Article

A Unified Approach to Spatial Proximity Query Processing in Dynamic Spatial Networks

Hyung-Ju Cho

Department of Software, Kyungpook National University, 2559 Gyeongsang-daero, Sangju-si 37224, Gyeongsangbuk-do, Korea; hyungju@knu.ac.kr

Abstract: Nearest neighbor (NN) and range (RN) queries are basic query types in spatial databases. In this study, we refer to collections of NN and RN queries as spatial proximity (SP) queries. At peak times, location-based services (LBS) need to quickly process SP queries that arrive simultaneously. Timely processing can be achieved by increasing the number of LBS servers; however, this also increases service costs. Existing solutions evaluate SP queries sequentially; thus, such solutions involve unnecessary distance calculations. This study proposes a unified batch algorithm (UBA) that can effectively process SP queries in dynamic spatial networks. With the proposed UBA, the distance between two points is indicated by the travel time on the shortest path connecting them. The shortest travel time changes frequently depending on traffic conditions. The goal of the proposed UBA is to avoid unnecessary distance calculations for nearby SP queries. Thus, the UBA clusters nearby SP queries and exploits shared distance calculations for query clusters. Extensive evaluations using real-world roadmaps demonstrated the superiority and scalability of UBA compared with state-of-the-art sequential solutions.

Keywords: spatial proximity query; nearest neighbor query; range query; unified batch algorithm; dynamic spatial network

check for updates

Citation: Cho, H.-J. A Unified Approach to Spatial Proximity Query Processing in Dynamic Spatial Networks. *Sensors* **2021**, *21*, 5258. https://doi.org/10.3390/s21165258

Academic Editors: Guenther Retscher, Ondrej Krejcar, Vassilis Gikas and Michal Kačmařík

Received: 28 May 2021
Accepted: 2 August 2021
Published: 4 August 2021

1. Introduction

This study investigates a unified batch approach to spatial proximity (SP) queries in dynamic spatial networks. In the investigated approach, the distance between two points is the travel time of the shortest path connecting them, and the shortest travel time frequently changes depending on traffic conditions, such as traffic volume and accidents. In this study, SP queries refer to a collection of nearest neighbor (NN) and range (RN) queries, which are basic query types in spatial databases. NN queries retrieve points of interest (POI), such as taxis and restaurants, closest to a query user [1,2], and RN queries retrieve POIs within a query distance [3–5]. Typically, location-based services (LBS), such as taxi-booking and ride-sharing services, use real-time spatial data to locate POIs close to the query user [6–10]. When multiple SP queries reach an LBS server simultaneously at peak times, if the SP queries are processed sequentially, it may not be possible to provide prompt responses to the query users. This difficulty can be addressed by increasing the number of LBS servers or by developing state-of-the-art algorithms based on "one-query-at-a-time processing" [3,11–15] to process the SP queries quickly.

SP queries have many potential applications in dynamic spatial networks, such as ride-hailing and car parking facilities. For example, in 2020, the ride-hailing company Uber accomplished an average of 18.7 million trips per day [16], demonstrating the significance of scalable and efficient solutions to promptly match Uber cabs with passengers. Another example is real-time parking management, which helps drivers find a parking space close to their destination.

Figure 1 shows two snapshots of SP queries in a dynamic spatial network, where a set Q of SP query points and a set P of data points are expressed as $Q = \{q_1^{NN}, q_2^{RN}, q_3^{NN}\}$

and $P = \{p_1, p_2, p_3\}$, respectively. This study assumes that both the query points and data points run freely within the spatial network and that spatial segments often change their weights. In this example, q_1^{NN} and q_3^{NN} find the data points closest to q_1^{NN} and q_3^{NN}, respectively, and q_2^{RN} finds data points within a query distance (e.g., 4 km) to q_2^{RN}. As shown in Figure 1a, at timestamp t_i, data point p_1 is the closest to both q_1^{NN} and q_3^{NN}, and data point p_2 is within 4 km of q_2^{RN}. However, as shown in Figure 1b, at timestamp t_j, data points p_3 and p_1 are closest to q_1^{NN} and q_3^{NN}, respectively, and no data point is within 4 km of q_2^{RN}. A simple solution sequentially retrieves data points that satisfy the condition of each SP query in Q. However, this simple solution involves unnecessary network traversal, which can result in prohibitively high computational costs when a large number of SP queries reach the LBS server during peak hours [3,12–15]. Thus, we propose a unified batch algorithm (UBA) that can process SP queries in dynamic spatial networks effectively and efficiently.

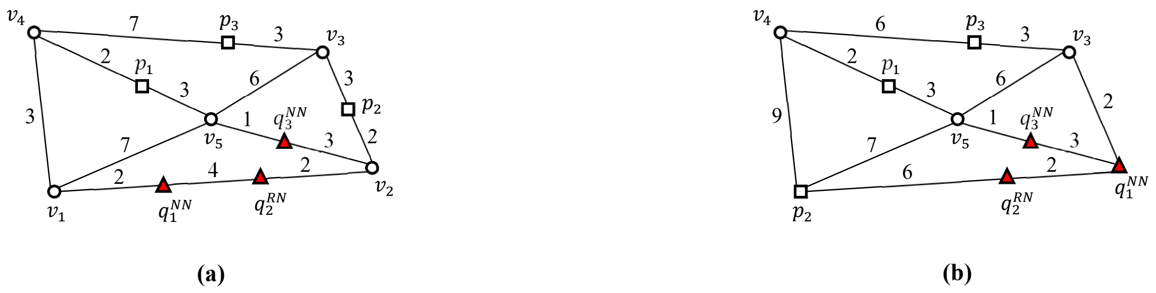

Figure 1. Snapshots of a set of SP queries, Q in a dynamic spatial network, where $Q = \{q_1^{NN}, q_2^{RN}, q_3^{NN}\}$: (**a**) at time t_i. (**b**) At time t_j.

Figure 2 is a system diagram of the proposed UBA between query points and LBS server. Query points send their locations and query requests to the LBS server (step 1). The LBS server collects the requests from query points and forwards them to the UBA (step 2). The UBA first groups nearby query points into query clusters for shared computation (step 3). Then, UBA retrieves candidate data points for each query cluster to avoid unnecessary network traversals (step 4). UBA evaluates each query using the candidate data points for the query cluster (step 5) and returns query results to the LBS server (step 6). Finally, the LBS server provides the result to each query point (step 7).

All nearest neighbor (ANN) queries [17,18] are similar to SP queries. However, ANN queries assume that each query point q in Q only finds a single data point closest to q and, therefore does not consider RN queries. This study considers a highly dynamic situation in which both query and data points run freely within a dynamic spatial network [19–21]. The proposed UBA can effectively process SP queries in dynamic spatial networks. For simplicity, this study considers NN queries rather than *k*NN queries, which retrieve k data points closest to the query user for a positive integer k. However, UBA can easily be extended to process *k*NN queries.

The primary contributions of this study are summarized as follows.

- A unified batch processing algorithm, i.e., UBA, is proposed for the batch processing of SP queries in dynamic spatial networks. The performance of UBA highly depends on the distribution of query points. Thus, UBA clearly outperforms sequential algorithms when query points display a skewed distribution. Conversely, UBA shows similar performance to sequential algorithms when query points display a uniform distribution.
- Clustering of SP queries and their shared computation are presented to avoid unnecessary distance computations. The correctness of UBA is proved using a lemma. Furthermore, a theoretical analysis is presented to establish the advantage of UBA over sequential algorithms, particularly when query points display a skewed distribution.

- An empirical study is conducted under various conditions to demonstrate the superiority and scalability of UBA compared with a sequential algorithm.

The remainder of this paper is organized as follows. Section 2 reviews related studies. Section 3 introduces the necessary preliminaries, including a definition of the notations and symbols used in this study. Section 4 explains how to cluster nearby SP queries into query clusters and presents the proposed UBA for SP queries in dynamic spatial networks. Section 5 presents an empirical study of UBA compared to a conventional algorithm under various conditions. Conclusions and suggestions for future work are presented in Section 6.

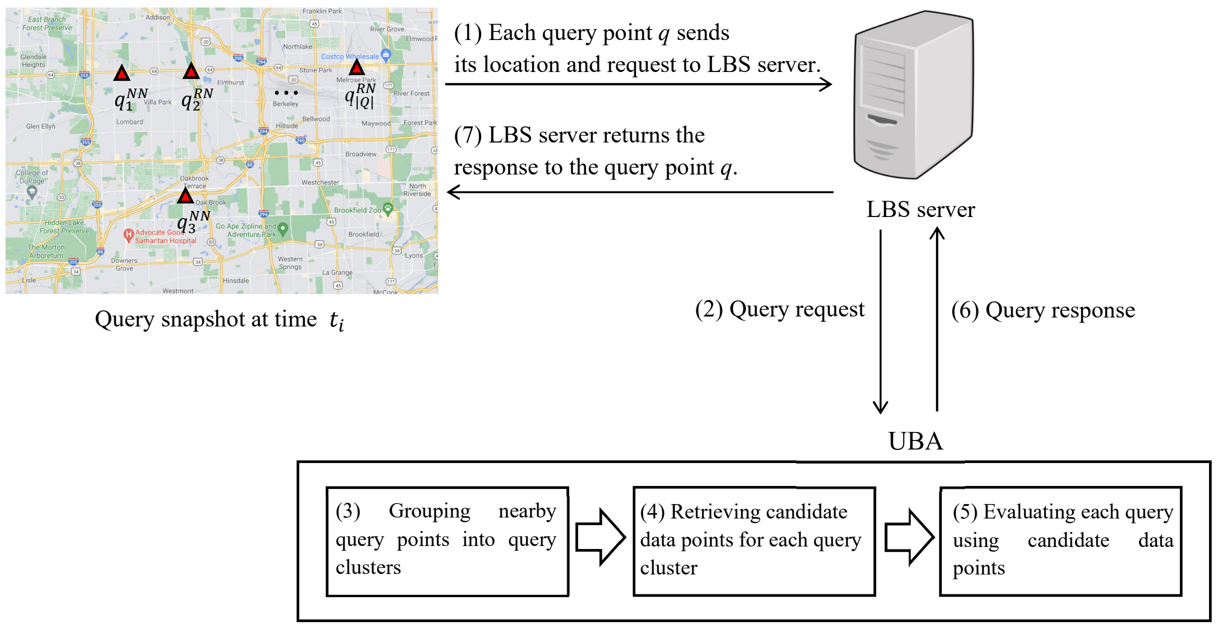
q_1^{NN}
q_2^{RN}
$q_{|Q|}^{RN}$
q_3^{NN}
Query snapshot at time t_i
(1) Each query point q sends its location and request to LBS server.
(7) LBS server returns the response to the query point q.
LBS server
(2) Query request
(6) Query response
UBA
(3) Grouping nearby query points into query clusters
(4) Retrieving candidate data points for each query cluster
(5) Evaluating each query using candidate data points

Figure 2. System diagram of UBA between query points and LBS server.

2. Related Work

Researchers developed algorithms and index structures to evaluate spatial queries, including NN and RN queries for LBSs [6,22–25]. When calculating the length of the shortest path between two points, spatial queries for dynamic spatial networks suffer from high computational cost because graph traversal is required at runtime. Therefore, numerous studies attempt to reduce the computational cost of the shortest path distance, to avoid unnecessary shortest-path computations [6,22–25]. Incremental Euclidean restriction (IER) and incremental network expansion (INE) were developed for NN queries [3]. IER assumes that the shortest path between two points is larger than or equal to the Euclidean distance. INE explores the spatial network incrementally from the query point, as in Dijkstra's algorithm, and investigates the data points in the encountered sequence. Range network expansion (RNE), which is similar to INE, was also developed for RN queries [3]. The route overlay and association directory method, ROAD [12], hierarchically divides the spatial network and pre-calculates the length of the shortest path between the border vertices within each partition. The distance-browsing method, DisBrw [13], exploits the spatially induced linkage cognizance index, and retains the length of the shortest path between each pair of vertices. G-tree [15] hierarchically divides the spatial network and uses an assembly based approach to compute the length of the shortest path between two vertices. V-tree [14] iteratively divides the spatial network into sub-networks and identifies the border vertices of each subnetwork. Then, the V-tree maintains a list of data points closest to each border vertex to quickly evaluate the *k*NN queries. A scalable and in-memory *k*NN query processing method called SIMkNN [26] was developed to quickly evaluate snapshot *k*NN queries over moving objects in a spatial network. The existing methods described in [6,22–25] are considered to be one-query-at-a-time processing algorithms because they aim to quickly evaluate a spatial query rather than a batch of spatial queries. This study is motivated by the observation that, with simple modifications, NN query processing algorithms can be applied to evaluate RN queries for spatial networks.

Multi-query optimization techniques were originally studied for relational database systems [27]. Their goal is to reduce the computational costs for a collection of queries that concurrently reach the database server by performing shared expressions once, materializing them temporarily, and then recycling them to evaluate other queries. Therefore, the subexpressions are typically evaluated once. These multi-query optimization techniques later expanded to involve query rewriting, query result caches, materialized views, and intermediate query results for relational database systems [28–36] and streaming processing systems [37–39]. Many applications involving high-load conditions have proven that batch processing algorithms can significantly reduce the query processing time for multiple simultaneous queries [19,30–43]. Furthermore, multi-query optimization techniques have received significant attention in spatial databases. Several batch shortest path algorithms also exist [19,40–44]. Furthermore, multi-query optimization techniques have received significant attention in spatial databases. Several batch shortest path algorithms [19,40–44] have been developed to efficiently evaluate multiple shortest path queries in spatial networks. However, these batch shortest path algorithms cannot be directly used to evaluate SP queries because of their diverse problem definitions. Several cache strategies for query results have been developed to efficiently process batches of *k*NN queries in spatial networks [6]. These strategies exploit the cached results of adjacent recently computed queries to efficiently process a batch of *k*NN queries. However, cache strategies have clear limitations in dynamic spatial networks, as their results may be invalidated by frequent updates to the weight of the spatial segments and by the movement of query points or data points. Finally, Li et al. [45–49] developed a series of algorithms for processing large complex networks, such as social networks. Specifically, they considered the trust management system based on game theory [47], dynamic clustering for electronic commerce systems [45], identifiability for the community detection [49], an optimal estimation of low-rank factors [48], and the identification of overlapping communities [46].

This work differs from existing studies in several respects. First, UBA considers SP queries in dynamic spatial networks. Second, UBA avoids dispensable network traversal by clustering SP queries and performing batch processing. Third, UBA can easily be incorporated into one-query-at-a-time processing algorithms for spatial networks [3,12,13,15].

3. Preliminaries

This section defines the terms and notations that are used in this paper.

Definition 1 (NN query [1,11,18,22,25])**.** *Given a query point q^{NN} and a set of data points P, an NN query retrieves data point p_{NN} closest to q^{NN} such that $dist(q^{NN}, p_{NN}) \leq dist(q^{NN}, p)$ holds for $\forall p_{NN} \in \Pi(q^{NN})$ and $\forall p \in P - \Pi(q^{NN})$.*

Definition 2 (RN query [3–5])**.** *Given a positive integer r, a query point q^{RN}, and a set of data points P, an RN query retrieves data points within query distance r to q^{RN} such that $dist(q^{RN}, p_{RN}) \leq r$ holds for $\forall p_{RN} \in \Pi(q^{RN})$.*

Definition 3 (Spatial network [3,9,11,25,26,41,50,51])**.** *A dynamic spatial network can be described as a dynamic weighted graph $G = \langle V, E, W \rangle$, where V, E, and W indicate the vertex set, edge set, and edge distance matrix, respectively. An edge has a nonnegative weight, e.g., travel time, and changes its weight frequently.*

Definition 4 (Intersection, intermediate, and terminal vertices)**.** *In this study, vertices are categorized via their degree. In this study, vertices are categorized via their degree as follows: (1) if the degree of a vertex is greater than or equal to three, the vertex is an intersection vertex; (2) if the degree is two, the vertex is an intermediate vertex; (3) if the degree is one, the vertex is a terminal* *vertex. For example, v_2 and v_3 in Figure 3 are intersection vertices, v_5 and v_6 are intermediate vertices, and v_1 and v_4 are terminal vertices.*

Definition 5 (Vertex sequence and segment)**.** *A vertex sequence $\overline{v_l v_{l+1} \ldots v_m}$ denotes a segment connecting two vertices v_l and v_m such that v_l and v_m are either an intersection vertex or a terminal vertex, and the other vertices in the segment, i.e., $v_{l+1}, \ldots, v_{m-1}$, are intermediate vertices. The length of a vertex sequence is the total weight of the edges in the vertex sequence. Parts of a vertex sequence are referred to as segments. By definition, a vertex sequence is also a segment. For example, Figure 3 has four vertex sequences $\overline{v_1v_2}$, $\overline{v_2v_3}$, $\overline{v_3v_4}$, and $\overline{v_2v_5v_6v_3}$. Examples of query segments in Figure 3 include $\overline{v_2v_5v_6}$, $\overline{v_5v_6}$ and $\overline{v_3v_6v_5}$.*

Table 1 summarizes the symbols and notations used in this study. Note that the query points are often used interchangeably to refer to SP queries. Figure 3 illustrates the difference between the distance and segment length between q_1 and q_2 in a spatial network. Here, the shortest path from q_1 to q_2 is $q_1 \rightarrow v_2 \rightarrow v_3 \rightarrow q_2$, whose distance $dist(q_1, q_2)$ is equal to eight. The segment connecting q_1 and q_2 (marked with a bold line) is $\overline{q_1v_5v_6q_2}$, and its length $len(\overline{q_1v_5v_6q_2})$ is equal to 10.

Table 1. Definitions of symbols.

Symbol	**Definition**
q^{NN} and q^{RN}	NN and RN queries, respectively
p	Data point
r	Query distance (e.g., 4 km)
P	Set of data points
Q	Set of query points
$\overline{Q_C}$ and $\overline{Q}$	Query cluster and a set of query clusters, respectively
$B(\overline{Q_C})$	Set of border points for $\overline{Q_C}$
$\Pi(q^{NN})$	Set of data points closest to query point q^{NN}
$\Pi(q^{RN})$	Set of data points within query distance r from a query point q^{RN}
$P(\overline{qp})$	Set of data points in a segment $\overline{qp}$
$dist(q, p)$	Length of the shortest path connecting points q and p
$len(\overline{qp})$	Length of the segment $\overline{qp}$ connecting points q and p
$\overline{v_l v_{l+1} \cdot\cdot\cdot v_m}$	Vertex sequence where v_l and v_m are not intermediate vertices, and the other vertices, $v_{l+1}, \ldots, v_{m-1}$, are intermediate vertices (in short, $\overline{v_l v_m}$)
$\overline{q_i q_{i+1} \cdot\cdot\cdot q_j}$	Query segment connecting nearby query points $q_i, q_{i+1}, \ldots, q_j$ (in short, $\overline{q_i q_j}$)

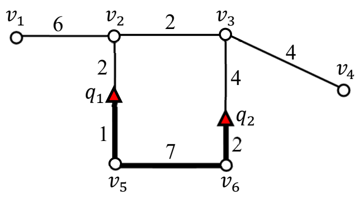

Figure 3. Difference between $dist(q_1, q_2) = 8$ and $len(\overline{q_1v_5v_6q_2}) = 10$.

4. Batch Processing of SP Queries in Spatial Networks

4.1. Clustering Nearby SP Queries

Here, we consider five SP queries $q_1^{NN}, q_2^{RN}, q_3^{NN}, q_4^{RN}$, and q_5^{NN} in a spatial network (Figure 4). Assume that the NN queries q_1^{NN}, q_3^{NN}, and q_5^{NN} find a data point closest to themselves and that the RN queries q_2^{RN} and q_4^{RN} find data points within query distance r (=4) to themselves.

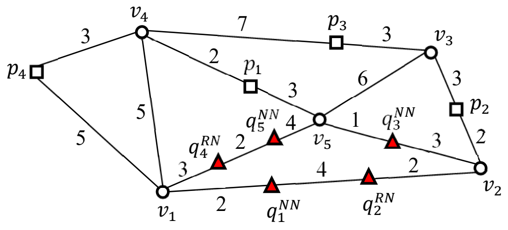

Figure 4. Population comprising five SP queries $q_1^{NN}, q_2^{RN}, q_3^{NN}, q_4^{RN}$, and q_5^{NN}.

Figure 5 shows an example of the two-step clustering method, which converts nearby query points into a query cluster. In the first step, query points in a vertex sequence are connected to a query segment (Figure 5a). As a result, three query segments $\overline{q_1^{NN}q_2^{RN}}$, q_3^{NN}, and $\overline{q_4^{RN}q_5^{NN}}$ are generated, where $\overline{q_1^{NN}q_2^{RN}}$ and $\overline{q_4^{RN}q_5^{NN}}$ connect two separate sets of query points, i.e., q_1^{NN} and q_2^{RN}, and q_4^{RN} and q_5^{NN}, respectively, in vertex sequences $\overline{v_1v_2}$ and $\overline{v_1v_5}$, respectively. In the second step, adjacent query segments are grouped into a query cluster using joint vertices (Figure 5b). The intersection vertex is referred to as a joint vertex when it is adjacent to greater than two query segments. As shown in Figure 5b, query segments $\overline{q_1^{NN}q_2^{RN}}$ and $\overline{q_4^{RN}q_5^{NN}}$ are adjacent to an intersection vertex v_1, which becomes a joint vertex for $\overline{q_1^{NN}q_2^{RN}}$ and $\overline{q_4^{RN}q_5^{NN}}$. Similarly, query segments, $\overline{q_1^{NN}q_2^{RN}}$ and q_3^{NN} are adjacent to intersection vertex v_2, which becomes a joint vertex for $\overline{q_1^{NN}q_2^{RN}}$ and q_3^{NN}. Finally, query segments q_3^{NN} and $\overline{q_4^{RN}q_5^{NN}}$ are adjacent to intersection vertex v_5, which becomes a joint vertex for q_3^{NN} and $\overline{q_4^{RN}q_5^{NN}}$. Therefore, the three query segments, $\overline{q_1^{NN}q_2^{RN}}$, q_3^{NN}, and $\overline{q_4^{RN}q_5^{NN}}$ are connected to a query cluster $\overline{Q_C} = \{\overline{q_1^{NN}q_2^{RN}}, q_3^{NN}, \overline{q_4^{RN}q_5^{NN}}\}$. In other words, the five query points $q_1^{NN}, q_2^{RN}, q_3^{NN}, q_4^{RN}$, and q_5^{NN} are clustered into query cluster $\overline{Q_C}$. Note that $\overline{Q_C}$ is represented by a set of query segments. Consequently, a set of query points $Q = \{q_1^{NN}, q_2^{RN}, q_3^{NN}, q_4^{RN},\ q_5^{NN}\}$ is converted into a set of query clusters $\overline{Q} = \{\{\overline{q_1^{NN}q_2^{RN}}, q_3^{NN}, \overline{q_4^{RN}q_5^{NN}}\}\}$.

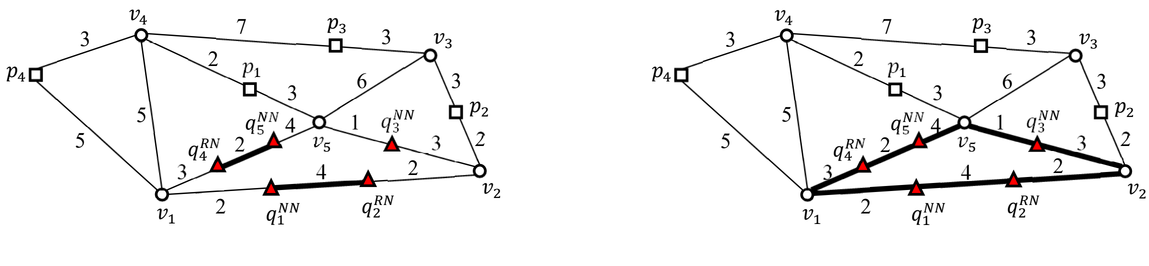

Figure 5. Clustering nearby query points into a query cluster: (**a**) connecting query points in a vertex sequence into a query segment. (**b**) Clustering adjacent query segments into a query cluster using joint vertices.

Next, we define the border point of query cluster $\overline{Q_C}$. Any point at which $\overline{Q_C}$ and its non-query cluster $G - \overline{Q_C}$ meet is referred to as the border point of $\overline{Q_C}$. In this example, $\overline{Q_C}$ has three border points, i.e., v_1, v_2, and v_5, where $\overline{Q_C}$ and its non-query cluster $G - \overline{Q_C}$ meet. Note that sequential solutions should evaluate the five SP queries shown in Figure 4. The two-step clustering method enables UBA to evaluate the three SP queries at border points v_1, v_2, and v_5 rather than at query points $q_1^{NN}, q_2^{RN}, q_3^{NN}, q_4^{RN}$, and q_5^{NN}.

Figure 6 illustrates the computation of the distance between query point q in query segment $\overline{q_iq_j}$ and data point p for the following cases: $p \notin \overline{q_iq_j}$ and $p \in \overline{q_iq_j}$. As shown in Figure 6a, when data point p is outside query segment $\overline{q_iq_j}$, i.e., $p \notin \overline{q_iq_j}$, the distance from q to p is given as $dist(q,p) = min\{len(\overline{qq_i}) + dist(q_i,p), len(\overline{qq_j}) + dist(q_j,p)\}$ because the shortest path between q and p is either $q \rightarrow q_i \rightarrow p$ or $q \rightarrow q_j \rightarrow p$. As shown in Figure 6b, when p is inside $\overline{q_iq_j}$, i.e., $p \in \overline{q_iq_j}$, the distance is given as $dist(q,p) = min\{len(\overline{qp}),\ len(\overline{qq_i}) + dist(q_i,p), len(\overline{qq_j}) + dist(q_j,p)\}$ because the shortest path between q and p is governed by one of the following three cases: $q \to p$, $q \to q_i \to p$, or $q \to q_j \to p$.

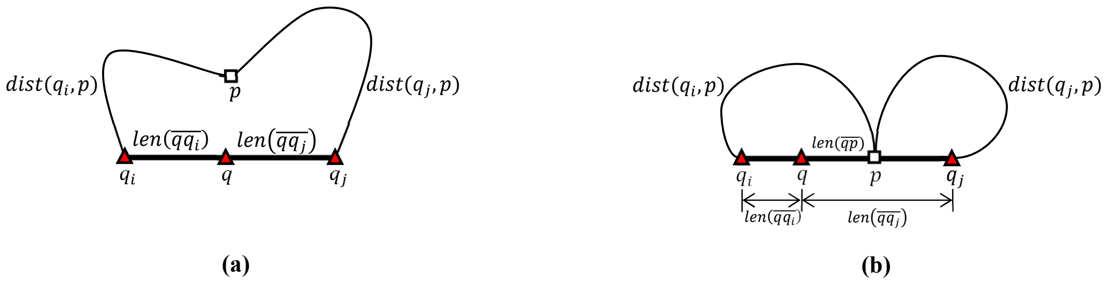

Figure 6. Computation of the distance between query point q in query segment $\overline{q_iq_j}$ and data point p: (**a**) $p \notin \overline{q_iq_j}$. (**b**) $p \in \overline{q_iq_j}$.

4.2. Unified Batch Processing Algorithm for SP Queries

Algorithm 1 provides the key concept of UBA for the unified batch processing of SP queries in a spatial network. Here, the result set $\Pi(Q)$ is initially set to an empty set (line 1). Then, the nearby query points are first grouped into query clusters (lines 2 and 3), as discussed in Section 4.1. A Cluster search then is executed for each query cluster $\overline{Q_C}$ to perform batch processing of the SP queries in $\overline{Q_C}$, and its query result is saved to $\Pi(\overline{Q_C})$ (line 6). Then, the query cluster result $\Pi(\overline{Q_C})$ is appended to $\Pi(Q)$, where $\Pi(\overline{Q_C}) = \{\langle q, \Pi(q)\rangle | q \in \overline{Q_C}\}$ and $\Pi(Q) = \{\langle q, \Pi(q)\rangle | q \in Q\}$ (line 7). When *cluster_search* (Algorithm 2) is performed for each query cluster in $\overline{Q}$, UBA terminates by returning the query result $\Pi(Q)$ (line 8).

Algorithm 1 $UBA(Q, P)$

Input: Q: collection of SP queries, P: collection of data points
Output: $\Pi(Q)$: collection of tuples of each SP query q in Q, and the query result for q, i.e., $\Pi(Q)=\{\langle q,\Pi(q)\rangle | q \in Q\}$

```
1: $\Pi(Q) \leftarrow \emptyset$    // The result set $\Pi(Q)$ is initially set to an empty set.
2: // Nearby query points are grouped into query clusters, as explained in Section 4.1.
3: $\overline{Q} \leftarrow cluster\_points(Q)$    // A set $Q$ of query points is changed into a set $\overline{Q}$ of query clusters.
4: // cluster_search function performs a batch processing of SP queries in $\overline{Q_C}$, as detailed in
   Algorithm 2.
5: for each query cluster $\overline{Q_C} \in \overline{Q}$ do
6:     $\Pi(\overline{Q_C}) \leftarrow cluster\_search(\overline{Q_C}, P)$    // Note that $\Pi(\overline{Q_C}) = \{\langle q,\Pi(q)\rangle | q \in \overline{Q_C}\}$.
7:     $\Pi(Q) \leftarrow \Pi(Q) \cup \Pi(\overline{Q_C})$    // The result for query points in a query cluster $\overline{Q_C}$, i.e., $\Pi(\overline{Q_C})$,
       is appended to $\Pi(Q)$.
8: return $\Pi(Q)$    // $\Pi(Q)$ is returned after the cluster search for all query clusters in $\overline{Q}$ is
   executed.
```

Algorithm 2 describes the cluster search algorithm employed to answer SP queries in query cluster $\overline{Q_C}$. Here, cluster search performs batch execution for a query cluster to avoid dispensable network traversal. This algorithm runs in two steps. In the first step, the SP queries are evaluated at the border points of $\overline{Q_C}$ rather than at the query points in $\overline{Q_C}$ (lines 3–6). Note that an SP query is either an NN or RN query; thus, the type of spatial query must be determined, which is evaluated at a border point b. If a query cluster $\overline{Q_C}$ includes only NN queries, an NN query is evaluated at the border point b, i.e., $SPQ(b, \overline{Q_C}) = \Pi(b^{NN})$. Similarly, if $\overline{Q_C}$ includes only RN queries, the $SPQ(b, \overline{Q_C})$ function evaluates an RN query at border point b, i.e., $SPQ(b, \overline{Q_C}) = \Pi(b^{RN})$. Finally, if $\overline{Q_C}$ includes both NN and RN queries, the $SPQ(b, \overline{Q_C})$ function evaluates the SP query that finds all the data points satisfying the NN or RN conditions at border point b, i.e., $SPQ(b, \overline{Q_C}) = \Pi(b^{NN}) \cup \Pi(b^{RN})$. In the second step, a shared computation is performed for each query segment $\overline{q_iq_j}$ in $\overline{Q_C}$ using the candidate data points obtained at the border points of $\overline{Q_C}$ (lines 7–10). Here, each SP query in $\overline{q_iq_j}$ chooses qualified data points from the candidate data points in $\Pi(b_i) \cup \Pi(b_j) \cup P(\overline{b_i b_j})$, where it is assumed that query segment $\overline{q_i q_j}$ belongs to segment $\overline{b_i b_j}$ in $\overline{Q_C}$. When the *segment_search* (Algorithm 3) is performed for each query segment in $\overline{Q_C}$, the *cluster_search* algorithm (Algorithm 2) terminates by returning the query result $\Pi(\overline{Q_C})$ (line 11).

Algorithm 2 *cluster_search*$(\overline{Q_C}, P)$

```
Input: $\overline{Q_C}$: query cluster, $P$: collection of data points
Output: $\Pi(\overline{Q_C})$: collection of tuples of each SP query $q$ in $\overline{Q_C}$, and the query result for $q$, i.e.,
    $\Pi(\overline{Q_C}) = \{\langle q, \Pi(q) \rangle | q \in \overline{Q_C}\}$
 1: // Note that $B(\overline{Q_C})$ refers to a set of border points in $\overline{Q_C}$.
 2: $\Pi(\overline{Q_C}) \leftarrow \emptyset$, $\Pi(B(\overline{Q_C})) \leftarrow \emptyset$   // Both $\Pi(\overline{Q_C})$ and $\Pi(B(\overline{Q_C}))$ are initially set to an empty set.
 3: // An SP query is evaluated at each border point $b$ of $\overline{Q_C}$ to retrieve candidate data points for
    $\overline{Q_C}$.
 4: for each border point $b \in B(\overline{Q_C})$ do
 5:     $\Pi(b) \leftarrow SPQ(b, \overline{Q_C})$   // An SP query is evaluated at a border point $b$, and its result is saved
        to $\Pi(b)$.
 6:     $\Pi(B(\overline{Q_C})) \leftarrow \Pi(B(\overline{Q_C})) \cup \Pi(b)$   // The query result at a border point $b$ of $\overline{Q_C}$ is appended
        to $\Pi(B(\overline{Q_C}))$.
 7: // $\overline{q_i q_j}$ is assumed to belong to a segment $\overline{b_i b_j}$ in $\overline{Q_C}$.
 8: for each query segment $\overline{q_i q_j} \in \overline{Q_C}$ do
 9:     $\Pi(\overline{q_i q_j}) \leftarrow segment\_search(\overline{q_i q_j}, \Pi(b_i) \cup \Pi(b_j) \cup P(\overline{b_i b_j}))$   // segment_search is detailed in
        Algorithm 3.
10:     $\Pi(\overline{Q_C}) \leftarrow \Pi(\overline{Q_C}) \cup \Pi(\overline{q_i q_j})$   // The result for a query segment $\overline{q_i q_j}$, i.e., $\Pi(\overline{q_i q_j})$, is
        appended to $\Pi(\overline{Q_C})$.
11: return return $\Pi(\overline{Q_C})$   // cluster_search ends by returning the batch result $\Pi(\overline{Q_C})$ for the SP
    queries in $\overline{Q_C}$.
```

Algorithm 3 *segment_search*$(\overline{q_i q_j}, \Pi(b_i) \cup \Pi(b_j) \cup P(\overline{b_i b_j}))$

```
Input: $\overline{q_i q_j}$: query segment in $\overline{Q_C}$, $\Pi(b_i) \cup \Pi(b_j) \cup P(\overline{b_i b_j})$: collection of candidate data points of SP
    queries in $\overline{q_i q_j}$
Output: $\Pi(\overline{q_i q_j})$: collection of tuples of each query $q$ in $\overline{q_i q_j}$ and the query result for $q$, i.e.,
    $\Pi(\overline{q_i q_j}) = \{\langle q, \Pi(q) \rangle | q \in \overline{q_i q_j}\}$
 1: $\Pi(\overline{q_i q_j}) \leftarrow \emptyset$   // $\Pi(\overline{q_i q_j})$ is initially set to an empty set.
 2: for each SP query $q \in \overline{q_i q_j}$ do
 3:     $\Pi(q) \leftarrow \emptyset$   // $\Pi(q)$ is initially set to an empty set.
 4:     for each candidate data point $p \in \Pi(b_i) \cup \Pi(b_j) \cup P(\overline{b_i b_j})$ do
 5:         // Step 1: $dist(q, p)$ is evaluated considering the two cases $p \notin \overline{b_i b_j}$ and $p \in \overline{b_i b_j}$, which are
            shown in Figure 6.
 6:         if $p$ is outside $\overline{b_i b_j}$ then
 7:             $dist(q, p) \leftarrow min\{len(\overline{q b_i}) + dist(b_i, p), len(\overline{q b_j}) + dist(b_j, p)\}$   // See Figure 6a.
 8:         else
 9:             $dist(q, p) \leftarrow min\{len(\overline{qp}), len(\overline{q b_i}) + dist(b_i, p), len(\overline{q b_j}) + dist(b_j, p)\}$   // See
                Figure 6b.
10:         // Step 2: $p$ is appended to $\Pi(q)$ when it satisfies the query condition.
11:         if $q = q^{NN}$ and $dist(q, p) \leq dist(q, p_{NN})$ then
12:             $\Pi(q) \leftarrow \Pi(q) \cup \{p\} - \{p_{NN}\}$   // $p$ replaces $p_{NN}$ that is the current NN of $q$ so far.
13:         else if $q = q^{RN}$ and $dist(q, p) \leq q.r$ then
14:             $\Pi(q) \leftarrow \Pi(q) \cup \{p\}$   // If $dist(q, p) \leq q.r$, $p$ is simply appended to $\Pi(q)$.
15:     $\Pi(\overline{q_i q_j}) \leftarrow \Pi(\overline{q_i q_j}) \cup \Pi(q)$
16: return $\Pi(\overline{q_i q_j})$   // segment_search ends by returning the batch result $\Pi(\overline{q_i q_j})$ for the SP
    queries in $\overline{q_i q_j}$.
```

Algorithm 3 describes the segment search algorithm employed to answer the SP queries in a query segment $\overline{Q_C}$ using the candidate data points in $\Pi(b_i) \cup \Pi(b_j) \cup P(\overline{b_i b_j})$. Here, the batch query result for $\overline{q_i q_j}$, i.e., $\Pi(\overline{q_i q_j})$, is initially set to an empty set (line 1). The distance between a query point q in $\overline{q_i q_j}$ and a candidate data point p, i.e., $dist(q, p)$ is then calculated (lines 5–9), as shown in Figure 6. When p is outside $\overline{b_i b_j}$, i.e., $p \notin \overline{b_i b_j}$, the distance from q to p is given as $dist(q,p) = min\{len(\overline{qb_i}) + dist(b_i,p), len(\overline{qb_j}) + dist(b_j,p)\}$. When p is inside $\overline{b_ib_j}$, i.e., $p \in \overline{b_ib_j}$, the distance from q to p is given as $dist(q,p) = min\{len(\overline{qp}), len(\overline{qb_i}) + dist(b_i,p), len(\overline{qb_j}) + dist(b_j,p)\}$. If query point q is an NN query and candidate data point p is closer to q than the current NN p_{NN}, then p is appended to $\Pi(q)$ and p_{NN} is removed from $\Pi(q)$, i.e., $\Pi(q) \leftarrow \Pi(q) \cup \{p\} - \{p_{NN}\}$ (lines 11–12). Similarly, if query point q is an RN query and $dist(q,p)$ is not greater than the query distance $q.r$, then p is simply appended to $\Pi(q)$, i.e., $\Pi(q) \leftarrow \Pi(q) \cup \{p\}$, where $q.r$ is the query distance of q (lines 13–14). The *Segment_search* algorithm (Algorithm 3) ends by returning the batch result $\Pi(\overline{q_iq_j})$ for $\overline{q_iq_j}$ (line 16).

Lemma 1 proves the correctness of UBA, which means that each query point q in a query cluster $\overline{Q_c}$ can retrieve its qualified data points from the candidate data points for $\overline{Q_c}$.

Lemma 1. *Each query point q in a query cluster $\overline{Q_c}$ can retrieve its qualified data points from the candidate data points for $\overline{Q_c}$.*

Proof. We prove Lemma 1 by contradiction under the assumption that there exists a qualified data point p for query point q in $\overline{Q_c}$ such that p is not a candidate data point for $\overline{Q_c}$. Clearly, set $\Sigma(\overline{Q_c})$ of candidate data points for $\overline{Q_c}$ is the union of set $P(\overline{Q_c})$ of data points inside $\overline{Q_c}$ and the SP query result $SPQ(b, \overline{Q_c})$ at each border point of $\overline{Q_c}$ as follows: $\Sigma(\overline{Q_c}) = P(\overline{Q_c}) \cup (SPQ(b_l, \overline{Q_c}) \cup SPQ(b_{l+1}, \overline{Q_c}) \cup \ldots \cup SPQ(b_m, \overline{Q_c}))$ where it is assumed that $B(\overline{Q_C}) = \{b_l, b_{l+1}, \ldots, b_m\}$. Clearly, this data point p must be outside $\overline{Q_c}$. This is because as illustrated in Figure 6b, qualified data point p inside $\overline{Q_c}$ becomes a candidate data point for $\overline{Q_c}$ according to the definition of $\Sigma(\overline{Q_c})$. When qualified data point p is outside $\overline{Q_c}$ as illustrated in Figure 6a, the following two cases should be considered: $\exists p((q^{RN} \in \overline{Q_c} \wedge p \in \prod(q^{RN})) \rightarrow p \notin \Sigma(\overline{Q_c}))$ and $\exists p((q^{NN} \in \overline{Q_c} \wedge p \in \prod(q^{NN})) \rightarrow p \notin \Sigma(\overline{Q_c}))$. In the first case, i.e., $\exists p((q^{RN} \in \overline{Q_c} \wedge p \in \prod(q^{RN})) \rightarrow p \notin \Sigma(\overline{Q_c}))$, qualified data point p satisfies the range query q^{RN}; however, it is not a candidate data point for $\overline{Q_c}$. In the second case, i.e., $\exists p((q^{NN} \in \overline{Q_c} \wedge p \in \prod(q^{NN})) \rightarrow p \notin \Sigma(\overline{Q_c}))$, qualified data point p satisfies the NN query q^{NN}; however, it is not a candidate data point for $\overline{Q_c}$. The shortest path from q^{RN} to p should pass through a border point of $\overline{Q_c}$. For convenience, assume that the shortest path from q^{RN} to p is $q^{RN} \rightarrow b_l \rightarrow p$ where b_l is a border point of $\overline{Q_c}$. Note that the distance from q^{RN} to p is less than or equal to query distance r, i.e., $dist(q^{RN}, p) \leq r$. Thus, the distance from the border point b_l to p is also less than or equal to r, i.e., $dist(b_l, p) \leq r$. This leads to a contradiction to the assumption that the qualified data point p for q^{RN} is not a candidate data point for $\overline{Q_c}$. Next, consider the second case that the qualified data point p for q^{NN} is not a candidate data point for $\overline{Q_c}$. For convenience, assume that the shortest path from q^{NN} to p is $q^{NN} \rightarrow b_l \rightarrow p$ and that a data point p_l is the NN of b_l rather than p. This means that p_l is closer to b_l than p, i.e., $dist(b_l, p_l) < dist(b_l, p)$. Note that the shortest path from q^{NN} to p (p_l) is $q^{NN} \rightarrow b_l \rightarrow p$ ($q^{NN} \rightarrow b_l \rightarrow p_l$). Thus, p_l should be the NN of q^{NN} rather than p. This leads to a contradiction to the assumption that p is the NN of q^{NN}. Therefore, each query point q in a query cluster $\overline{Q_c}$ can retrieve its qualified data points from the candidate data points for $\overline{Q_c}$. □

Table 2 compares the time complexities of UBA and sequential algorithms, such as INE [3] and RNE [3], for dynamic spatial networks. Note that UBA is independent of the one-query-at-a-time processing algorithms [3,11–15] and can be easily incorporated into these algorithms. For simplicity, INE and RNE are considered to evaluate a single SP query in dynamic spatial networks, and their time complexity is $O(|E| + |V| \cdot \log|V|)$. UBA evaluates as many as $M \cdot |\overline{Q}|$ SP queries, where $|\overline{Q}|$ is the number of query clusters in $\overline{Q}$ and M is the maximum number of border points in $\overline{Q_C}$, i.e., $M = max\{|B(\overline{Q_C})| \mid \overline{Q_C} \in \overline{Q}\}$. Conversely, sequential algorithms evaluate as many as $|Q|$ SP queries because each query point should be handled individually. Thus, the time complexities of UBA and the sequential algorithms are $O(|\overline{Q}| \cdot (|E| + |V| \cdot \log|V|))$ and $O(|Q| \cdot (|E| + |V| \cdot \log|V|))$, respectively. The results of the time complexity analysis indicate that UBA is superior to sequential algorithms, particularly when $|\overline{Q}| \ll |Q|$, i.e., the query points exhibit a highly skewed distribution. In addition, the results demonstrate that UBA shows similar performance to sequential algorithms when $|\overline{Q}| \cong |Q|$, i.e., the query points exhibit a uniform distribution.

Table 2. Comparison of time complexities of UBA and sequential algorithms.

	UBA	**Sequential Algorithms**
Number of SP queries to be evaluated	$M \cdot |\overline{Q}|$	$|Q|$
Time complexity to evaluate a SP query	$O(|E| + |V| \cdot \log|V|)$	$O(|E| + |V| \cdot \log|V|)$
Time complexity to evaluate SP queries in Q	$O(|\overline{Q}| \cdot (|E| + |V| \cdot \log|V|))$	$O(|Q| \cdot (|E| + |V| \cdot \log|V|))$

4.3. Evaluation of Example SP Queries Using UBA

This section describes the process used to evaluate five example SP queries using UBA. As shown in Figure 5, the five SP queries q_1^{NN}, q_2^{RN}, q_3^{NN}, q_4^{RN}, and q_5^{NN} are grouped into a query cluster $\overline{Q_C} = \{\overline{q_1^{NN}q_2^{RN}}, q_3^{NN}, \overline{q_4^{RN}q_5^{NN}}\}$, whose border points are v_1, v_2, and v_5. Clearly, a set of query points $Q=\{q_1^{NN}, q_2^{RN}, q_3^{NN}, q_4^{RN}, q_5^{NN}\}$ is transformed into a set of query clusters $\overline{Q} = \{\{\overline{q_1^{NN}q_2^{RN}}, q_3^{NN}, \overline{q_4^{RN}q_5^{NN}}\}\}$. Note that UBA evaluates only three SP queries at the border points of $\overline{Q}$ rather than the five query points q_1^{NN}, q_2^{RN}, q_3^{NN}, q_4^{RN}, and q_5^{NN}. Note that $\overline{Q_C}$ includes both the NN queries (q_1^{NN}, q_3^{NN}, and q_5^{NN}) and RN queries (q_2^{RN} and q_4^{RN}); thus the results of the SP queries at the border points v_1, v_2, and v_5 should be $\Pi(v_1) = \Pi(v_1^{NN}) \cup \Pi(v_1^{RN})$, $\Pi(v_2) = \Pi(v_2^{NN}) \cup \Pi(v_2^{RN})$, and $\Pi(v_5) = \Pi(v_5^{NN}) \cup \Pi(v_5^{RN})$, respectively. Table 3 shows the results of the SP queries at the three border points v_1, v_2, and v_5.

Table 3. Computation of the SP queries at the border points.

Border Point b	$\boldsymbol{\Pi(b^{NN})}$	$\boldsymbol{\Pi(b^{RN})}$	$\boldsymbol{\Pi(b^{NN}) \cup \Pi(b^{RN})}$
v_1	$\{p_4\}$	$\varnothing$	$\{p_4\}$
v_2	$\{p_2\}$	$\{p_2\}$	$\{p_2\}$
v_5	$\{p_1\}$	$\{p_1\}$	$\{p_1\}$

The *Segment_search* algorithm (Algorithm 3) is called for each query segment in $\overline{Q_C}$. For convenience, the three query segments $\overline{q_1^{NN}q_2^{RN}}$, q_3^{NN}, and $\overline{q_4^{RN}q_5^{NN}}$ are processed sequentially. First, the *segment_search* function evaluates the SP queries in $\overline{q_1^{NN}q_2^{RN}}$ with the candidate data points in $\Pi(v_1^{NN}) \cup \Pi(v_1^{RN}) \cup \Pi(v_2^{NN}) \cup \Pi(v_2^{RN}) \cup P(\overline{v_1v_2}) = \{p_2, p_4\}$. This function computes the distance between each pair of query points q_1^{NN} and q_2^{RN} in $\overline{q_1^{NN}q_2^{RN}}$, and the candidate data points p_2 and p_4. Table 4 summarizes the distances between each pair of query points q in query segment $\overline{q_iq_j}$ and their candidate data points p. Here, the SP query q_1^{NN} finds the data point closest to q_1^{NN} from the candidate data points p_2 and p_4. Consequently, p_4 is the chosen NN of q_1^{NN} because p_4 is closer to q_1^{NN} than p_2 (Table 4). Similarly, the SP query q_2^{RN} locates data points within a query distance $r = 4$ to q_2^{RN}. Accordingly, p_2 is included in the result of q_2^{RN} because $dist(q_2^{RN}, p_2) = 4$ and $dist(q_2^{RN}, p_4) = 11$ (Table 4). The query result for $\overline{q_1^{NN}q_2^{RN}}$ is $\Pi(\overline{q_1^{NN}q_2^{RN}}) = \Pi(q_1^{NN}) \cup \Pi(q_2^{RN}) = \{\langle q_1^{NN}, \{p_4\}\rangle, \langle q_2^{RN}, \{p_2\}\rangle\}$.

Next, the *segment_search* function evaluates the SP queries in q_3^{NN} with the candidate data points in $\Pi(v_2^{NN}) \cup \Pi(v_2^{RN}) \cup \Pi(v_5^{NN}) \cup \Pi(v_5^{RN}) \cup P(\overline{v_2v_5}) = \{p_1, p_2\}$. First, the distance between each pair of query points q_3^{NN} and then candidate data points p_1 and p_2 is computed. Then, the SP query q_3^{NN} locates the data point that is closest to q_3^{NN} in p_1 and p_2. Consequently, p_1 is the chosen NN of q_3^{NN} because p_1 is closer to q_3^{NN} than p_2 (Table 4). The query result for q_3^{NN} is $\Pi(q_3^{NN}) = \{\langle q_3^{NN}, \{p_1\}\rangle\}$.

Table 4. Computation of the distances between the queries and the candidate data points.

q	p	**Condition**	$dist(q,p)$	$\Pi(q)$
q_1^{NN}	p_2	$p_2 \notin \overline{v_1v_2}$	$dist(q_1^{NN}, p_2) = 8$	$\Pi(q_1^{NN}) = \{p_4\}$
	p_4	$p_4 \notin \overline{v_1v_2}$	$dist(q_1^{NN}, p_4) = 7$	
q_2^{RN}	p_2	$p_2 \notin \overline{v_1v_2}$	$dist(q_2^{RN}, p_2) = 4$	$\Pi(q_2^{RN}) = \{p_2\}$
	p_4	$p_4 \notin \overline{v_1v_2}$	$dist(q_2^{RN}, p_4) = 11$	
q_3^{NN}	p_1	$p_1 \notin \overline{v_2v_5}$	$dist(q_3^{NN}, p_1) = 4$	$\Pi(q_3^{NN}) = \{p_1\}$
	p_2	$p_2 \notin \overline{v_2v_5}$	$dist(q_3^{NN}, p_2) = 5$	
q_4^{RN}	p_1	$p_1 \notin \overline{v_1v_5}$	$dist(q_4^{RN}, p_1) = 9$	$\Pi(q_4^{RN}) = \varnothing$
	p_4	$p_4 \notin \overline{v_1v_5}$	$dist(q_4^{RN}, p_4) = 8$	
q_5^{NN}	p_1	$p_1 \notin \overline{v_1v_5}$	$dist(q_5^{NN}, p_1) = 7$	$\Pi(q_5^{NN}) = \{p_1\}$
	p_4	$p_4 \notin \overline{v_1v_5}$	$dist(q_5^{NN}, p_4) = 10$	

Finally, the *segment_search* function evaluates the SP queries in $\overline{q_4^{RN}q_5^{NN}}$ using the candidate data points in $\Pi(v_1^{NN}) \cup \Pi(v_1^{RN}) \cup \Pi(v_5^{NN}) \cup \Pi(v_5^{RN}) \cup P(\overline{v_1v_5}) = \{p_1, p_4\}$. First, the distances between each pair of query points in $\overline{q_4^{RN}q_5^{NN}}$ and then the candidate data points p_1 and p_4 are calculated. The SP query q_4^{RN} locates the data points within a query distance $r = 4$ to q_4^{RN}. No data points belong to the result set of q_4^{RN} because $dist(q_4^{RN}, p_1) = 9$ and $dist(q_4^{RN}, p_4) = 8$ (Table 4). The SP query q_5^{NN} identifies the data point that is closest to q_5^{NN} in p_1 and p_4. Consequently, p_1 is the chosen NN of q_5^{NN} because p_1 is closer to q_5^{NN} than p_4 (Table 4). The query result for $\overline{q_4^{RN}q_5^{NN}}$ is $\Pi(\overline{q_4^{RN}q_5^{NN}}) = \Pi(q_4^{RN}) \cup \Pi(q_5^{NN}) = \{\langle q_4^{RN}, \varnothing\rangle, \langle q_5^{NN}, \{p_1\}\rangle\}$. Clearly, the results of the SP queries in Q are the union of the results for the query segments in $\overline{Q_C}$: $\Pi(Q) = \Pi(\overline{q_1^{NN}q_2^{RN}}) \cup \Pi(q_3^{NN}) \cup \Pi(\overline{q_4^{RN}q_5^{NN}}) = \{\langle q_1^{NN}, \{p_4\}\rangle, \langle q_2^{RN}, \{p_2\}\rangle, \langle q_3^{NN}, \{p_1\}\rangle, \langle q_4^{RN}, \varnothing\rangle, \langle q_5^{NN}, \{p_1\}\rangle\}$.

5. Performance Study

In this section, the results from an empirical analysis of UBA are presented and compared with those of the conventional method [3]. The experimental settings are described in Section 5.1 and the experimental results are presented in Section 5.2.

5.1. Experimental Settings

Three real-world spatial networks [52] (Table 5) were used for the empirical study. These real-world spatial networks have different sizes and are part of the United States road network. For convenience, the extents of the spatial networks were normalized to a unit square $[0,\ 1]^2$, and the query distance r was set to 10^{-2}. The query points followed a centroid distribution, and the data points followed either a centroid or uniform distribution. Here, centroid-based points were generated to mimic highly skewed distributions of POIs in the real world. First, the centroids $c_1, c_2, \ldots, c_{|C|}$ were selected randomly based on the extent of the spatial networks, where $|C|$ is to the number of centroids. The points around each centroid followed a normal distribution, with the mean indicating the centroid, and the standard deviation was set to $\sigma = 10^{-2}$. A total of 1–10 centroids were selected as the query points, and five centroids were selected as the data points. The number of NN queries was the same as that of the RN queries for the SP queries. The experimental parameters are listed in Table 6. In each experiment, a single parameter was varied within the range, and the other parameters were maintained at their default values (shown in bold).

Table 5. Real-world roadmaps.

Name	Description	Vertices	Edges	Intersection Vertices	Vertex Sequences
CAL	California and Nevada	1,890,815	2,315,222	995,408	1,794,708
FLA	Florida	1,070,376	1,343,951	615,172	1,100,675
COL	Colorado	435,666	521,200	206,069	374,355

Table 6. Experimental parameter settings.

Parameter	Range
Number of query points ($\|Q\|$)	1, 3, 5, 7, **10** ($\times 10^3$)
Number of data points ($\|P\|$)	1, 3, 5, 7, **10** ($\times 10^3$)
Distribution of query points in Q	**(C)entroid**
Distribution of data points in P	(U)niform, (C)entroid
Number of centroids for query points in Q	**1**, 3, 5, 7, 10
Number of centroids for data points in P	**5**
Standard deviation for normal distribution (σ)	$\mathbf{10^{-2}}$
Query distance (r)	$\mathbf{10^{-2}}$
Number of NN queries in Q	$0.5 \times \|Q\|$
Roadmap	CAL, FLA, COL

Next, the proposed UBA was compared in terms of query processing time and the number of evaluated SP queries to a sequential algorithm called SEQ, which computes SP queries sequentially. Here, it was assumed that the query and data points moved freely within the dynamic spatial networks. Note that it is impractical to exploit the precomputation techniques presented in the literature [12,13,15] because the precomputed distances might be invalidated frequently when the query and data points run freely within a dynamic spatial network. UBA and SEQ use common subroutines for similar tasks, e.g., the evaluation of SP queries at a single query point; thus, both algorithms were implemented in C++ using the Microsoft Visual Studio 2019 development environment. The experiments were executed on a desktop computer running the Windows 10 operating system with 32 GB RAM and a 3.1 GHz processor (i9-9900). As in many recent studies [11,26,53], the indexing structures for UBA and SEQ remained in main memory to provide prompt responses, which are crucial in online map services. The experiments were repeated 10 times, and the average processing time was measured to determine the SP queries in Q. As stated previously, the proposed UBA is orthogonal to one-query-at-a-time processing algorithms [3,11–15] and can be easily incorporated into these algorithms. In this study, INE [3] and RNE [3] were used to evaluate the NN and RN queries, respectively, for the dynamic spatial networks because INE and RNE are based on network expansion similar to Dijkstra's algorithm, which is well-suited to dynamic spatial networks.

5.2. Experimental Results

Figure 7 compares the query processing times of UBA and SEQ to evaluate the SP queries in the CAL roadmap. In Figures 7–9, the three upper-row and three bottom-row charts show the experimental results when the data points followed a uniform distribution and a centroid distribution, respectively. Each chart shows the query processing time and number of evaluated SP queries by varying one parameter at a time (Table 6). The values in parentheses in Figures 7–10 indicate the number of SP queries evaluated by the proposed UBA. Note that the numbers of SP queries evaluated by SEQ were omitted because these numbers were exactly equal to $|Q|$ of the SP queries in Q. Figure 7a shows the query processing times of UBA and SEQ when $|Q|$ of the query points was between 1 K and 10 K, i.e., 1 K $\leq |Q| \leq$ 10 K. As can be seen, the proposed UBA clearly outperformed SEQ as the number of SP queries in Q increased. In terms of query processing times, UBA was up to 2.9 times faster than SEQ for $|Q| = 7$ K. However, UBA was up to 2.59 times slower than SEQ for $|Q| = 1$ K. Note that the proposed UBA was not sensitive to $|Q|$, unlike SEQ, which means that the effectiveness of batch processing in UBA increased as $|Q|$ increased. When $|Q| = 1$ K, 3 K, 5 K, 7 K, and 10 K, UBA evaluated fewer SP queries than SEQ by 75%, 89%, 88%, 91%, and 92%, respectively. Figure 7b shows the query processing times when $|P|$ of data points was varied between 1 K and 10 K, i.e., 1 K $\leq |P| \leq$ 10 K. Thus, UBA clearly outperformed SEQ in all cases. The query processing times of UBA were up to 8.9 times lower than those of SEQ when $|P| = 1$ K. As the $|P|$ value decreased, the search space for the NN query processing increased. Regardless of the change in $|P|$, UBA and SEQ evaluated 789 and 10,000 SP queries, respectively. Figure 7c shows the query processing times when $|C|$ of the centroids for the query points was varied between 1 and 10, i.e., $1 \leq |C| \leq 10$. The proposed UBA was up to 2.3 times faster than SEQ for all cases. As $|C|$ increased, the difference in query processing times between UBA and SEQ decreased because increasing $|C|$ led to a reduced density of the query points, which resulted in an increased $|\overline{Q}|$ value. Specifically, when $|C|$ =1, 3, 5, 7, and 10, UBA evaluated 789, 1196, 2438, 3928, and 4015 SP queries, respectively, whereas SEQ evaluated 10 K SP queries for all these cases.

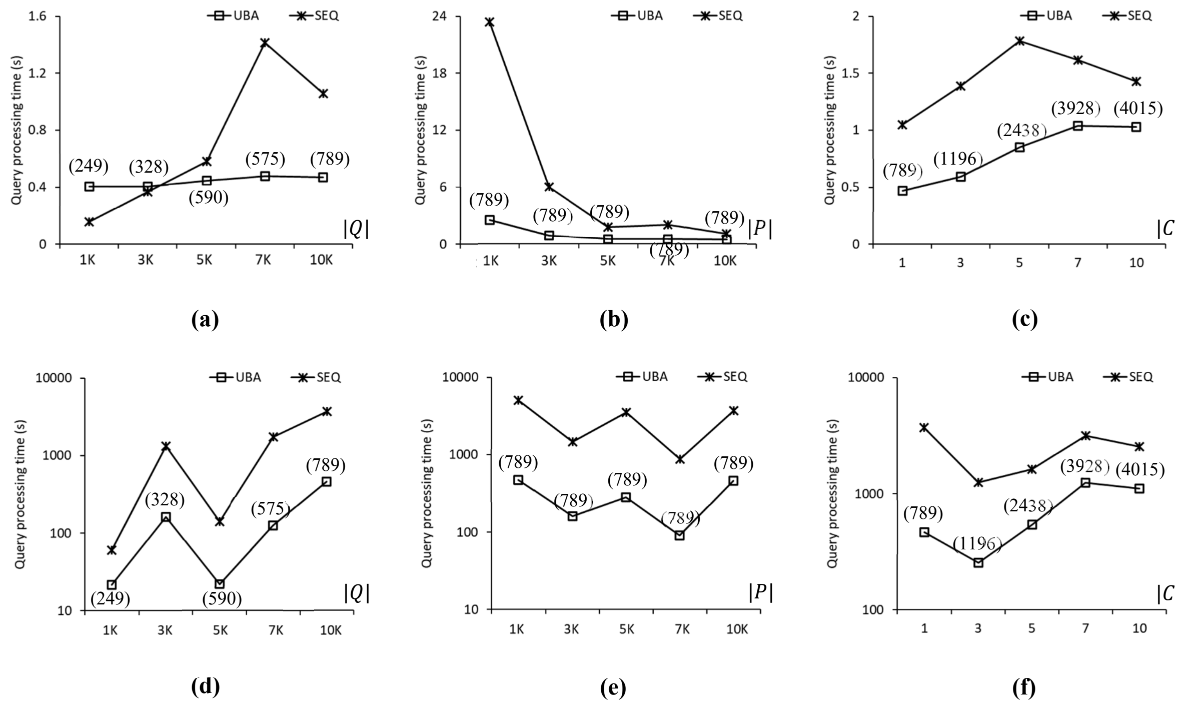

Figure 7. Comparison of query processing times for the CAL roadmap: (**a**) 1 K $\leq |Q| \leq$ 10 K. (**b**) 1 K $\leq |P| \leq$ 10 K. (**c**) $1 \leq |C| \leq 10$. (**d**) 1 K $\leq |Q| \leq$ 10 K. (**e**) 1 K $\leq |P| \leq$ 10 K. (**f**) $1 \leq |C| \leq 10$.

Figure 7d–f show the query processing times of UBA and SEQ when the data points followed a centroid distribution. The query processing times of the proposed UBA were up to 18.95 times lower than those of SEQ for all cases. Unlike the case shown in Figure 7a, the query processing times of UBA and SEQ did not increase with $|Q|$, as shown in Figure 7d, which means that the query processing time was more sensitive to the distribution of data points than $|Q|$ when the data points followed a highly skewed distribution. When $|Q| = 1$ K, 3 K, 5 K, 7 K, and 10 K, the query processing times of UBA were 21.7, 162.8, 21.9, 126.8, and 468.7 s, respectively. As shown in Figure 7d–f, UBA was faster than SEQ in all cases. The difference in query processing times between UBA and SEQ for a centroid distribution of data points was up to several orders of magnitude greater than that for a uniform distribution of data points.

Figure 8 compares the query processing times obtained when using UBA and SEQ to evaluate the SP queries in the FLA roadmap. Figure 8a shows the query processing time as a function of $|Q|$. We found that the proposed UBA was up to 2.2 times faster than

SEQ for $|Q| \geq 3$ K. However, SEQ was 2.7 times faster than UBA for $|Q| = 1$ K because the batch processing of UBA was for a large number rather than a small number of SP queries. Figure 8b shows the query processing time as a function of $|P|$. UBA was 5.5 and 2.2 times faster than SEQ for $|P| = 1$ K and 10 K, respectively, even though UBA and SEQ evaluated 1601 and 10,000 SP queries, respectively, for these two cases. This is because the search space for the NN queries when $|P| = 1$ K was greater than that when $|P| = 10$ K. Figure 8c shows the query processing time as a function of $|C|$, which, for UBA was up to 2.1 times shorter than that of SEQ in all cases. Clearly, the number of query clusters increased with $|C|$, which adversely affected the performance of the proposed UBA. As shown in Figure 8d–f, UBA was up to 11 times faster than SEQ in all cases. The query processing times of both UBA and SEQ fluctuated, which means that the distribution of highly skewed data points affected the NN query processing time significantly. Specifically, as shown in Figure 8d, the query processing time of UBA for $|Q| = 1$ K was 8.9 times longer than that for $|Q| = 3$ K despite the difference in the number of SP queries in Q.

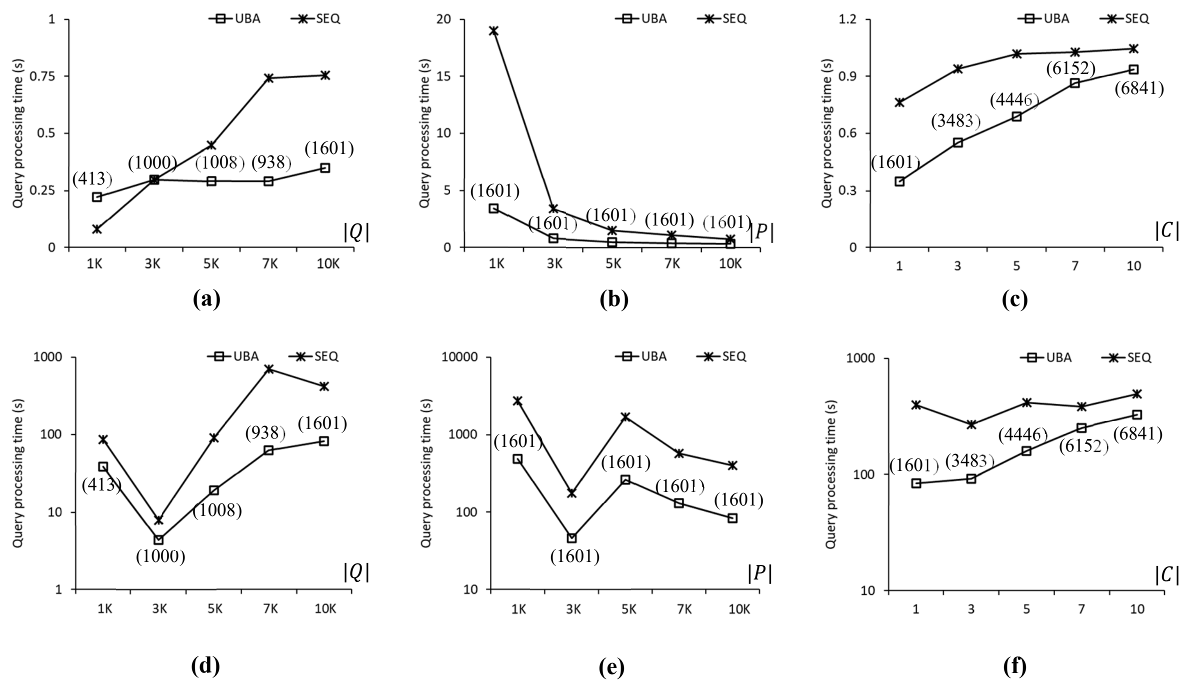

Figure 8. Comparison of query processing times for the FLA roadmap: (**a**) 1 K $\leq |Q| \leq$ 10 K. (**b**) 1 K $\leq |P| \leq$ 10 K. (**c**) $1 \leq |C| \leq 10$. (**d**) 1 K $\leq |Q| \leq$ 10 K. (**e**) 1 K $\leq |P| \leq$ 10 K. (**f**) $1 \leq |C| \leq 10$.

Figure 9 compares the query processing times obtained using UBA and SEQ with the COL roadmap. As shown in Figure 9a, the proposed UBA was up to 3.1 times faster than SEQ when 5K $\leq |Q| \leq$ 10 K. Here, as $|Q|$ increased, UBA was superior to SEQ. As shown in Figure 9b, UBA was up to 16.3 times faster than SEQ regardless of the $|P|$ value because UBA and SEQ evaluated 409 and 10,000 SP queries, respectively. Clearly, this difference in the number of evaluated SP queries (i.e., 9591) occurred the proposed UBA can exploit the batch processing of the clustered SP queries; thus, unnecessary distance computations can be avoided. As shown in Figure 9c, UBA clearly outperformed SEQ in all cases of $|C|$. As $|C|$ increased, the density of the query points decreased, which was ineffective for the batch processing of UBA. As shown in Figure 9d–f, UBA was up to 26.6 times faster than SEQ in all cases. As shown in Figure 9d, the query processing times of UBA and SEQ fluctuated significantly because the highly skewed distributions of data points affected the search space of the NN queries significantly.

Two versions of UBA, i.e., UBA$_{SEG}$ and UBA$_{CLS}$, were implemented and evaluated to investigate the effect of the two-step clustering method on the batch processing of UBA and its scalability in terms of $|Q|$. UBA$_{SEG}$ transforms nearby query points into query segments, and UBA$_{CLS}$ transforms nearby query points into query clusters. UBA$_{SEG}$ and UBA$_{CLS}$ are illustrated in Figure 5a,b, respectively. Figure 10 compares the query processing times using UBA$_{SEG}$ and UBA$_{CLS}$ with the CAL roadmap, where the two values in the parentheses indicate the number of SP queries evaluated by UBA$_{SEG}$ and UBA$_{CLS}$, respectively. As can be seen, the number of SP queries evaluated by UBA$_{SEG}$ was greater than that of UBA$_{CLS}$. As shown in Figure 10a, when the data points exhibited a uniform distribution, UBA$_{SEG}$ was up to 6.1 times faster than UBA$_{CLS}$ for 1 K $\leq |Q| \leq$ 10 K. However, as $|Q|$ increased, UBA$_{CLS}$ was faster than UBA$_{SEG}$, which means that UBA$_{CLS}$ scaled better than UBA$_{SEG}$ with $|Q|$. Specifically, UBA$_{CLS}$ was 1.5 times faster than UBA$_{SEG}$ for $|Q| = 100$ K. As shown in Figure 10b, when the data points exhibited a centroid distribution, UBA$_{CLS}$ was up to 2.2 times faster than UBA$_{SEG}$ in all cases. Therefore, UBA$_{CLS}$ scaled with $|Q|$ better than UBA$_{SEG}$. It is clear that the distribution of data points affected query processing time significantly. Specifically, when the data points exhibited uniform and centroid distributions, the query processing times of UBA$_{CLS}$ were 1.5 and 497.7 s, respectively, for $|Q| = 100$ K.

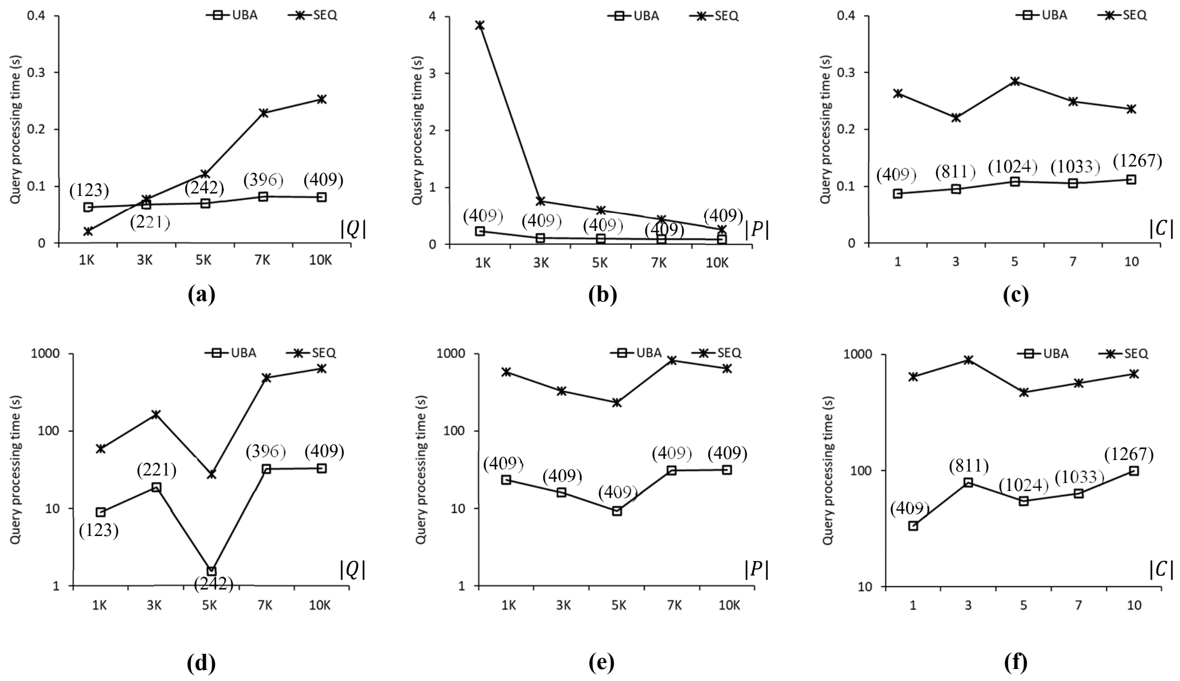

Figure 9. Comparison of query processing times for the COL roadmap: (**a**) 1 K $\leq |Q| \leq$ 10 K. (**b**) 1 K $\leq |P| \leq$ 10 K. (**c**) $1 \leq |C| \leq 10$. (**d**) 1 K $\leq |Q| \leq$ 10 K. (**e**) 1 K $\leq |P| \leq$ 10 K. (**f**) $1 \leq |C| \leq 10$.

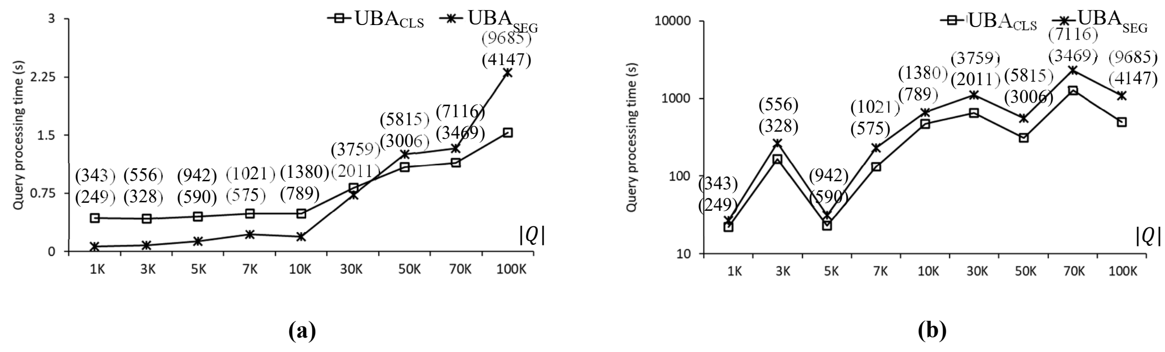

Figure 10. Effect of two-step clustering on the CAL roadmap: (**a**) uniform data points. (**b**) Skewed data points.

6. Conclusions

This paper has proposed the UBA to efficiently process SP queries comprising NN and RN queries in dynamic spatial networks. The goal of the proposed UBA is to avoid dispensable distance computations during batch processing. Accordingly, UBA performs two-step clustering of SP queries and their batch processing to reduce the number of SP queries evaluated for query clusters. The experimental results have confirmed that the proposed UBA outperformed a conventional algorithm based on one-query-at-a-time processing and scaled well with the number of queries. We found that the proposed UBA was up to 26.6 times faster than the compared conventional algorithm. The proposed UBA has several advantages. First, UBA avoids dispensable network traversal by clustering SP queries and performing batch processing. Second, UBA can easily be incorporated into one-query-at-a-time processing algorithms for spatial networks [3,12,13,15]. However, the proposed UBA also exhibits a disadvantage, i.e., its performance is very sensitive to the distribution of query points. Thus, UBA demonstrates similar performance to that of sequential algorithms, particularly when the query points exhibit a uniform distribution. The proposed UBA clearly outperforms sequential algorithms when the query points exhibit a highly skewed distribution. In future, we plan to apply this unified batch solution to extremely large spatial networks for distributed batch processing of sophisticated spatial queries, e.g., spatial join queries [54] and spatial keyword queries [2,50].

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (NRF-2020R1I1A3052713).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mouratidis, K.; Yiu, M.L.; Papadias, D.; Mamoulis, N. Continuous nearest neighbor monitoring in road networks. In Proceedings of the International Conference on Very Large Data Bases, Seoul, Korea, 12–15 September 2006; pp. 43–54.
2. Zheng, B.; Zheng, K.; Xiao, X.; Su, H.; Yin, H.; Zhou, X.; Li, G. Keyword-aware continuous knn query on road networks. In Proceedings of the International Conference on Data Engineering, Helsinki, Finland, 16–20 May 2016; pp. 871–882.
3. Papadias, D.; Zhang, J.; Mamoulis, N.; Tao, Y. Query processing in spatial network databases. In Proceedings of the International Conference on Very Large Data Bases, Berlin, Germany, 9–12 September 2003; pp. 802–813.
4. Taniar, D.; Rahayu, W. A taxonomy for region queries in spatial databases. *J. Comput. Syst. Sci.* **2015**, *81*, 1508–1531. [CrossRef]
5. Zacharatou, E.T.; Sidlauskas, D.; Tauheed, F.; Heinis, T.; Ailamaki, A. Efficient bundled spatial range queries. In Proceedings of the International Conference on Advances in Geographic Information Systems, Chicago, IL, USA, 5–8 November 2019; pp. 139–148.
6. Huang, X.; Jensen, C.S.; Saltenis, S. Multiple k nearest neighbor query processing in spatial network databases. In Proceedings of the European Conference on Advances in Databases and Information Systems, Thessaloniki, Greece, 3–7 September 2006; pp. 266–281.
7. Liu, Y.; Peng, M.; Shou, G. Mobile edge computing-enhanced proximity detection in time-aware road networks. *IEEE Access* **2019**, *7*, 167958–167972. [CrossRef]
8. Miao, X.; Gao, Y.; Mai, G.; Chen, G.; Li, Q. On efficiently monitoring continuous aggregate k nearest neighbors in road networks. *IEEE Trans. Mob. Comput.* **2020**, *19*, 1664–1676. [CrossRef]
9. Ouyang, D.; Wen, D.; Qin, L.; Chang, L.; Zhang, Y.; Lin, X. Progressive top-k nearest neighbors search in large road networks. In Proceedings of the International Conference on Management of Data, Portland, OR, USA, 14–19 June 2020; pp. 1781–1795.
10. Tang, X.; Chai, M.; Chen, X.; Chen, W. Spatio-temporal reachable area calculation based on urban traffic data. *IEEE Syst. J.* **2021**, *15*, 641–652. [CrossRef]
11. Abeywickrama, T.; Cheema, M.A.; Taniar, D. k-Nearest neighbors on road networks: A journey in experimentation and in-memory implementation. *Proc. VLDB Endow.* **2016**, *9*, 492–503. [CrossRef]
12. Lee, K.C.K.; Lee, W.-C.; Zheng, B.; Tian, Y. ROAD: A new spatial object search framework for road networks. *IEEE Trans. Knowl. Data Eng.* **2012**, *24*, 547–560. [CrossRef]
13. Samet, H.; Sankaranarayanan, J.; Alborzi, H. Scalable network distance browsing in spatial databases. In Proceedings of the International Conference on Mobile Data Management, Beijing, China, 27–30 April 2008; pp. 43–54.

14. Shen, B.; Zhao, Y.; Li, G.; Zheng, W.; Qin, Y.; Yuan, B.; Rao, Y. V-tree: Efficient knn search on moving objects with road-network constraints. In Proceedings of the International Conference on Data Engineering, San Diego, CA, USA, 19–22 April 2017; pp. 609–620.
15. Zhong, R.; Li, G.; Tan, K.-L.; Zhou, L.; Gong, Z. G-tree: An efficient and scalable index for spatial search on road networks. *IEEE Trans. Knowl. Data Eng.* **2015**, *27*, 2175–2189. [CrossRef]
16. Uber Revenue and Usage Statistics. Available online: https://www.businessofapps.com/data/uber-statistics/ (accessed on 22 July 2021).
17. Xu, Y.; Qi, J.; Borovica-Gajic, R.; Kulik, L. Finding all nearest neighbors with a single graph traversal. In Proceedings of the International Conference on Database Systems for Advanced Applications, Gold Coast, Australia, 21–24 May 2018; pp. 221–238.
18. Zhang, J.; Mamoulis, N.; Papadias, D.; Tao, Y. All-nearest-neighbors queries in spatial databases. In Proceedings of the International Conference on Scientific and Statistical Database Management, Santorini Island, Greece, 21–23 June 2004; pp. 297–306.
19. Li, L.; Zhang, M.; Hua, W.; Zhou, X. Fast query decomposition for batch shortest path processing in road networks. In Proceedings of the International Conference on Data Engineering, Dallas, TX, USA, 20–24 April 2020; pp. 1189–1200.
20. Wang, Y.; Li, G.; Tang, N. Querying shortest paths on time dependent road networks. *Proc. VLDB Endow.* **2019**, *12*, 1249–1261. [CrossRef]
21. Wei, V.J.; Wong, R.C.-W.; Long, C. Architecture-intact oracle for fastest path and time queries on dynamic road networks. In Proceedings of the International Conference on Management of Data, Portland, OR, USA, 14–19 June 2020; pp. 1841–1856.
22. Dong, T.; Lulu, Y.; Shang, Y.; Ye, Y.; Zhang, L. Direction-aware continuous moving k-nearest-neighbor query in road networks. *ISPRS Int. J. Geo Inf.* **2019**, *8*, 379. [CrossRef]
23. Luo, S.; Kao, B.; Li, G.; Hu, J.; Cheng, R.; Zheng, Y. TOAIN: A throughput optimizing adaptive index for answering dynamic knn queries on road networks. *Proc. VLDB Endow.* **2018**, *11*, 594–606. [CrossRef]
24. Yang, Y.; Li, H.; Wang, J.; Hu, Q.; Wang, X.; Leng, M. A novel index method for k nearest object query over time-dependent road networks. *Complexity* **2019**, *2019*, 4829164. [CrossRef]
25. Abeywickrama, T.; Cheema, M.A. Efficient landmark-based candidate generation for knn queries on road networks. In Proceedings of the International Conference on Database Systems for Advanced Applications, Suzhou, China, 27–30 March 2017; pp. 425–440.
26. Cao, B.; Hou, C.; Li, S.; Fan, J.; Yin, J.; Zheng, B.; Bao, J. SIMkNN: A scalable method for in-memory knn search over moving objects in road networks. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 1957–1970. [CrossRef]
27. Sellis, T.K. Multiple-query optimization. *ACM Trans. Database Syst.* **1988**, *13*, 23–52. [CrossRef]
28. Eslami, M.; Tu, Y.; Charkhgard, H.; Xu, Z.; Liu, J. PsiDB: A framework for batched query processing and optimization. In Proceedings of the International Conference on Big Data, Los Angeles, CA, USA, 9–12 December 2019; pp. 6046–6048.
29. Giannikis, G.; Alonso, G.; Kossmann, D. SharedDB: killing one thousand queries with one stone. *Proc. VLDB Endow.* **2012**, *5*, 526–537. [CrossRef]
30. Giannikis, G.; Makreshanski, D.; Alonso, G.; Kossmann, D. Shared workload optimization. *Proc. VLDB Endow.* **2014**, *7*, 429–440. [CrossRef]
31. Makreshanski, D.; Giannikis, G.; Alonso, G.; Kossmann, D. MQJoin: Efficient shared execution of main-memory joins. *Proc. VLDB Endow.* **2016**, *9*, 480–491. [CrossRef]
32. Makreshanski, D.; Giannikis, G.; Alonso, G.; Kossmann, D. Many-query join: Efficient shared execution of relational joins on modern hardware. *VLDB J.* **2018**, *27*, 669–692. [CrossRef]
33. Marroquin, R.; Müller, I.; Makreshanski, D.; Alonso, G. Pay one, get hundreds for free: Reducing cloud costs through shared query execution. In Proceedings of the Symposium on Cloud Computing, Carlsbad, CA, USA, 11–13 October 2018; pp. 439–450.
34. Michiardi, P.; Carra, D.; Migliorini, S. In-memory caching for multi-query optimization of data-intensive scalable computing workloads. In Proceedings of the Workshops of the EDBT/ICDT Joint Conference, Lisbon, Portugal, 26 March 2019.
35. Psaroudakis, I.; Athanassoulis, M.; Ailamaki, A. Sharing data and work across concurrent analytical queries. *Proc. VLDB Endow.* **2013**, *6*, 637–648. [CrossRef]
36. Rehrmann, R.; Binnig, C.; Böhm, A.; Kim, K.; Lehner, W.; Rizk, A. OLTPShare: The case for sharing in oltp workloads. *Proc. VLDB Endow.* **2018**, *11*, 1769–1780. [CrossRef]
37. Jonathan, A.; Chandra, A.; Weissman, J.B. Multi-query optimization in wide-area streaming analytics. In Proceedings of the Symposium on Cloud Computing, Carlsbad, CA, USA, 11–13 October 2018; pp. 412–425.
38. Karimov, J.; Rabl, T.; Markl, V. AStream: Ad-hoc shared stream processing. In Proceedings of the International Conference on Management of Data, Amsterdam, The Netherlands, 30 June–5 July 2019; pp. 607–622.
39. Karimov, J.; Rabl, T.; Markl, V. AJoin: Ad-hoc stream joins at scale. *Proc. VLDB Endow.* **2019**, *13*, 435–448. [CrossRef]
40. Mahmud, H.; Amin, A.M.; Ali, M.E.; Hashem, T.; Nutanong, S. A group based approach for path queries in road networks. In Proceedings of the International Symposium on Advances in Spatial and Temporal Databases, Munich, Germany, 21–23 August 2013; pp. 367–385.
41. Reza, R.M.; Ali, M.E.; Hashem, T. Group processing of simultaneous shortest path queries in road networks. In Proceedings of the International Conference on Mobile Data Management, Pittsburgh, PA, USA, 15–18 June 2015; pp. 128–133.
42. Zhang, M.; Li, L.; Hua, W.; Zhou, X. Efficient batch processing of shortest path queries in road networks. In Proceedings of the International Conference on Mobile Data Management, Hong Kong, China, 10–13 June 2019; pp. 100–105.

43. Zhang, M.; Li, L.; Hua, W.; Zhou, X. Batch processing of shortest path queries in road networks. In Proceedings of the Australasian Database Conference on Databases Theory and Applications, Sydney, Australia, 29 January–1 February 2019; pp. 3–16.
44. Thomsen, J.R.; Yiu, M.L.; Jensen, C.S. Effective caching of shortest paths for location-based services. In Proceedings of the International Conference on Management of Data, Scottsdale, AZ, USA, 20–24 May 2012; pp. 313–324.
45. Li, H.-J.; Bu, Z.; Wang, Z.; Cao, J. Dynamical clustering in electronic commerce systems via optimization and leadership expansion. *IEEE Trans. Ind. Inform.* **2020**, *16*, 5327–5334. [CrossRef]
46. Li, H.-J.; Zhang, J.; Liu, Z.-P.; Chen, L.; Zhang, X.-S. Identifying overlapping communities in social networks using multi-scale local information expansion. *Eur. Phys. J. B* **2012**, *85*, 190. [CrossRef]
47. Li, H.-J.; Wang, Q.; Liu, S.; Hu, J. Exploring the trust management mechanism in self-organizing complex network based on game theory. *Phys. A Stat. Mech. Appl.* **2020**, *542*, 123514. [CrossRef]
48. Li, H.-J.; Wang, Z.; Pei, J.; Cao, J.; Shi, Y. Optimal estimation of low-rank factors via feature level data fusion of multiplex signal systems. *IEEE Trans. Knowl. Data Eng.* **2020**. [CrossRef]
49. Li, H.-J.; Wang, L.; Zhang, Y.; Perc, M. Optimization of identifiability for efficient community detection. *New J. Phys.* **2020**, *22*, 063035. [CrossRef]
50. Attique, M.; Afzal, M.; Ali, F.; Mehmood, I.; Ijaz, M.F.; Cho, H.-J. Geo-social top-k and skyline keyword queries on road networks. *Sensors* **2020**, *20*, 798. [CrossRef]
51. Cho, H.-J.; Attique, M. Group processing of multiple k-farthest neighbor queries in road networks. *IEEE Access* **2020**, *8*, 110959–110973. [CrossRef]
52. 9th DIMACS Implementation Challenge: Shortest Paths. Available online: http://www.dis.uniroma1.it/challenge9/download.shtml (accessed on 22 July 2021).
53. Wu, L.; Xiao, X.; Deng, D.; Cong, G.; Zhu, A.D.; Zhou, S. Shortest path and distance queries on road networks: An experimental evaluation. *Proc. VLDB Endow.* **2012**, *5*, 406–417. [CrossRef]
54. Corral, A.; Manolopoulos, Y.; Theodoridis, Y.; Vassilakopoulos, M. Multi-way distance join queries in spatial databases. *GeoInformatica* **2004**, *8*, 373–402. [CrossRef]