



Efficient methods for finding an optimal network location for travel planning

Junkyu Lee¹ · Seog Park¹

Accepted: 25 March 2021 / Published online: 9 April 2021
© Author(s) 2021, corrected publication 2021

Abstract

In this paper, we describe an optimal network location for travel planning (ONLTP) query, a type of optimal location query. In trip planning, finding the optimal point for a group of users is a fundamental problem in spatial group query processing. Many previous studies have considered the problem of finding the optimal point. However, their queries using an exact method perform efficiently only when the users are closely distributed, not spread out in large road networks. In contrast, approximation methods use two different spatial indices, but they cannot control the trade-off between query performance and accuracy. We propose a method using G-trees [1, 2] to remedy these drawbacks. Our exact method is a concrete implementation of the best-first search in G-trees, and our approximation method further reduces the visited nodes of the exact method.

Keywords Spatial databases · Trip planning query · G-tree · Optimal location query · Optimal meeting point

1 Introduction

A search for the optimal location is used in various applications. For example, when people select a dining location, they can choose the optimal meeting point, considering the distance from their home locations, and a travel agent can choose where to have people picked up for transportation to the airport with the minimum number of miles. This type of query, called optimal meeting point (OMP) [3], finds the optimal location in the entire spatial network using the objective function for a given query. The objective function for the optimal network location for travel planning (ONLTP)

✉ Seog Park
spark@sogang.ac.kr

Junkyu Lee
ljk7776@sogang.ac.kr

¹ Database Laboratory, Department of Computer Science and Engineering, Sogang University, Seoul, Korea

consists of two functions: **MINSUM** minimizes the total travel distance for all people, and **MINMAX** minimizes the longest travel distance. The definitions of **MINSUM** and **MINMAX** are as follows:

- **MINSUM**: minimize $\sum dist(q_i, m) + dist(m, d)$, and
- **MINMAX**: minimize $\max dist(q_i, m) + dist(m, d)$,

where m is an arbitrary meeting point, q_i is the i th member of the set of query points $\{q_1, q_2, \dots, q_{|Q|}\}$, d is the destination, and $dist$ is the shortest distance between m and d . For an illustration of these formulas, see Fig. 1, an example of a road network. Given a set of query points $Q = \{v_2, v_5, v_6\}$ and a destination $d = v_8$, the **ONLTP** function **MINSUM** returns v_3 , with a cost of 11.

$$\text{MINSUM} = dist(v_2, v_3) + dist(v_5, v_3) + dist(v_6, v_3) + dist(v_3, v_8) = 16. \quad (1)$$

However, the **ONLTP** function **MINMAX** returns v_8 , and the cost is 9.

$$\text{MINMAX} = \max[dist(v_2, v_8), dist(v_5, v_8), dist(v_6, v_8)] + dist(v_8, v_8) = 9. \quad (2)$$

To solve the **ONLTP** with a naive approach, we should first compute the distance from each query point to each vertex. Then, we must find the optimal meeting point that minimizes the given objective function. However, this has a high computational cost. The main challenges of the **ONLTP** are, first, reducing the number of candidate meeting points to the extent possible and, second, computing the nontrivial shortest distance in the given road network. In this paper, we propose a branch-and-bound method based on the G-tree index, a scalable index of road networks, and a dynamic-programming approach that arises when applying the branch-and-bound method to solve the **ONLTP**. Furthermore, we propose a greedy-based approximation method to speed up query performance while maintaining high accuracy.

Our contributions are summarized as follows:

- We propose a concrete implementation of the branch-and-bound methodology of **ONLTP** using the G-tree index structure.
- We propose a greedy-based approximation method, which can control the trade-off between query performance and accuracy.

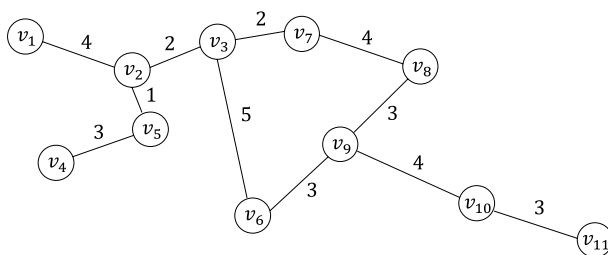


Fig. 1 Example of a road network

- We experimentally validated the efficiency of our method using road networks of various sizes, particularly with large datasets.

The remainder of the paper is as follows. In Sect. 2, we present several studies relevant to our work. In Sect. 3, we introduce a traditional branch-and-bound method and apply it to a G-tree index. In Sect. 4, we propose a dynamic programming method to solve the branch-and-bound problem in a G-tree that suffers from redundant computation. In Sect. 5, we propose a greedy-based approximation method to improve the query performance while maintaining a high degree of accuracy. In Sect. 6, we experimentally evaluate the query performance of our proposed method compared to existing related methods. In Sect. 7, we summarize our work.

2 Related work

Because we are studying an efficient processing method for the ONLTP, in this section, we consider studies related to the optimal meeting point or relevant to it.

Group nearest neighbor Papadias et al. [4] presented group nearest neighbor (GNN) queries, which find the point(s) in set P with the smallest sum of the distances to all locations in query set Q . They proposed three algorithms based on R-tree to solve a GNN query: the multiple query, single point, and minimum-bounding methods. These methods are used to minimize I/O and computational costs in memory-resident cases. Furthermore, they proposed two alternative methods based on the multiple query and the minimum-bounding methods to solve the problem when the query set does not fit in memory. However, their sum function only works in a Euclidean space.

Aggregate nearest neighbor Yiu et al. [5] presented aggregate nearest neighbor (ANN) queries, which consider network distances and several aggregate functions in GNN queries. Essentially, their work is different from GNN because of the nontrivial computation of the network distances. They proposed three algorithms—incremental Euclidean restriction, a threshold algorithm, and concurrent expansion—to minimize I/O costs during network traversal for nontrivial computation.

Collective travel planning Shang et al. [6] proposed collective travel planning (CTP) queries, which find the lowest-cost route that connects multiple query points and a destination through at most k meeting points (points of interest). Formally, given a set Q of query points, a set M of meeting points, a destination d , and an integer threshold k , the CTP query finds the subset A of M of maximum size k with the minimum cost. CTP consists of an exponential number of ANN [5] problems for distance calculation between the user and a meeting point. Shang et al. showed that the CTP problem is NP-hard, so they proposed a heuristic search strategy to process scenarios with a large $|Q|$.

However, all these studies considered only a scenario with a given set of points of interest. They are not directly comparable to our work because ONLTP considers the entire set of vertices in a road network as points of interest. Thus, ONLTP should be compared primarily to other studies more closely related to it.

Optimal meeting point Yan et al. [3] proposed an OMP query that returns the point in a road network with the minimum sum of network distances to all query points Q . An OMP query is different from a GNN query in that all vertices of a road network, rather than only points of interest, are candidates. They used two aggregate functions, *minsum* and *minmax*, to minimize the total travel distance and elapsed travel time. They proposed two algorithms: two-phase online convex-hull-based pruning and fast greedy. The two-phase pruning method works as follows: In the first phase, a convex hull is formed in Euclidean space using the query points. In the second phase, the sides of the convex hull transform into the shortest paths of the network. Only the vertices in the region surrounding the hull are checked for a correct answer, including the query points. The fast greedy method is more straightforward. Yan et al. used the convexity property of the sum of the network distance function and adopted the gradient descent method [7, 8] to find the optimal meeting point. First, the initial point is computed, along with the center of gravity of query points Q , and a vertex is obtained using the nearest neighbor query on KD-tree [9]. The process of searching for neighbor vertices is repeated until there are no better neighbor vertices.

The ONLTP query differs only in that one point of the query is used as an arrival point and is not otherwise fundamentally different from the OMP query. However, in the real world, the distance from the query point to the destination is usually greater than the distance between all query points. We propose a new technique performs better than the methods in the previous studies in both the aggregate function used in the existing OMP and the aggregate function following the travel planning case.

Other types of queries There are many other different types of queries for a road network environment. Zhong et al. [1, 2] proposed G-tree, which is a road network index structure that supports multiple types of road network queries, including shortest path, k -nearest neighbor (kNN), and keyword-based kNN queries. G-tree partitions a road network into multiple subgraphs, and it constructs a hierarchical tree structure of the subgraphs. To solve queries on a road network, Zhong et al. also presented an assembly-based method that traverses each G-tree node and computes sub-results in a dynamic programming approach. Li et al. [17] proposed G*-tree to address the inefficiency problem of G-tree. G*-tree is similar to G-tree, but it builds shortcuts between selected leaf nodes. Each shortcut stores the distances between the borders of two leaf nodes, so it efficiently supports shortest path, kNN, and range queries. Jung et al. [18] developed a personalized route planning algorithm. They proposed a variation of the pathfinding problem that assumes the user preference changes over time. To find the best path, they suggested MPG-tree, which stores the dynamic attribute list of each node. Chen et al. [19] presented a continuous path-based range query in which the query point moves along a given path. To answer the query, they reduced it to finding the set of points for which the answer set changes. To enhance the query performance, they developed a backbone network index structure. The method exploits G*-tree to store the backbone network, which is a simplified graph of the original road network and stores detailed network information by the region directory. The authors also developed a two-phase query processing algorithm based on the filter-and-verification framework. Zhao et al. [20] proposed a time-aware spatial keyword (TSK) query, which finds the most suitable

k objects satisfying spatial, temporal, and textual constraints. To efficiently address the TSK query, they developed the TG index, which consists of a TA file and G-tree. the TA file stores the time and keyword information about an object and maintains their aggregate values. To address the TSK query, they adopt a filter and refinement framework using the TG index. These studies considered spatial queries using a G-tree-like index structure, but all of the above methods fail to consider multiple users, which lead to inefficiency or even infeasibility compared to our query.

3 Preliminaries

3.1 Problem definitions

In this section, we formally define the ONLTP and several relevant definitions. First, we formalize the spatial database in which our problem is identified.

Definition 1 *Road network and dist* Given a vertex set V and an edge set $E(\subseteq V \times V)$ with edge costs, the edge costs are nonnegative, symmetrical, and satisfy the triangle inequality. The shortest distance between two vertices $u \in V$ and $v \in V$ is defined as the minimum accumulated cost value of a path between u and v and is denoted by $dist(u, v)$.

We call the shortest distance in a spatial database the *network distance*.

Definition 2 *Optimal Network Location for Travel Planning (ONLTP)* Given a road network $G(V, E)$, a set Q of query points, a destination d , and a cost function F (namely, MINSUM or MINMAX), the ONLTP finds the vertex $v \in V$ that is the optimal point based on the cost function F . The MINSUM and MINMAX functions are defined as follows:

- MINSUM: $\arg \min_{v \in V} \{ \sum_{q_i \in Q} dist(q_i, v) + dist(v, d) \};$
- MINMAX: $\arg \min_{v \in V} \{ \max_{q_i \in Q} dist(q_i, v) + dist(v, d) \}.$

To solve the ONLTP in a brute-force manner, we first compute the distance from each query point to each vertex. Next, we compute the distance from the destination to each vertex. Finally, we can determine the optimal meeting point using the Dijkstra approach of $O(|V| \log |V|)$ to calculate the shortest path distance. Thus, the naive method has $O(|Q||V| \log |V|)$ time complexity. This method is not feasible for query-processing systems because the computational cost is so high.

3.2 Branch-and-bound method

The branch-and-bound method is a structured search of the space of all feasible solutions. The space of all feasible solutions is repeatedly partitioned into smaller subspaces, and a lower bound is used as the value to represent each subspace. Yan

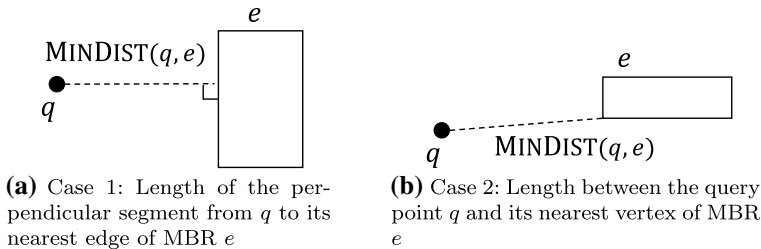


Fig. 2 MINDIST computations in R-trees

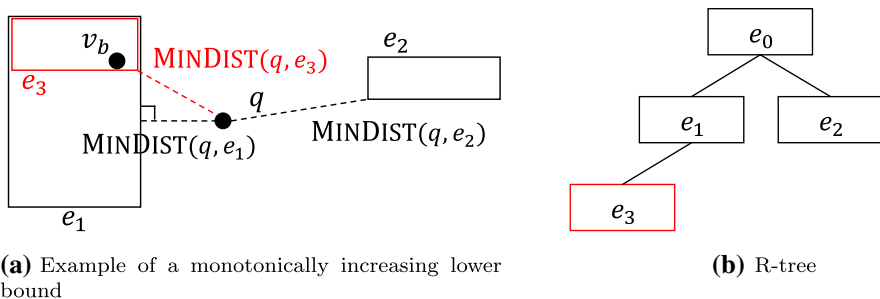


Fig. 3 Example of the branch-and-bound method in R-trees

et al. extended their work [3] to [11]. Specifically, with R-tree [10], they define the lower bound as $\sum_{q \in Q} \text{MINDIST}(q, e)$, where Q is a set of query points and e is the minimum bounding rectangle (MBR) in an R-tree. Figure 2 is an illustration of MINDIST computations in R-trees. There are three computations involved: First, the distance from q to the nearest edge of the MBR e , by a segment drawn perpendicularly to the nearest edge of e , as depicted in Fig. 2a; second, the distance between q and the nearest vertex of e , as depicted in Fig. 2b; and third, if the query point q is in the MBR e , then $\text{MINDIST}(q, e) = 0$. Since the sum of MINDIST is always less than or equal to the network distance, this value is used as the lower bound in R-trees. According to the borders property [1, 2], to reach a subgraph, we must visit one of its borders.

Figure 3 is an example of a branch-and-bound method. Given a set of query points $Q = \{q\}$, assume that the current search node is e_3 , the nearest vertex is q , v_b is inside e_3 , and the search order is depth-first (that is, $e_0 \rightarrow e_1 \rightarrow e_3 \rightarrow e_2$). The current best answer is found if the search process reaches the leaf node of the R-tree. Therefore, we can find the nearest vertex v_b while we visit e_3 . Then, the current best cost is $\text{dist}(q, v_b)$. Next, according to the depth-first search order, the next node to be searched is e_2 . We can end the search without traversing vertices inside e_2 if e_2 's lower bound $\text{MINDIST}(q, e_2)$ is greater than the current best answer, $\sum_{q \in Q} \text{dist}(q, v_b)$. To apply the branch-and-bound method, two properties must be satisfied: First, the lower bound of the parent node is less than or equal to the lower bound of its children, and second, the cost increases monotonically as the cardinality of Q increases.

3.3 G-tree index structure

A G-tree [1, 2] is a scalable index based on a multi-level graph partitioning method. Similarly to R-tree, G-tree also constructs a hierarchical tree structure. It partitions the road network into sub-networks and organizes them hierarchically. Each node of the G-tree maintains the vertices of the edges that connect them between subgraphs, called *borders*. Figure 4 is an example of a G-tree index structure. Fig. 4a is an example of a road network, and Fig. 4b is a G-tree index structure. In Fig. 4b, the numbers of the tree nodes are its subgraph borders. Each matrix of a non-leaf node has the pair-wise network distances between the borders of its child nodes. According to the border property, to reach a subgraph, we must visit one of its borders. Thus, using this property, we define GMINDIST, the shortest distance between a vertex and a subgraph of a G-tree.

Definition 3 *Minimum distance between a vertex and a subgraph* Given a vertex $v \in V$ and a G-tree node \mathcal{G}_n , the shortest distance between v and \mathcal{G}_n is denoted by $\text{GMINDIST}(v, \mathcal{G}_n)$ and computed as follows.

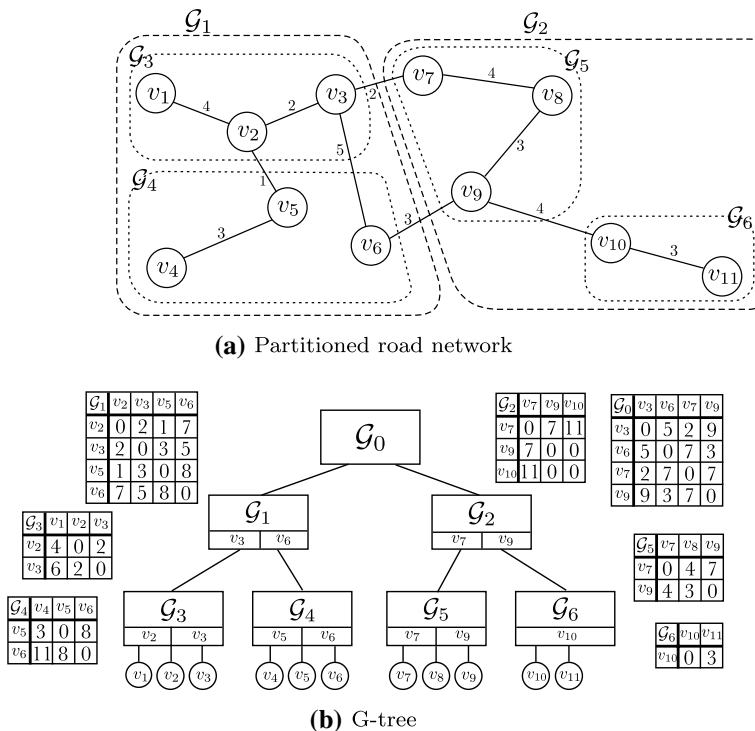


Fig. 4 Example of G-tree index structure

$$\text{GMINDIST}(v, \mathcal{G}_n) = \begin{cases} 0 & \text{if } v \text{ is in } \mathcal{G}_n \\ \min_{b_i \in \mathcal{B}(\mathcal{G}_n)} \text{dist}(v, b_i) & \text{otherwise} \end{cases}, \quad (3)$$

where $\mathcal{B}(\mathcal{G}_n)$ is the border set of the subgraph \mathcal{G}_n .

As we mentioned in Sect. 3.2, to apply the branch-and-bound method to a G-tree, we have to show that the G-tree follows two properties: a monotonically increasing GMINDIST and a monotonically increasing aggregate cost function.

4 Exact method

4.1 Best-first search

To apply a best-first search method, we need to define a lower bound in a G-tree and show the two properties of monotonicity.

Definition 4 *Lower bound on MinSum in a G-tree* Given a set Q of query points, a destination d , and a node \mathcal{G}_n of a G-tree, the lower bound of ONLTP in the G-tree is denoted $\text{LOWERBOUND}(Q, d, \mathcal{G}_n)$ and is defined as follows.

$$\text{LOWERBOUND}(Q, d, \mathcal{G}_n) = \sum_{q_i \in Q} \text{GMINDIST}(q_i, \mathcal{G}_n) + \text{GMINDIST}(d, \mathcal{G}_n) \quad (4)$$

Definition 5 *Lower bound on MINMAX in a G-tree* Given a set Q of query points, a destination d , and a node \mathcal{G}_n of a G-tree, the lower bound of ONLTP in the G-tree is denoted $\text{LOWERBOUND}(Q, d, \mathcal{G}_n)$ and is defined as follows.

$$\text{LOWERBOUND}(Q, d, \mathcal{G}_n) = \max_{q_i \in Q} \text{GMINDIST}(q_i, \mathcal{G}_n) + \text{GMINDIST}(d, \mathcal{G}_n) \quad (5)$$

Theorem 1 *Monotonous lower bound in a G-tree* In a G-tree, the LOWERBOUND increases monotonically when searching in top-down order.

Proof of Theorem 1 Because GMINDIST, the unit value of the LOWERBOUND, is defined in two cases, it can be seen that the LOWERBOUND increases monotonically if both cases increase monotonically.

1. If a query point q_i is not in \mathcal{G}_n , given a child node \mathcal{G}_{n+1} of node \mathcal{G}_n , we compute $\text{LOWERBOUND}(q_i, d, \mathcal{G}_{n+1})$ and $\text{LOWERBOUND}(q_i, d, \mathcal{G}_n)$ as follows.

$$\begin{aligned}
\text{GMINDIST}(q_i, \mathcal{G}_{n+1}) &= \min_{b_{n+1} \in \mathcal{B}(\mathcal{G}_{n+1})} \text{dist}(q_i, b_1) \\
&= \min_{u_x \in \mathcal{B}(\mathcal{G}_x)} (\text{dist}(q_i, u_x) \\
&\quad + \min_{u_{x-1} \in \mathcal{B}(\mathcal{G}_{x-1})} (\text{dist}(u_x, u_{x-1}) \\
&\quad + \dots + \min_{b_n \in \mathcal{B}(\mathcal{G}_n)} (\text{dist}(u_1, b_n) \\
&\quad + \min_{b_{n+1} \in \mathcal{B}(\mathcal{G}_{n+1})} \text{dist}(b_n, b_{n+1}))). \tag{6}
\end{aligned}$$

$$\begin{aligned}
\text{GMINDIST}(q_i, \mathcal{G}_n) &= \min_{b_n \in \mathcal{B}(\mathcal{G}_n)} \text{dist}(q_i, b_1) \\
&= \min_{u_x \in \mathcal{B}(\mathcal{G}_x)} (\text{dist}(q_i, u_x) \\
&\quad + \min_{u_{x-1} \in \mathcal{B}(\mathcal{G}_{x-1})} (\text{dist}(u_x, u_{x-1}) \\
&\quad + \dots + \min_{u_1 \in \mathcal{B}(\mathcal{G}_1)} (\text{dist}(u_2, u_1) \\
&\quad + \min_{b_n \in \mathcal{B}(\mathcal{G}_n)} \text{dist}(u_1, b_n))). \tag{7}
\end{aligned}$$

Now we assume that the GMINDIST does not increase monotonically, that is, $\text{GMINDIST}(q_i, \mathcal{G}_{n+1}) < \text{GMINDIST}(q_i, \mathcal{G}_n)$. Then, according to (6) and (7), we obtain the following equation:

$$\min_{b_n \in \mathcal{B}(\mathcal{G}_n)} (\text{dist}(u_1, b_n) + \min_{b_{n+1} \in \mathcal{B}(\mathcal{G}_{n+1})} \text{dist}(b_n, b_{n+1})) < \min_{b_n \in \mathcal{B}(\mathcal{G}_n)} \text{dist}(u_1, b_n). \tag{8}$$

Since it is clear that $\min_{b_{n+1} \in \mathcal{B}(\mathcal{G}_{n+1})} \text{dist}(b_n, b_{n+1}) \geq 0$, we obtain following equation:

$$\min_{b_n \in \mathcal{B}(\mathcal{G}_n)} \text{dist}(u_1, b_n) < \min_{b_n \in \mathcal{B}(\mathcal{G}_n)} \text{dist}(u_1, b_n). \tag{9}$$

However, this is a contradiction. Hence, GMINDIST increases monotonically in the first case.

2. If a query point q_i is in \mathcal{G}_n , $\text{GMINDIST}(q_i, \mathcal{G}_n)$ is 0. Because $\text{GMINDIST}(q_i, \mathcal{G}_{n+1})$ is a network distance, it is always greater than or equal to 0. Therefore, GMINDIST increases monotonically in the second case.

In this paper, we use only two cost functions, MINSUM and MINMAX, which are already widely known to increase monotonally. Therefore, we do not prove that the functions are monotonically increasing. Now, we can use the branch-and-bound framework with LOWERBOUND as defined in Definitions 4 and 5. Alg. 1 is an implementation of the branch-and-bound method based on a G-tree. The search order is the best-first search, which examines the smallest lower bound first, and then it traverses the G-tree nodes until the termination condition is satisfied. This is the process by which the branch-and-bound method is applied to the G-tree index structure. However, this method wastes computational resources when calculating lower bounds by not using a pruning process. Figure 5 illustrates a redundant computation

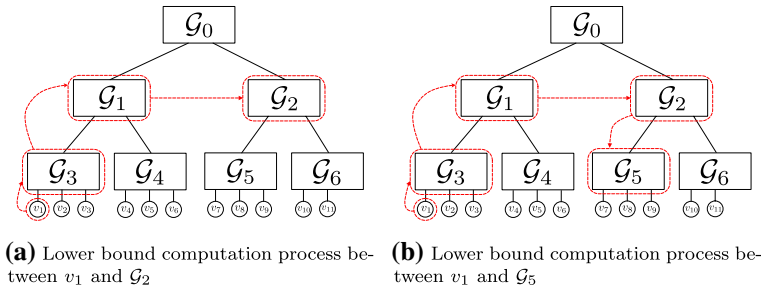


Fig. 5 Redundant computation of lower bound in G-tree

process of a lower bound. Figure 5b shows the computation process that occurs after Figure 5a. The top-down search always maintains this order, so the same subprocess is repeated when calculating the lower bound of each node.

Algorithm 1 Best-first search method based on G-tree

Input: a root node \mathcal{G}_0 of G-tree, a set Q of query points, and a destination d

Output: an optimal network location OPT

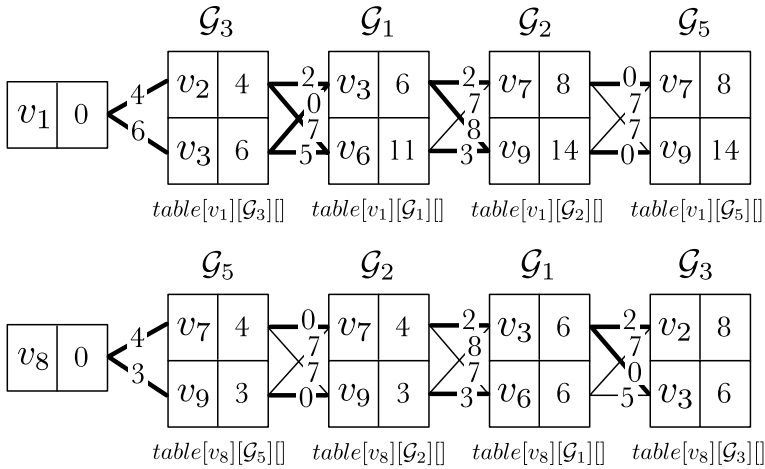
```

1: create an empty priority queue  $PQ$ 
2:  $l = \text{LOWERBOUND}(Q, d, \mathcal{G}_0)$ 
3:  $PQ.\text{Enqueue}(\langle l, \mathcal{G}_0 \rangle)$ 
4:  $OPT = \text{NULL}$ 
5:  $\text{current\_best} = \infty$ 
6: while  $PQ$  is not empty do
7:    $\langle l_p, \mathcal{G}_p \rangle = PQ.\text{ExtractMin}()$ 
8:   if  $l_p > \text{current\_best}$  then
9:     continue
10:  else if  $\mathcal{G}_p$  is nonleaf node then
11:    for  $\mathcal{G}_c$  in  $\mathcal{G}_p.\text{children}$  do
12:       $l_c = \text{LOWERBOUND}(Q, d, \mathcal{G}_c)$ 
13:      if  $l_c \leq \text{current\_best}$  then
14:         $PQ.\text{Enqueue}(\langle l_c, \mathcal{G}_c \rangle)$ 
15:  else
16:    for vertex  $v$  in  $\mathcal{G}_p.\text{vertices}$  do
17:       $\text{cost} = \sum_{q_i \in Q} \text{dist}(q_i, v) + \text{dist}(v, d)$ 
18:      if  $\text{current\_best} > \text{cost}$  then
19:         $\text{current\_best} = \text{cost}$ 
20:         $OPT = v$ 
21: return  $OPT$ 

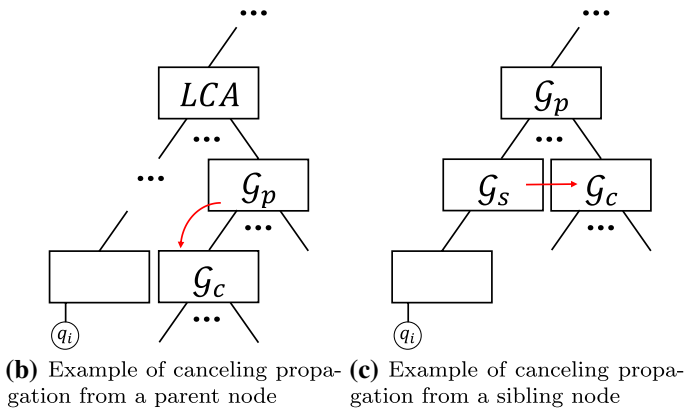
```

4.2 Redundancy cancellation method

To mitigate redundancy in the lower bound computation, it is necessary to store the partially computed distance matrix. Like the SPSP [1, 2] query in G-tree, we apply a method based on dynamic programming to cancel out the redundancy. We store the distance from a query point to a G-tree node border, as depicted in Fig. 6a. We calculate the distance from each query point to the borders of the G-tree node, store it



(a) Memory storage status based on dynamic programming



(b) Example of canceling propagation from a parent node (c) Example of canceling propagation from a sibling node

Fig. 6 Redundancy canceling based on dynamic programming

in memory, and retain it until the query is complete. Fig. 6b and c show examples of canceling propagation from parent and sibling nodes, respectively. If the distance from a parent to its child node has already been computed, we do not need to compute the distance from the child to its parent. Likewise, if the distance from a node and its sibling has already been computed, we do not need to compute the distance from the sibling to the original node. We simply propagate the results and compute only the additional distances needed. The following expresses Fig. 6 in equation form:

$$table[q_i][\mathcal{G}_j][b_k] = \begin{cases} \min table[q_i][\mathcal{G}_c][b_l] + \mathcal{G}_j.mat[b_l][b_k] & \text{if the child node } \mathcal{G}_c \text{ is already visited} \\ \min table[q_i][\mathcal{G}_s][b_l] + \mathcal{G}_p.mat[b_l][b_k] & \text{if the sibling node } \mathcal{G}_s \text{ is already visited} \\ \min table[q_i][\mathcal{G}_p][b_l] + \mathcal{G}_p.mat[b_l][b_k] & \text{if the parent node } \mathcal{G}_p \text{ is already visited} \end{cases}, \quad (10)$$

where *table* is the storage for redundancy canceling and $\mathcal{G}.mat$ is the distance matrix of the node \mathcal{G} . In Alg. 1, Line 17, the only additional calculation needed is from the borders in storage to the candidate meeting point, thereby improving the distance calculation time.

5 Approximation method

Even after the redundancy cancellation method is applied, nodes close to the root have a significant difference from the actual distance value that the lower bound can calculate on a leaf. Thus, these nodes are unlikely to be pruned. Applying the R-tree approximation methods proposed in existing studies [12–15] does not solve this problem. The following paragraphs are brief descriptions of the existing techniques.

ϵ -approximation In ϵ -approximation [12, 13, 15], given a non-negative real value ϵ ($\epsilon \geq 0$), if we visit a non-leaf (internal) node, its child node \mathcal{G}_c is inserted in the priority queue if $LOWERBOUND(Q, d, \mathcal{G}_c) \leq current_best / (1 + \epsilon)$.

α -allowance In α -allowance [14, 15], given a non-negative real value α ($0 \leq \alpha \leq 1$), if we visit a non-leaf (internal) node, its child node \mathcal{G}_c is inserted in the priority queue if $LOWERBOUND(Q, d, \mathcal{G}_c) \leq current_best * (1 - \alpha)$.

5.1 Lower bound approximation

To compensate for the problem of a node close to the root having a small lower bound, we propose an approximate lower bound estimation method. First, however, we look briefly at the drawback of comparing non-leaf nodes with the *current_best* value. In the top-down approach in G-trees, the child's lower bound is updated by selecting the minimum distance between the stored distance matrix in *table* and the distance between its border and its parent border. For both methods, ϵ -approximation and α -allowance, to work efficiently, the lower bound must be almost the same as previously, when the search was conducted downward to the child node. However, if the increment is 0, the child and parent nodes share the same border, which means a number of vertices equal to the depth of the border must be connected to different vertices or have a high probability of being connected to the vertices. In a real road network, the ratio of vertices to edges is close to one. Therefore, nodes close to the root are very unlikely to be pruned.

Fortunately, the increase in the lower bound has an easily determined maximum. Since both a border of the parent node and a border of the child node are in the

subgraph of the parent node, the distance between the two borders is necessarily less than or equal to the parent's subgraph's diameter, which is bounded as follows:

$$\max_{\mathcal{G}_c \in \mathcal{G}_p.\text{children}} \text{diameter}(\mathcal{G}_c) \leq \text{diameter}(\mathcal{G}_p), \quad (11)$$

where \mathcal{G}_p is a parent node and \mathcal{G}_c is a child node. Using this monotonicity, we can calculate an approximate lower bound, which is the expected value of the lower bound of a leaf node.

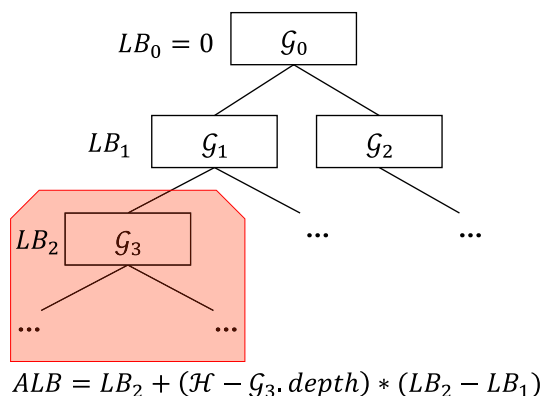
Definition 6 (Approximate lower bound in a G-tree) Given a set Q of query points, a destination d , and a node \mathcal{G}_n of a G-tree, the approximate lower bound of ONLTP in the G-tree is denoted $\text{ALBOUND}(Q, d, \mathcal{G}_n)$ and is defined as follows:

$$\begin{aligned} \text{ALBOUND}(Q, d, \mathcal{G}_n) = & \text{LOWERBOUND}(Q, d, \mathcal{G}_n) + (\mathcal{H} - \mathcal{G}_n.\text{depth}) \\ & * (\text{LOWERBOUND}(Q, d, \mathcal{G}_p) - \text{LOWERBOUND}(Q, d, \mathcal{G}_n)), \end{aligned} \quad (12)$$

where \mathcal{H} is the height of the G-tree and $\mathcal{G}_n.\text{depth}$ is the depth of node \mathcal{G}_n . Figure 7 is an example of the approximate lower bound. At the root node (depth=1) and nodes at depth 2, the approximate lower bound is not calculated. We calculate the approximate lower bounds on nodes with a depth of 3 or greater that are not leaves. This approximate lower bound value does not always follow the monotonicity of the diameter. It may be larger than the actual lower bounds of their leaf nodes. However, considering the ONLTP is an aggregate query, it can be predicted that each G-tree node follows the monotonicity with a high probability. This will be demonstrated as an approximation experiment in the experimental section.

Using the approximate lower bound, we have implemented a new approximation scheme. This new method combines ϵ -approximation and α -allowance and applies them to the branch-and-bound method using the approximate lower bound. In the approximation method, the initial cost is not set to infinity, but rather a vertex is selected and the pruning effect increases. We use the same method for selecting an initial point for each cost function, MINSUM and MINMAX, as the initial point for the gradient descent method in the Euclidean space proposed in [11]. Alg. 2 is an

Fig. 7 Example of approximate lower bound of \mathcal{G}_3 , $\text{ALBOUND}(Q, d, \mathcal{G}_3)$



implementation of the best-first search method with an approximate lower bound. We control the accuracy by adjusting the ϵ from -1 to 1 in Line 18. When ϵ is less than 0, it behaves similarly to α -allowance. When ϵ is greater than 0, it increases the accuracy and works in the opposite direction of ϵ -approximation. In Alg. 1, nodes with small lower bounds were searched first, but in Alg. 2, because *current_best* must be determined, the approximate lower bound can be used and the search order is from the nodes at the depth-first. If two nodes have the same smallest depth, then the node with the smaller lower bound is searched first.

Algorithm 2 Approximate best-first search method based on G-tree

Input: a root node \mathcal{G}_0 of G-tree, a set Q of query points, and a destination d

Output: an optimal network location OPT

```

1: create an empty priority queue  $PQ$ 
2:  $l = \text{LOWERBOUND}(Q, d, \mathcal{G}_0)$ 
3:  $PQ.\text{Enqueue}(\langle -1, l, \mathcal{G}_0 \rangle)$ 
4:  $OPT = \text{NULL}$ 
5:  $\text{current\_best} = \text{cost}$  for selecting initial vertex which proposed in [11]
6: while  $PQ$  is not empty do
7:    $\langle \text{negative\_depth}, l_p, \mathcal{G}_p \rangle = PQ.\text{ExtractMin}()$ 
8:   if  $l_p > \text{current\_best}$  then
9:     continue
10:  else if  $\mathcal{G}_p$  is nonleaf node then
11:    for  $\mathcal{G}_c$  in  $\mathcal{G}_p.\text{children}$  do
12:      if  $\mathcal{G}_c.\text{depth}$  (depth of  $\mathcal{G}_c$  in G-tree)  $\leq 2$  then
13:         $l_c = \text{LOWERBOUND}(Q, d, \mathcal{G}_c)$ 
14:        if  $l_c \leq \text{current\_best}$  then
15:           $PQ.\text{Enqueue}(\langle \text{negative\_depth} - 1, l_c, \mathcal{G}_c \rangle)$ 
16:      else
17:         $\text{approx}_c = l_c + (\mathcal{H} - \mathcal{G}_c.\text{depth}) * (l_c - l_p)$ 
18:        if  $\text{approx}_c \leq \text{current\_best} * (1 + \epsilon)$  then
19:           $PQ.\text{Enqueue}(\langle \text{negative\_depth} - 1, l_c, \mathcal{G}_c \rangle)$ 
20:    else
21:      for vertex  $v$  in  $\mathcal{G}_p.\text{vertices}$  do
22:         $\text{cost} = \sum_{q_i \in Q} \text{dist}(q_i, v) + \text{dist}(v, d)$ 
23:        if  $\text{current\_best} > \text{cost}$  then
24:           $\text{current\_best} = \text{cost}$ 
25:           $OPT = v$ 
26: return  $OPT$ 

```

6 Experiments

6.1 Experimental settings

We conducted experiments on real datasets of various sizes, with the results presented in Table 1. We used the gradient descent (GD) [3] for an approximate solution. We attempted to use the branch-and-bound (BB) [11] method as another comparison technique, but we had to exclude the BB from the experiments because the process did not end in time on a large graph. We used eight real-world datasets [16], which are presented in Table 1. The table lists the number of vertices and edges in

Table 1 Road network statistics

Graph	NY	BAY	COL	FLA	NW	CAL	LKS	W
# Vertices	264,346	321,270	435,666	1,070,376	1,207,945	1,890,815	2,758,119	6,262,104
# Edges	733,846	800,172	1,057,066	2,712,798	2,840,208	4,657,742	6,885,658	15,248,146

New York City (NY), the San Francisco Bay Area (BAY), Colorado (COL), Florida (FLA), Northwest USA (NW), California and Nevada (CAL), the Great Lakes region (LKS), and Western USA (W), all of which are available in [16]. All experiments were conducted on an Ubuntu 16.04 LTS platform with an Intel Xeon(R) E5-2620 v2 (2.10GHz) processor and 64 GB memory. The algorithms were implemented in C++ (GCC 5.4.0 with -O3 option). In the following subsections, we first investigate the performance and accuracy of ONLTP queries with uniform distributions. The G-tree [1, 2] parameters are fixed at $f = 4$, and the maximum leaf size τ is set to 128 for NY, BAY, and COL and to 256 for FLA, NW, CAL, LKS, and W, the same as were used in [17], where f is fanout, the number of children at a non-leaf node, and τ is the maximum number of vertices at a leaf node.

6.2 Evaluation of the query performance and the accuracy

In this subsection, only RCM (redundancy cancellation method) and ALBP (approximate lower bound-based pruning), which are the exact and approximation methods, respectively, proposed in our study, and the GD method are compared. The BB method was not used in this subsection because the query performed poorly when the query points were distributed over the entire road network. Hence, it is compared in other subsections, but not here.

Tables 2 and 3 show the query performance results and accuracy, respectively, of MINSUM with a query size of 100. The GD method had the best query performance but the worst accuracy. In [3], the authors claimed that the optimal meeting point query on the sum of the distance follows the convexity, so they proposed a gradient descent-based method. However, as shown in Table 3, the optimal meeting points do not follow the convexity in large graphs. Therefore, the GD method had a fast query time, but the results were far from the correct answers. On the other hand, our proposed exact method, RCM, produced query results within an acceptable time, unlike the BB method. Furthermore, our approximation method, ALBP, had nearly

Table 2 Query performance of MINSUM (ms)

Graph	NY	BAY	COL	FLA	NW	CAL	LKS	W
RCM	333.056	147.141	336.333	224.224	283.504	623.894	2053.33	832.695
ALBP ($\epsilon=0.1$)	167.592	62.0828	124.804	133.822	162.097	263.757	1296.61	306.857
ALBP ($\epsilon=0$)	131.522	26.4952	56.8279	74.987	136.459	130.574	1134.9	261.928
GD [3]	27.3299	8.07486	10.8545	13.8029	15.9969	28.286	126.262	81.6928

Table 3 Accuracy of MinSUM (percentage)

Method	NY	BAY	COL	FLA	NW	CAL	LKS	W
RCM	100	100	100	100	100	100	100	100
ALBP ($\epsilon=0.1$)	99.9245	99.7951	99.1447	99.8406	99.9098	97.0645	99.9548	99.767
ALBP ($\epsilon=0$)	99.8824	96.4123	97.0905	97.2013	99.7308	92.5424	99.9544	98.8638
GD [3]	96.4302	95.2682	91.8355	86.327	89.7163	86.8864	75.7879	97.64

99% accuracy, and it ran in approximately half the time of the RCM. Tables 4 and 5 show the results for MinMAX. These experiments were similar to those of MinSUM, but it was confirmed that the average accuracy of GD was worse than the proposed method.

6.3 Evaluation of varying approximation factor

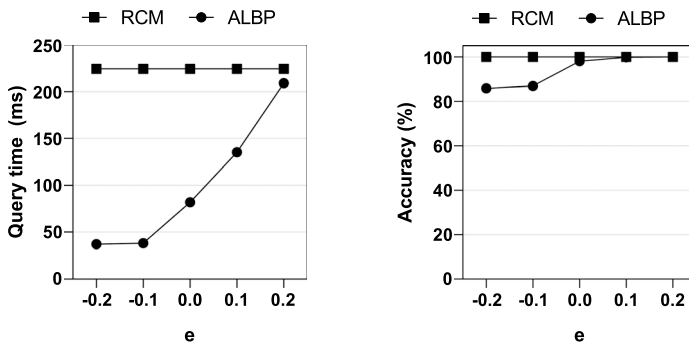
In this subsection, we investigated how the proposed method changes the query run time and accuracy as the approximation factor ϵ changes. Figures 8 and 9 show the experimental results for query time and accuracy of MinSUM and MinMAX, respectively, in the FLA road network when the query size $|Q|$ is 100. The baseline accuracy is approximately 85% or higher for MinSUM and 95% for MinMAX. The higher the approximation factor, the closer the accuracy is to 100%. Thus, the execution time could be improved by half while maintaining an accuracy rate of up to 85%, and it could be improved by a factor of 4 while maintaining an accuracy of up to 85%. Note that although in the proposed method, the trade-off between query time and accuracy can be adjusted through the parameter ϵ , the previous method, GD, could not control the trade-off between query time and accuracy. The accuracy of

Table 4 Query performance of MinMAX (ms)

Graph	NY	BAY	COL	FLA	NW	CAL	LKS	W
RCM	234.827	100.106	160.076	327.496	276.031	577.175	2285.77	952.722
ALBP ($\epsilon=0.1$)	182.089	84.1372	105.07	162.997	185.481	233.833	1181.49	296.557
ALBP ($\epsilon=0$)	137.287	53.5182	58.7826	105.823	128.511	143.515	775.881	258.895
GD [3]	24.3173	8.61378	11.2979	12.003	18.5047	33.5813	134.555	73.7306

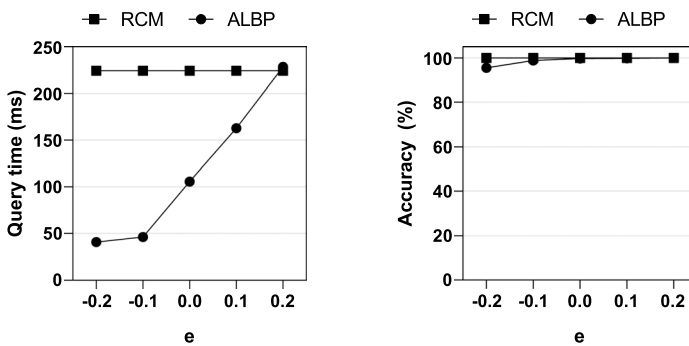
Table 5 Accuracy of MinMAX (percentage)

Method	NY	BAY	COL	FLA	NW	CAL	LKS	W
RCM	100	100	100	100	100	100	100	100
ALBP ($\epsilon=0.1$)	99.8923	99.7991	99.3219	99.8251	99.5362	97.0645	99.9548	99.767
ALBP ($\epsilon=0$)	99.3976	99.6125	98.6053	99.7393	98.8157	94.4174	97.1274	96.0884
GD [3]	89.2295	82.8375	90.5935	82.5875	90.3994	88.4228	83.5183	92.4451



(a) Query time of MINSUM on FLA (b) Accuracy of MINSUM on FLA

Fig. 8 Query run time and accuracy with varying ϵ of MINSUM on FLA



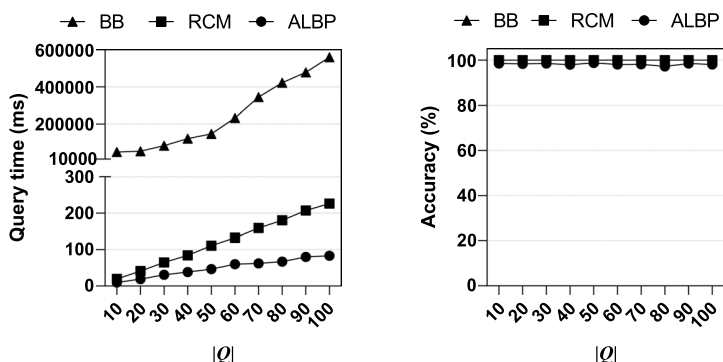
(a) Query time of MINMAX on FLA (b) Accuracy of MINMAX on FLA

Fig. 9 Query run time and accuracy with varying ϵ of MINMAX on FLA

GD is sufficient for real-world applications only when the distance distribution in the road network is convex.

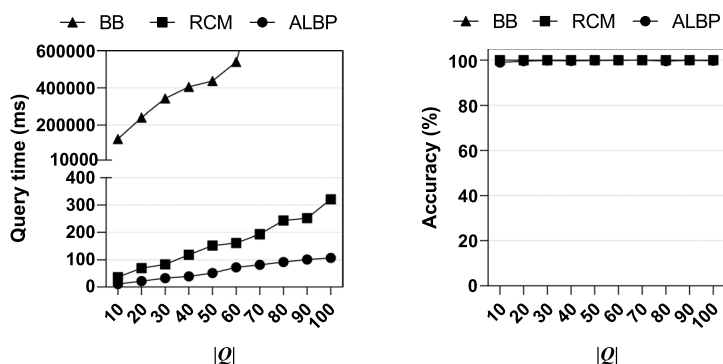
6.4 Scalability of varying query size

We investigated how the proposed methods change the query performance and accuracy according to various query sizes on the FLA dataset. The ALBP approximation factor ϵ was fixed at 0.0, and the query size was changed from 10 to 100. The BB method was performed up to 600 000 ms and forcibly terminated beyond that limit. Figures 10 and 11 show the query time and accuracy results with varying query sizes on MINSUM and MinMax, respectively. It can be seen that the accuracy of the query is close to 98%, while the approximation method, ALBP, performed nearly 50% faster than exact method, RCM. This result fits the reasoning of the lower bound tendency when designing the approximate lower bound method in Sect. 5.



(a) Query time of MINSUM on FLA (b) Accuracy of MINSUM on FLA

Fig. 10 Query run time and accuracy with varying $|Q|$ of MINSUM on FLA ($\epsilon = 0.0$)



(a) Query time of MINMAX on FLA (b) Accuracy of MINMAX on FLA

Fig. 11 Query run time and accuracy with varying $|Q|$ of MINMAX on FLA ($\epsilon = 0.0$)

However, the existing method, BB, is very time-consuming because the BB method uses the Euclidean distance to prune the search space and does not prune accurately.

7 Conclusion

In this paper, we present a concrete implementation of the ONLTP query in eight large road network scenarios. Our proposed exact method is efficient and always gives the correct answer. Furthermore, our approximation method is more accurate than the existing GD framework and is a more flexible way to control trade-offs between query time and accuracy. We experimentally demonstrated that our approximation method is nearly 99% accurate in all eight large road network scenarios while the query runs within an acceptable time.

Acknowledgements This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2019R1A2C1088126).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Zhong R, Li G, Tan K -L, Zhou L (2013) G-Tree: an efficient index for knn search on road networks. In: ACM CIKM, pp. 39–48
2. Zhong R, Li G, Tan K-L, Zhou L, Gong Z (2015) G-tree: an efficient and scalable index for spatial search on road networks. *IEEE Trans Knowl Data Eng* 27(8):2175–2189
3. Yan D, Zhao Z, Ng W (2011) Efficient algorithms for finding optimal meeting point on road networks. In: Proceedings of the VLDB Endowment, pp 1–11
4. Papadias D, Shen Q, Tao Y, Mouratidis K (2004) Group nearest neighbor queries. In: IEEE, 20th International Conference on Data Engineering, pp 301–312
5. Yiu ML, Mamoulis N, Papadias D (2005) Aggregate nearest neighbor queries in road networks. *IEEE Trans Knowl Data Eng* 17(6):820–833
6. Shang S, Chen L, Wei Z, Jensen CS, Wen JR, Kalnis P (2016) Collective travel planning in spatial networks. *IEEE Trans knowl Data Eng* 28(5):1132–1146
7. Cooper L (1968) An extension of the generalized Weber problem. *J Reg Sci* 8(2):181–197
8. Cooper L (1977) The multifacility location problem: applications and descent theorems. *J Reg Sci* 17(3):409–419
9. Sproull RF (1991) Refinements to nearest-neighbor searching in k -dimensional trees. *Algorithmica* 6:579–589
10. Guttman A (1984) R-trees: A dynamic index structure for spatial searching. In: ACM SIGMOD, International Conference on Management of Data, pp 47–57
11. Yan D, Zhao Z, Ng W (2015) Efficient processing of optimal meeting point queries in euclidean space and road networks. *Knowl Inf Syst* 42(2):319–351
12. Arya S, Mount DM, Netanyahu NS (1998) An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J ACM* 45(6):891–923
13. Ciaccia P, Patella M (2000) PAC nearest neighbor queries: approximate and controlled search in high dimensional and metric spaces. In: IEEE Proceedings of 16th International Conference on Data Engineering, pp 244–255
14. Corral A, Cañadas J, Vassilakopoulos M (2002) Approximate algorithms for distance-based queries in high-dimensional data spaces using R-trees. In: East European Conference on Advances in Databases and Information Systems, pp163-176
15. Corral A, Vassilakopoulos M (2005) On approximate algorithms for distance-based queries using R-trees. *Comput J* 48(2):220–238
16. Demetrescu C (2010) Road network datasets. <http://users.diag.uniroma1.it/challenge9/download.shtml> Accessed 1 July 2020
17. Li Z, Chen L, Wang Y (2019) G*-tree: an efficient spatial index on road networks. In: IEEE 35th International Conference on Data Engineering (ICDE), pp 268–279
18. Jung J, Park S, Kim Y, Park S (2019) Route recommendation with dynamic user preference on road networks. In: 2019 IEEE International Conference on Big Data and Smart Computing (BigComp), Kyoto, Japan, 2019, pp. 1–7. <https://doi.org/10.1109/BIGCOMP.2019.8679379>

19. Chen F, Zhang P, Lin H, Tang S (2019) Continuous path-based range keyword queries on road networks. In: 2019 IEEE International Conference on Big Knowledge (ICBK), Beijing, China, 2019, pp. 42–49. <https://doi.org/10.1109/ICBK.2019.00014>
20. Zhao J, Gao Y, Chen G, Chen R (2017) Towards efficient framework for time-aware spatial keyword queries on road networks. *ACM Transactions on Information Systems (TOIS)* 36, 3, Article 24 (April 2018), 48 pages. <https://doi.org/10.1145/3143802>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.