPDist}(q, x) \leq \mathrm{SPDist}(q, y). \end{aligned} \tag{9}$$

When giving a query point q, we query the k-nearest moving objects to q. First, we determine the subgraph G_i in which q is located. The subgraph contains the number of moving objects M_N and the starting node v_i of the edge. According to the node distance set $\mathrm{DisSet}(D_i)$, we then find the distance between v_i and the end node v_e where each moving object is located. Next, we sort the items in heap H from small to large and process the first element of the heap.

For example, when given the query point q, $k = 3$ means that q is in the subgraph G_6, and G_6 contains 5 moving objects. The starting node of its edge is v_{31}, and we need to find the 3 moving objects closest to v_{31}.

Here, we introduce the calculation for the k-NN for moving objects on the road network. In the existing k-NN algorithm for moving objects on the road network, there are redundant calculations. For this reason, we propose the DCDP-based moving object k-NN (DMOK) algorithm.

In the road network environment, for a given query point $q(v_s, q.\mathrm{offset})$, the k moving objects closest to the query point q are found, namely, $\mathrm{Query}(q, k)$. Figure 12 shows the processing of $\mathrm{Query}(q, 4)$. The specific calculation process is as follows:

(1) Determine the subgraph G_i of the starting node v_s of q

(2) Calculate the distance between the border node and the active node to v_s in G_i, and add the distance to H_{Ac}. Calculate the distance from the inactive node in G_i to v_s, and add the distance to H_{InAc}

Expansion of this node

Processing node	The border point and the vertex of the moving object	Expansion of this node	H_{Ac}	H_{InAc}	R
$v_{31} \in G_6$		$SPDist(v_9, v_{31}) = 8$ $SPDist(v_{10}, v_{31}) = 8$ $SPDist(v_{15}, v_{31}) = 7$ $SPDist(v_{25}, v_{31}) = 2$ $SPDist(v_{23}, v_{31}) = 14$ $SPDist(v_{24}, v_{31}) = 6$ $SPDist(v_{32}, v_{31}) = 12$	$\boxed{v_{25}(2) \in G_6}$ $v_{15}(7) \in G_6$ $v_9(8) \in G_2$ $v_{10}(8) \in G_2$ $v_{23}(14) \in G_2$	$v_{24}(6) \in G_6$ $v_{32}(12) \in G_2$	
$v_{25}(2) \in G_6$	$G_5(v_{11}, v_{13}, v_{26})$ $V_6(v_{25})$	$Dist((v_{11}, v_{31})\|v_{25}) = [7,13]$ $Dist((v_{13}, v_{31})\|v_{25}) = [13,19]$ $Dist((v_{26}, v_{31})\|v_{25}) = 12$	$\boxed{v_{15} \to v_{31}(7)}$ $v_{11} \to v_{31}[7, 13]$ …… $v_{23} \to v_{31}(14)$	$v_{24}(6) \in G_6$ $v_{32}(12) \in G_2$	$m_{25}(5)$
$v_{15}(7) \in V_6$	$V_6(v_{15})$		$\boxed{v_9 \to v_{31}(8)}$ $v_{10} \to v_{31}(8)$ …… $v_{13} \to v_{31}[18, 19]$	$v_{24}(6) \in G_6$ $v_{32}(12) \in G_2$	$m_{25}(5)$, $m_{21}(7.3)$
……	……	……	……	……	……
$v_{26}(12) \in G_5$	$V_5(v_{14}, v_{29})$ $N_5(v_{12}, v_{27}, v_{28})$	$Dist((v_{14}, v_{31})\|v_{26}) = [14,19]$ $Dist((v_{29}, v_{31})\|v_{26}) = [16,21]$ …… $Dist((v_{28}, v_{31})\|v_{26}) = 18$	$\boxed{v_{11} \to v_{31}(13)}$ $v_4 \to v_{31}[14, 20]$ …… $v_{13} \to v_{31}(19)$	$v_{24} \to v_{31}(6)$ $v_{32} \to v_{31}(12)$ …… $v_{28} \to v_{31}(18)$	$m_{25}(5)$, $m_{21}(7.3)$, $m_3(12)$, $m_{20}(13)$
$v_{11}(13) \in G_5$	$V_5(v_{11}, v_{14}, v_{29})$ $N_5(v_{12})$	$Dist((v_{14}, v_{31})\|v_{11}) = [15,17]$ $Dist((v_{29}, v_{31})\|v_{11}) = [17,19]$ $Dist((v_{12}, v_{31})\|v_{11}) = [17,23]$	$\boxed{v_{23} \to v_{31}(14)}$ $v_4 \to v_{31}[14, 20]$ …… $v_{13} \to v_{31}(19)$	$v_{24} \to v_{31}(6)$ $v_{32} \to v_{31}(12)$ …… $v_{28} \to v_{31}(18)$	$m_{25}(5)$, $m_{21}(7.3)$, $m_3(12)$, $m_{20}(13)$, $m_{19}(14.5)$
$v_{23}(14) \in G_2$	$SPDist(v_{14}, v_{31}) = 14 \geq \text{Dth} = SPDist\ (m_{20}, v_{31}) = 13$				$m_{25}(5)$, $m_{21}(7.3)$, $m_3(12)$, $m_{20}(13)$

Figure 12: Processing of Query $(q, 4)$.

(3) Expand according to the expansion strategy. According to the first element V_{cur} of heap H_{Ac}, if V_{cur} is the border node, calculate the distance between the border node of the adjacent subgraph and v_s, and add the node to H_{Ac} (whose elements are sorted from smallest to largest). If V_{cur} is the active node, calculate the distance between the moving object m and v_s, and add m to the moving object candidate set Cset

(4) According to the order of heap H_{Ac}, the first element V_{cur} of the heap is processed. When the number of moving object candidate set elements Cset.size $\geq k$, take the distance between the kth m and v_s in the moving object candidate set as the moving object distance threshold Dth, update Dth. When Dist $(V_{cur}, v_s) \geq$ Dth, the processing of heap H_{Ac} ends

(5) Output the result set R of moving objects

According to the node expansion strategy, and combined with the updating model, the set of moving objects that meet the query criteria can be quickly determined. Algorithm 3 shows the specific process.

For example, the process of solving query$(q, 4)$ according to Algorithm 3 is as follows:

(1) Query point is $q(V_S, q.\text{offset})$, $V_s = v_{31}$, and the offset from the starting node v_{31} is $q.\text{offset} = 0.5$

(2) $q \in G_6$, and according to the moving object table, the number of moving objects in the subgraph G_6 is 3 because the number of the moving objects $M_N = 3 < k = 4$; it needs to further expand to adjacent subgraphs. Determine the distance from the end node V_e of the moving object in the subgraph G_6 to the start node v_{31} of the query point q, calculate the distance from the border node of G_6 to v_{31}, and add nodes to H_{Ac}. In the same way, add the node from the inactive node in the subgraph G_6 to the starting node v_{31} of the query point q and add them to the heap H_{InAc}

(3) Expand according to the node expansion strategy. When the distance between the same two nodes is an interval, the lower bound of the distance interval takes a larger value, and the upper bound takes a smaller value. At the same time, the nodes in the

Input: $q(v_s, q.offset)$
Output: moving object result set R

Determine the starting node v_s of q and its subgraph;
$V_{cur} = \varnothing$; $Dth = 0$; $C\,set = \varnothing$;
Calculate the distance between v_s and its border nodes and active nodes in the subgraph, and add these distances to the H_{Ac} i order from small to large, calculate the distance between v_s and the inactive nodes in the subgraph, and add these distances to the heap of H_{InAc} in order from small to large;
while H_{Ac} *is not empty* **do**
  Take the first element V_{cur} of H_{Ac};
  if V_{cur} *is the border point* **then**
    if V_{cur} and v_s *are not in the same subgraph* **then**
      Calculate the distance between the active node of the subgraph where V_{cur} is located and v_s, and add it to the heap H_{Ac}
    if $Cset.size < k$ **then**
      Calculate the distance between the border node of the V_{cur} adjacency subgraph and v_s, and add it to the heap H_{Ac}
    else
      According to the expansion strategy, the nodes that meet the conditions are added to the H_{Ac}
  else if V_{cur} is the active node **then**
    Obtain the set M of moving objects corresponding to V_{cur} through the Moving Object List and sort them in descending order of the distance from V_{cur};
    for *Take each moving object* $m \in M$ *and process if* **do**
      Calculate **Dist**(m, v_s);
      if $Cset, size < k$ **then**
        Continue to Line 10;
      else
        if *Dth value is empty* **then**
          Determine the initial threshold Dth;
        else
          if Dist$(m, v_s) < Dth$ **then**
            Use m and **Dist**(m, v_s) to update $Cset$ and threshold Dth respectively;
          else
            The algorithm ends;
Output $R = Cset$

Algorithm 3: DMOK algorithm().

heap H_{Ac} follow the upper bound of the distance interval and are sorted in ascending order

(4) When the candidate set of moving objects Cset.size = 4, use the maximum value in Cset at this time as the distance threshold Dth = 13

(5) When the first element V_{cur} of heap H_{Ac}, the distance from v_{11} to v_{31}, $\mathrm{Dist}(v_{11}, v_{31}) = 13$ is less than or equal to Dth = 13, continue to expand the node for the first element V_{cur} of the heap H_{Ac}, and update the distance threshold Dth. When the distance from the first element in H_{Ac} to v_{31} $\mathrm{Dist}(v_{11}, v_{31}) \geq$ Dth, the node expansion ends

(6) Output the result set of moving objects $R = \{m_{25}, m_{21}, m_{22}, m_4\}$, and terminate the algorithm

6. Results and Discussion

The experimental environment is configured as a PC with Intel Core i7-6700 CPU @ 3.40 GHz, with 16 GB memory, 2 T hard disk, and Windows 10 operating system. The algorithm is written in *C++* language.

The data set used in this experiment is a real-world road network from an urban road network [39]. In order to verify the effectiveness of the algorithm proposed in this paper, we selected four data sets of them for experimental comparison. The detailed information is shown in Table 8.

For V-tree and $G*$-tree ($G*$-tree means the algorithm in the G-tree paper [25]), we set $f = 4$ and $\tau = 32$. We evaluated the k-NN search efficiency of DMOK and used TEN-Index, V-tree, and G^*-tree as baselines. We vary k as 10, 20, 50, and 100. We present the search time of the four

Table 8: Road network data statistics.

Data set	$\|V\|$	$\|E\|$
Pune (PN)	28649	36925
Baghdad (BD)	60108	88876
Tehran (TR)	110580	147339
Bangkok (BK)	154352	187798

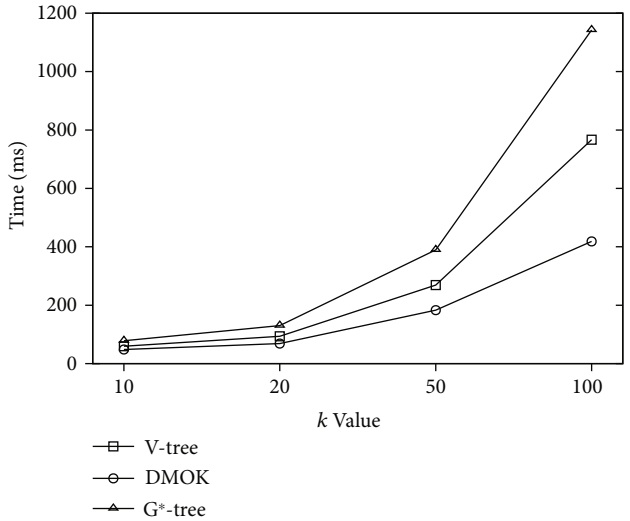


Figure 13: Comparison of various algorithms under different k values.

algorithms on the BK data set. The abscissa is the variation range of k.

As can be seen from Figure 13, for different values of k, the search time of our algorithms is the least compared with the other two algorithms; this is due to the node processing strategy proposed in this algorithm. Because the TEN-Index can only query k-NN within the preset k value of the index, the query cannot be processed when the k value of k-NN is greater than the preset k value; the query is not compared to it in this set of experiments.

In Figures 14 and 15, we compared the DTH index construction time and space consumption with V-tree, TEN-Index, and G^*-tree on the four data sets. Object density θ = candidate nodes/total number of vertices. In the experiment, $k = 10$, and $\theta = 0.3$. We randomly generate 1000 vertices as query points and take the average total query time as the final query time. From the histogram, we can clearly see that building the DTH index proposed in the paper takes less time than other indices. Conversely, the building index times of V-tree and $G*$-tree are high; the main reason for this is that they set up an index of structural information such as nodes, edges, active nodes, and moving objects. Meanwhile, our algorithm constructs indices separately for road network information and moving objects. When moving objects are updated, we only need to update locally, which saves a lot of time. TEN-Index built the index the fastest because it uses H2H-Index for the shortest distance calculation of the road networks, unlike the other algorithms that use Dijkstra. The disadvantage of TEN-Index is that it depends on the

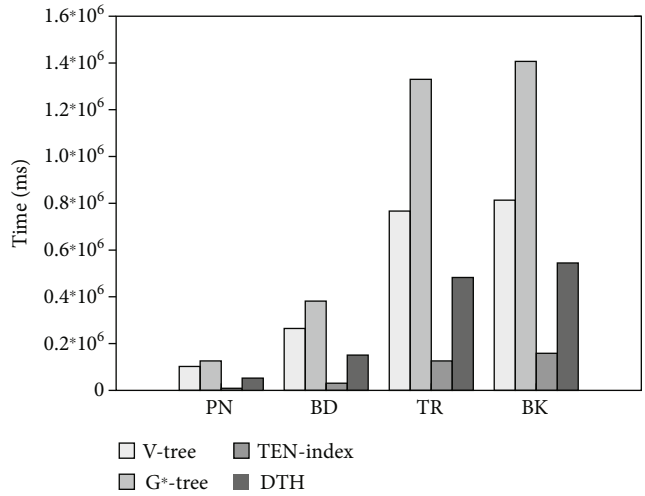


Figure 14: Comparison of index construction time.

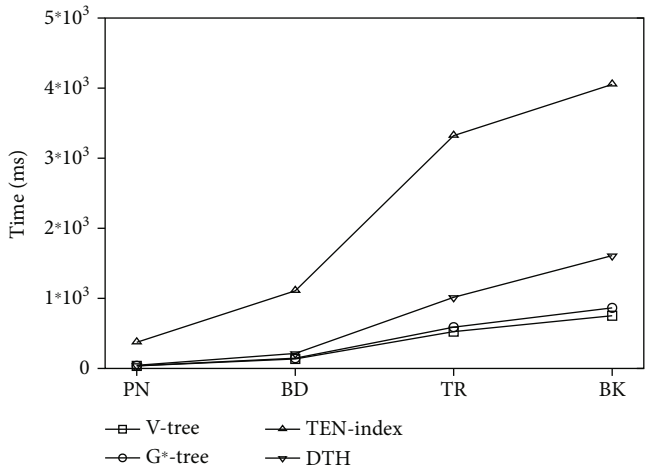


Figure 15: Index construction space.

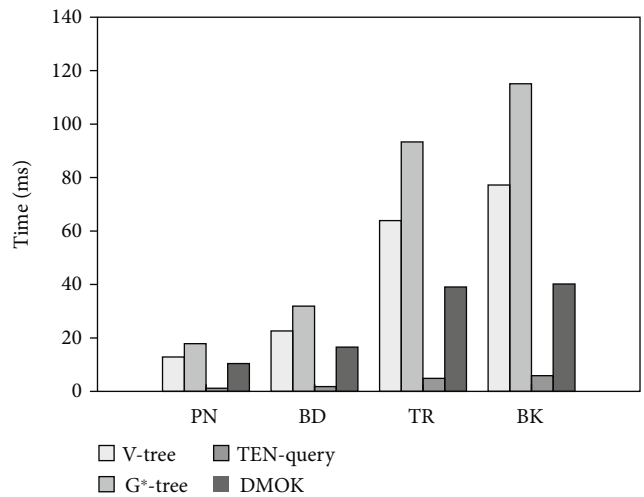


Figure 16: Comparison of search time.

value of k; that is, the value of k needs to be specified when building the index, and only queries within the value of k can be processed. Therefore, when the value of k is fixed, TEN-Index is relatively fast. In addition, its constructed index consumes much space and needs to be stored during tree decomposition. Considering a large amount of information, the index structure proposed in this article has certain advantages.

It can be seen from Figure 16 that the TEN-Index algorithm has weak adaptability to the value k, and the method proposed in this paper has advantages over the other algorithms in terms of search time. The reason for this is that in the initial stage of index construction, although a certain amount of space consumption is sacrificed, in the latter search, the proposed algorithm dramatically improves the search time due to its node strategy and avoidance of redundant calculations.

7. Conclusions

This paper studies a fast tree-indexed k-NN search algorithm of moving objects on a road network. Firstly, a novel index structure, called the double tree-hash index, is designed to reorganize the relationships between moving objects and the road network. The DTH index builds a weak bridge between tree and hash structures. The tree structure encapsulates the nodes, edges, and quantized relationships of intersubgraphs. The hash index records the moving objects. Through the weak bridge, the DTH index can realize activity-oriented updates of moving objects in the road network. Secondly, an index-enhanced k-NN search algorithm is proposed to quickly find the k-nearest neighbors of moving objects on the road network. This algorithm is called the DMOK algorithm. An updating model is established to quickly update the moving object and effectively expand the positive nodes. Through the conditional distance rules, the number of expanded negative nodes can be reduced. Then, multiple answers are provided to queries based on the DTH index. Finally, experiments with real data sets and artificial synthetic data sets verify the algorithm's effectiveness.

Data Availability

The data used to support the findings of this study have been deposited in the Road Networks, Urban (2016): Urban Road Network Data. figshare. Dataset. https://doi.org/10.6084/m9.figshare.2061897.v1.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61602076, Grant 61702072, Grant 62002039, and Grant 61976032; in part by the China Postdoctoral Science Foundation funded projects under Grant 2017M611211, Grant 2017M621122, and Grant 2019M661077; in part by the Natural Science Foundation of Liaoning Province under Grant 20180540003; and in part by the CERNET Innovation Project under Grant NGII20190902.

References

[1] N. Saeed, H. Nam, T. Y. al-Naffouri, and M. S. Alouini, "A state-of-the-art survey on multidimensional scaling-based localization techniques," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3565–3583, 2019.

[2] J. Q. Xu, R. H. Güting, Y. Zheng, and O. Ouri Wolfson, "Moving Objects with Transportation Modes: A Survey," *Journal of Computer Science and Technology*, vol. 34, no. 4, pp. 709–726, 2019.

[3] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Transactions on Database Systems*, vol. 24, no. 2, pp. 265–318, 1999.

[4] M. Sharifzadeh and C. Shahabi, "Vor-tree: R-trees with Voronoi diagrams for efficient processing of spatial nearest neighbor queries," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1231–1242, 2010.

[5] S. Ji and C. Li, "Location-based instant search," in *Scientific and Statistical Database Management. SSDBM 2011*, J. Bayard Cushing, J. French, and S. Bowers, Eds., vol. 6809 of Lecture Notes in Computer Science, pp. 17–36, Springer, Berlin, Heidelberg, 2011.

[6] K. C. K. Lee, W. C. Lee, and B. Zheng, "Fast object search on road networks," in *Proceedings of the 12th International Conference on Extending Database Technology Advances in Database Technology - EDBT '09*, pp. 1018–1029, New York, NY, USA, 2009.

[7] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Efficient query processing on spatial networks," in *Proceedings of the 2005 international workshop on Geographic information systems - GIS '05*, pp. 200–209, New York, NY, USA, 2005.

[8] S. Rogers, P. Langley, and C. Wilson, "Mining GPS data to augment road models," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '99*, pp. 104–113, San Diego, CA, USA, 1999.

[9] S. Berchtold, B. Ertl, D. A. Keim et al., "Fast nearest neighbor search in high-dimensional space," in *Proceedings 14th International Conference on Data Engineering*, pp. 209–218, Orlando, FL, USA, 1998.

[10] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, pp. 43–54, New York, NY, USA, 2008.

[11] H. J. Cho and R. Jin, "Efficient processing of moving k-range nearest neighbor queries in directed and dynamic spatial networks," *Mobile Information Systems*, vol. 2016, Article ID 2406142, 17 pages, 2016.

[12] H. J. Cho, R. Jin, and T. S. Chung, "A collaborative approach to moving-nearest neighbor queries in directed and dynamic road networks," *Pervasive and Mobile Computing*, vol. 17, Part A, pp. 139–156, 2015.

[13] G. Li, P. Fan, Y. Li, and J. Du, "An efficient technique for continuous k-nearest neighbor query processing on moving

objects in a road network," in *2010 10th IEEE International Conference on Computer and Information Technology*, pp. 627–634, Bradford, UK, 2010.

[14] Z. Yu, Y. Liu, X. Yu, and K. Q. Pu, "Scalable distributed processing of k nearest neighbor queries over moving objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 5, pp. 1383–1396, 2015.

[15] L. Heendaliya, M. Wisely, D. Lin, S. S. Sarvestani, and A. Hurson, "Indexing and querying techniques for moving objects in both euclidean space and road network[M]," in *Advances in Computers*, vol. 102, pp. 111–170, Springer, 2016.

[16] D. He, S. Wang, X. Zhou, and R. Cheng, "An efficient framework for correctness-aware kNN queries on road networks," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 1298–1309, Macao, China, 2019.

[17] B. Shen, Y. Zhao, G. Li et al., "V-tree: efficient knn search on moving objects with road-network constraints," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, San Diego, CA, USA, 2017.

[18] D. Ouyang, D. Wen, L. Qin, L. Chang, Y. Zhang, and X. Lin, "Progressive top-K nearest neighbors search in large road networks," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 1781–1795, New York, NY, USA, 2020.

[19] M. B. Wells and E. Donald, "Book review: the art of computer programming, volume 1. Fundamental algorithms and volume 2. Seminumerical algorithms," *Bulletin of the American Mathematical Society*, vol. 79, no. 3, pp. 501–510, 1973.

[20] A. Mahmood and W. G. Aref, "Query processing techniques for big spatial-keyword data," in *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 1777–1782, New York, NY, USA, 2017.

[21] M. Kolahdouzan and C. Shahabi, "Voronoi-based K nearest search in spatial network databases," in *Proceedings of the 30th International Conference on Very Large DataBases*, pp. 840–851, Canada, Toronto, 2004.

[22] P. Zhang, H. Lin, Y. Gao, and D. Lu, "Aggregate keyword nearest neighbor queries on road networks," *GeoInformatica*, vol. 22, no. 2, pp. 237–268, 2018.

[23] X. Huang, C. S. Jensen, H. Lu, and S. Šaltenis, "S-GRID: a versatile approach to efficient query processing in spatial networks," in *Advances in Spatial and Temporal Databases. SSTD 2007*, D. Papadias, D. Zhang, and G. Kollios, Eds., vol. 4605 of Lecture Notes in Computer Science, pp. 93–111, Springer, Berlin, Heidelberg, 2007.

[24] K. C. K. Lee, W. C. Lee, B. Zheng, and Y. Tian, "ROAD: a new spatial object search framework for road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 3, pp. 547–560, 2012.

[25] 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.

[26] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *Proceedings of the 1995 ACM SIGMOD international conference on Management of data - SIGMOD '95*, pp. 71–79, New York, NY, USA, 1995.

[27] Y. Liu, G. Liu, and Z. He, "Spatial index technology for multiscale and large scale spatial data," in *2010 18th International Conference on Geoinformatics*, pp. 1–4, Beijing, China, 2010.

[28] E. Frentzos, "Indexing objects moving on fixed networks," in *Advances in Spatial and Temporal Databases. SSTD 2003*, T. Hadzilacos, Y. Manolopoulos, J. Roddick, and Y. Theodoridis, Eds., vol. 2750 of Lecture Notes in Computer Science, pp. 289–305, Springer, Berlin, Heidelberg, 2003.

[29] V. T. de Almeida and R. H. Güting, "Indexing the trajectories of moving objects in networks," *GeoInformatica*, vol. 9, no. 1, pp. 33–60, 2005.

[30] N. Du, J. Zhan, M. Zhao, D. Xiao, and Y. Xie, "Spatio-temporal data index model of moving objects on fixed networks using hbase," in *2015 IEEE International Conference on Computational Intelligence & Communication Technology*, pp. 247–251, Ghaziabad, India, 2015.

[31] L. Liu, W. Li, Y. Guo, and J. Le, "Supporting high updates disk-based index in road network," in *2008 IEEE International Conference on e-Business Engineering*, pp. 517–522, Xi'an, China, 2008.

[32] Y. Theodoridis, M. Vazirgiannis, and T. Sellis, "Spatio-temporal indexing for large multimedia applications," in *Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems*, pp. 441–448, Hiroshima, Japan, 1996.

[33] I. Kamel and C. Faloutsos, *Hilbert R-Tree: An Improved R-Tree Using Fractals*, Institute for Systems Research Technical Reports, DRUM, 1993.

[34] S. T. Leutenegger, M. A. Lopez, and J. Edgington, "STR: a simple and efficient algorithm for R-tree packing," in *Proceedings 13th International Conference on Data Engineering*, pp. 497–506, Birmingham, UK, 1997.

[35] S. Luo, B. Kao, G. Li, J. Hu, R. Cheng, and Y. Zheng, "TOAIN: a throughput optimizing adaptive index for answering dynamic k-nn queries on road networks," *Proceedings of the VLDB Endowment*, vol. 11, no. 5, pp. 594–606, 2018.

[36] D. Tianyang, Y. Lulu, C. Qiang, C. Bin, and F. Jing, "Direction-aware KNN queries for moving objects in a road network," *World Wide Web*, vol. 22, no. 4, pp. 1765–1797, 2019.

[37] R. H. Güting, V. T. De Almeida, and Z. Ding, "Modeling and querying moving objects in networks," *The VLDB Journal*, vol. 15, no. 2, pp. 165–190, 2006.

[38] "V-tree: efficient knn search on moving objects with road-network constraintsTsinghua Technical Reporthttp://dbgroup.cs.tsinghua.edu.cn/ligl/v-tree.pdf.

[39] https://figshare.com/articles/dataset/Urban_Road_Network_Data/2061897/1.