International Journal of Geo-Information

MDPI

Article

Shared Execution Approach to ε-Distance Join Queries in Dynamic Road Networks

Hyung-Ju Cho

Department of Software, Kyungpook National University, 2559, Gyeongsang-daero, Sangju-si, Gyeongsangbuk-do 37224, Korea; hyungju@knu.ac.kr; Tel.: +82-54-530-1455

Received: 30 April 2018; Accepted: 6 July 2018; Published: 10 July 2018

Abstract: Given a threshold distance ε and two object sets R and S in a road network, an ε-distance join query finds object pairs from $R \times$ S that are within the threshold distance ε (e.g., find passenger and taxicab pairs within a five-minute driving distance). Although this is a well-studied problem in the Euclidean space, little attention has been paid to dynamic road networks where the weights of road segments (e.g., travel times) are frequently updated and the distance between two objects is the length of the shortest path connecting them. In this work, we address the problem of ε-distance join queries in dynamic road networks by proposing an optimized ε-distance join algorithm called EDISON, the key concept of which is to cluster adjacent objects of the same type into a group, and then to optimize shared execution for the group to avoid redundant network traversal. The proposed method is intuitive and easy to implement, thereby allowing its simple integration with existing range query algorithms in road networks. We conduct an extensive experimental study using real-world roadmaps to show the efficiency and scalability of our shared execution approach.

Keywords: ε-distance join query; shared execution; dynamic road network

1. Introduction

Recent advances in mobile technologies and map-based applications enable users to access a wide range of location-based services such as shortest path queries [1–7], distance queries [2,3,5,8,9], range queries [3,10], and k-nearest neighbor (kNN) queries [3,7,10–12]. Owing to the popularity of map-based applications among users, processing spatial queries in road networks efficiently has become an important research area in recent years. In this work, we address ε-distance join queries in road networks. Given a threshold distance ε and two object sets R and S, an ε-distance join query explores all object pairs $(r, s) \in R \times S$ and returns a set of object pairs (r, s) that are within the threshold distance ε. The ε-distance join queries are useful in real-life applications such as data mining and similarity joins [13–18]. Therefore, many studies have evaluated the efficiency of ε-distance join queries in the Euclidean space [19,20] and in the metric space [13–18]. These studies focus primarily on designing elegant indexing techniques to avoid scanning the entire dataset repeatedly, and on pruning as many distance computations as possible.

Figure 1 presents an example of dynamic road networks where objects r_1 through r_5 denote passengers (represented by rectangles), and objects s_1 through s_3 denote taxicabs (represented by triangles). In this example, an ε-distance join query for a taxicab company involves sending an available taxicab to each passenger who is located within a five-minute driving distance from the taxicab. The travel time should be frequently updated depending on traffic conditions. For example, at time t_1, taxicab s_2 is closer to passenger r_4 than taxicab s_3, as shown in Figure 1a. However, as shown in Figure 1b, because the taxicab s_2 is in a traffic-congested area, it cannot reach the passenger r_4 faster than the taxicab s_3 at time t_2.

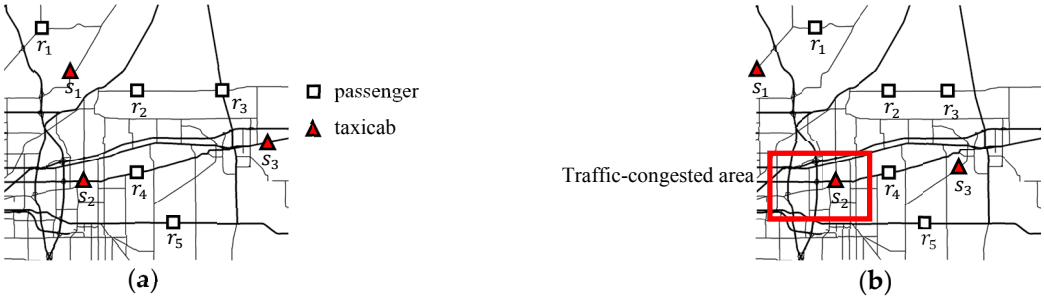

Figure 1. Example of dynamic road networks (**a**) traffic condition at time t_1; (**b**) traffic condition at time t_2.

Owing to the dynamic nature of road networks, many studies have recently addressed a variety of spatial queries in dynamic road networks such as shortest path queries [1,6,21], distance queries [8], and *k*NN queries [22]. Although there have been a few studies [10,23] on ε-distance join queries in road networks, they proposed sophisticated index structures and algorithms to boost ε-distance join queries in static road networks, where the weights of road segments are rarely updated. Specifically, the state-of-the-art solution to ε-distance join queries in road networks is the distance join algorithm proposed in Reference [23], based on the spatially induced linkage cognizance (SILC) framework introduced in Reference [3]. The algorithm pre-computes the shortest path between each pair of vertices in a road network and stores all paths. This approach is not suitable for dynamic road networks because in these road networks, the weights of road segments are frequently updated, and the pre-computed distances between vertices often become obsolete. Various techniques [24–34] for preserving location privacy in road networks have recently been developed, which focus on moving objects such as vehicles and users, and therefore cannot be applied to the problem studied here.

Therefore, in this study, we propose an optimized ε-distance join algorithm called EDISON to efficiently support ε-distance join queries in dynamic road networks, where materialized structures such as the SILC framework cannot be used to dynamically compute the shortest path and distances from a source point to one or more destination points. Intuitively, a simple solution involves computing the distance between every object pair from two datasets. This simple solution is very inefficient, because for each element in an object set R, the algorithm must traverse another object set S, which can lead to redundant network traversal. The proposed solution is to cluster adjacent objects of the same type into a group and then optimize shared execution for the group to avoid redundant network traversal. Although the shared execution of queries has received much attention [21,35–38], no shared execution strategy has been applied to evaluate ε-distance join queries in dynamic road networks to date. The contributions of this study are as follows.

- We propose an efficient ε-distance join algorithm called EDISON for dynamic road networks that optimizes the shared execution paradigm to accelerate query response times. To the best of our knowledge, this is the first attempt to evaluate ε-distance join queries effectively in dynamic road networks.
- We design algorithms that reduce the number of range queries required to evaluate ε-distance join queries in dynamic road networks. The proposed method is intuitive and easy to implement, thereby allowing its simple integration with existing range query algorithms (e.g., [10]) in road networks.
- We conduct extensive experiments under different setups to demonstrate the efficiency and scalability of our approach over conventional solutions.

The remainder of this paper is organized as follows. In Section 2, we review related studies. In Section 3, we provide some background information. We present a simple solution called the

naive EDISON method based on shared execution in Section 4. In Section 5, we present an optimized solution that avoids redundant network traversal called the EDISON method. In Section 6, we evaluate our proposed solutions experimentally under different setups by comparing them to a conventional solution. We conclude the paper in Section 7.

2. Related Work

2.1. Dynamic Road Networks

The geographic information system (GIS) community has shown interest in processing spatial queries in road networks, which form an integral aspect of geospatial applications, such as location-based services and locational analysis [1–3,5–7,10,11,22,23,39,40]. Previous works have studied a variety of spatial queries in road networks, which include shortest path queries [1–7], distance queries [2,3,5,8,9], range queries [3,10], and kNN queries [3,7,10,11]. The previous studies focused primarily on adopting sophisticated index structures to achieve good performance under the assumption that road networks are stable. For example, Sankaranarayanan et al. [3,4] proposed the SILC and path-coherent pairs decomposition frameworks based on spatial path coherence to determine the shortest path and the shortest distance between every pair of vertices. In particular, the SILC framework can support various queries in road networks including path queries, distance queries, range queries, and kNN queries. Because these frameworks have high pre-computational overhead and space complexity even in static road networks, they are not suitable for processing spatial queries in dynamic road networks where the weights of road segments are often updated. Recently, distance queries [8] and shortest path queries [1,6,21] have been actively studied in dynamic road networks. Techniques for distance queries and shortest path queries cannot be applied directly to processing ε-distance join queries in dynamic road networks owing to the inconsistent problem requirements. Recently, Arain et al. [24–26], Domenic et al. [27], Gustav et al. [28], Kamenyi et al. [29], and Memon et al. [30–34] developed novel techniques for location privacy preservation over road networks. These techniques focus on protecting the location privacy of moving objects in road networks. However, it is inappropriate to extend these location privacy techniques to solve the ε-distance join problem in dynamic road networks owing to differences in the problem definition.

2.2. ε-Distance Join Queries

Given a query point q, a threshold distance ε, and a set of objects S, a range query retrieves objects that are within the threshold distance ε from the query point q. Several algorithms have been proposed to support range queries in road networks. Papadias et al. [10] proposed the range Euclidean restriction (RER) and range network expansion (RNE) algorithms for processing range queries in road networks. The ε-distance join query is another important query in road networks. Papadias et al. [10] proposed join Euclidean restriction (JER) and join network expansion (JNE) algorithms to support ε-distance join queries in road networks. The JER algorithm uses the Euclidean distance as a heuristic to search for object pairs within the threshold distance ε based on their network distance. Therefore, it performs poorly when the Euclidean distance and the network distance are very different, which is a common scenario in practice. If the edge weight is defined as the travel time in road networks, the Euclidean distance cannot confine the search space unless additional assumptions are made, such as assuming maximum speed. Therefore, JER is not appropriate for processing ε-distance join queries in dynamic road networks. Sankaranarayanan et al. [23] proposed a distance join algorithm that can support ε-distance join queries in road networks. The distance join algorithm, based on the SILC framework introduced in [3], exploits the pre-computation of the shortest path between each pair of vertices in a road network and stores all paths in a quad-tree. However, the distance join algorithm in [23] is not suitable for dynamic road networks as it suffers from excessive storage costs for large road networks. The ε-distance join queries are well studied in the Euclidean space [19,20] and in the metric space [13–18]. Nonetheless, all approaches are unsuitable for processing ε-distance join queries,

because they utilize some geometric properties (e.g., MBR [20], plane-sweep [20], and space-filling curve [41]) that are not available for dynamic road networks.

As a means to process a large number of queries in database systems, the shared execution of queries has recently received much attention [35–38,42]. The key idea of shared query execution is to cluster similar queries (i.e., those that share some common execution path) into a group and then execute the group as a single query in the system. These shared execution methods are found to be effective in many applications involving high load conditions. In this study, we optimize the shared execution strategy to boost ε-distance join queries in dynamic road networks. Our proposed solution differs from existing studies in several aspects. First, our solution represents the first attempt to evaluate ε-distance join queries efficiently in dynamic road networks, where materialized index structures (e.g., SILC [3]) cannot be used. Second, our solution considers applying a shared execution strategy to avoid redundant network traversal while processing ε-distance join queries. Finally, our solution can be implemented easily using the best-known solutions (e.g., RNE [10]) to range queries in road networks, which is considered a desirable property in practice.

3. Preliminaries

Section 3.1 defines the terms and notations that are used in this paper. Section 3.2 defines the ε-distance join query applied to road networks.

3.1. Definition of Terms and Notations

Road network: We represent a road network by an undirected weighted graph $\langle G = V, E, W \rangle$, where V, E, and W indicate the vertex set, the edge set, and the edge distance matrix, respectively. Each edge $\overline{v_i v_j}$ has a nonnegative weight representing the network distance, such as the travel time.

Classification of vertices: We divide vertices into three categories based on their degree. (1) If the degree of a vertex is larger than or equal to 3, the vertex is referred to as an intersection vertex; (2) If the degree is 2, the vertex is an intermediate vertex; (3) If the degree is 1, the vertex is a terminal vertex.

Vertex sequence and segment: A vertex sequence $\overline{v_l v_{l+1} \dots v_m}$ denotes a path between two vertices v_l and v_m, such that v_l (v_m) is either an intersection vertex or a terminal vertex, and the other vertices in the path, $v_{l+1}, \dots, v_{m-1}$, are intermediate vertices. The length of a vertex sequence is the total weight of the edges in the vertex sequence. A part of a vertex sequence is referred to as a segment. By definition, a vertex sequence is also a segment.

Table 1 summarizes the symbols used in this paper. Note that to simplify presentation, we use $\overline{r_i r_j}$ to denote $\overline{r_i r_{i+1} \cdots r_j}$, where adjacent outer objects $r_i, r_{i+1}, \cdots, r_j$ are located in the same vertex sequence. In this study, we employ the basic concept of clustering adjacent outer objects in a vertex sequence into an outer segment. The advantage of this clustering method is that if we evaluate two range queries at the two end points, r_i and r_j, of an outer segment $\overline{r_i r_j}$, we can retrieve inner objects within distance ε from each outer object $r \in \overline{r_i r_j}$ without duplicating the network traversal. In other words, the two range queries at r_i and r_j are sufficient to retrieve inner objects within distance ε from all outer objects in $\overline{r_i r_j}$, which is proved in Lemma 1.

Figure 2 shows the difference between the distance and the segment length between two objects r and s in a road network, where the numbers at the edges indicate the distance between two adjacent points (e.g., $dist(v_1, v_2) = 6$). The shortest path from r to s is denoted by $r \rightarrow v_2 \rightarrow v_5 \rightarrow s$ where the distance between r and s is $dist(r, s) = 6$ in this case. The segment connecting r and s in the same vertex sequence $\overline{v_2 v_3 v_4 v_5}$ becomes $\overline{r v_3 v_4 s}$ with length equal to $len(r, s) = 9$. We recall that $len(r, s)$ is defined for two objects located in the same vertex sequence.

Table 1. Symbols and their meaning.

Symbol	Definition
ε	Threshold distance
ε_p	Query distance at a point p such that $\varepsilon_p \leq \varepsilon$
r	Outer object $r \in R$
s	Inner object $s \in S$
$dist(p_1, p_2)$	Length of the shortest path connecting two points p_1 and p_2 in the road network
$len(p_1, p_2)$	Length of the segment connecting two points p_1 and p_2, such that both p_1 and p_2 are located in the same vertex sequence
$\overline{v_l v_{l+1} \dots v_m}$	Vertex sequence where v_l and v_m are not intermediate vertices and the other vertices, $v_{l+1}, \dots, v_{m-1}$, are intermediate vertices
$\overline{r_i r_{i+1} \dots r_j}$	Outer segment that consists of outer objects $r_i, r_{i+1}, \dots, r_j$ in a vertex sequence
S_p^ε	Set of inner objects within distance ε from a point p, i.e., $S_p^\varepsilon = \{s \mid dist(p,s) \leq \varepsilon \text{ for } s \in S\}$
$S(\overline{p_1 p_2})$	Set of inner objects in a segment $\overline{p_1 p_2}$, i.e., $S(\overline{p_1 p_2}) = \{s \mid s \in \overline{p_1 p_2} \text{ for } \forall s \in S\}$
$RNQ(\varepsilon, p)$	Range query that returns a set S_p^ε of inner objects within distance ε from a point p
$\Omega(v_x)$	Number of outer segments in vertex sequences adjacent to an intersection vertex v_x

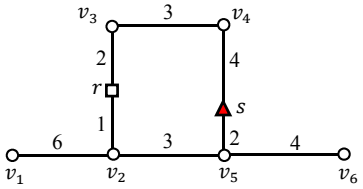

Figure 2. Example where $dist(r,s) = 6$ and $len(r,s) = 9$.

3.2. ε-Distance Join Query

We consider two object sets R and S in the road network G. For convenience, R is referred to as the set of outer objects and S is referred to as the set of inner objects. Accordingly, $r \in R$ and $s \in S$ are referred to as outer and inner objects, respectively.

Definition 1. *(ε-distance join query): Given a threshold distance ε and two object sets R and S, where $R = \left\{r_1, r_2, \cdots, r_{|R|}\right\}$ and $S = \left\{s_1, s_2, \cdots, s_{|S|}\right\}$, the ε-distance join query, denoted by $R \bowtie_\varepsilon S$, returns the set of all possible pairs of objects from $R \times S$ that are within the threshold distance ε:*

$$R \bowtie_\varepsilon S = \{(r,s) \mid dist(r,s) \leq \varepsilon \text{ for } \forall (r,s) \in R \times S\}.$$

Naturally, $R \bowtie_\varepsilon S$ is a subset of $R \times S$. The ε-distance join query is commutative, i.e., $R \bowtie_\varepsilon S = S \bowtie_\varepsilon R$.

4. Naive Shared Execution for ε-Distance Join Queries in Road Networks

4.1. Grouping of Objects in a Vertex Sequence

Figure 3 presents an example of an ε-distance join query in a road network, which will be discussed throughout this section. In this figure, there are five outer objects, r_1 through r_5, and six inner objects, s_1 through s_6. Given a threshold distance $\varepsilon = 5$ and two object sets $R = \{r_1, r_2, r_3, r_4, r_5\}$ and $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$, we consider an ε-distance join query $R \bowtie_\varepsilon S$ of the following form: $R \bowtie_\varepsilon S = \{(r,s) \mid dist(r,s) \leq 5 \text{ for } \forall (r,s) \in R \times S\}$.

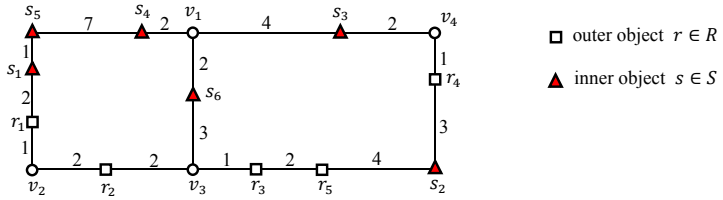

Figure 3. Example of an ε-distance join query in a road network.

Figure 4 shows the sample grouping of outer objects in a vertex sequence and the sample grouping of inner objects in a vertex sequence. As shown in Figure 4a, two outer objects r_1 and r_2 in a vertex sequence $\overline{v_1v_2v_3}$ are grouped into an outer segment $\overline{r_1r_2}$, and three outer objects r_3, r_4, and r_5 in a vertex sequence $\overline{v_1v_4v_3}$ are grouped into another outer segment $\overline{r_3r_5r_4}$. Similarly, as shown in Figure 4b, three inner objects s_1, s_4, and s_5 in a vertex sequence $\overline{v_1v_2v_3}$ are grouped into an inner segment $\overline{s_1s_5s_4}$, and two inner objects s_2 and s_3 in a vertex sequence $\overline{v_1v_4v_3}$ are grouped into another inner segment $\overline{s_2s_3}$. Therefore, a set of outer objects $R = \{r_1, r_2, r_3, r_4, r_5\}$ is transformed into $\overline{R} = \{\overline{r_1r_2}, \overline{r_3r_5r_4}\}$, where $\overline{R}$ is the set of outer segments generated from the outer objects. Similarly, a set of inner objects $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ is transformed into $\overline{S} = \{\overline{s_1s_5s_4}, \overline{s_2s_3}, s_6\}$, where $\overline{S}$ is the set of inner segments generated from the inner objects. Without loss of generality, we assume that $|\overline{R}| \leq |\overline{S}|$, because $R \bowtie_\varepsilon S = S \bowtie_\varepsilon R$. If $|\overline{R}| > |\overline{S}|$, then $S \bowtie_\varepsilon R$ is evaluated instead of $R \bowtie_\varepsilon S$.

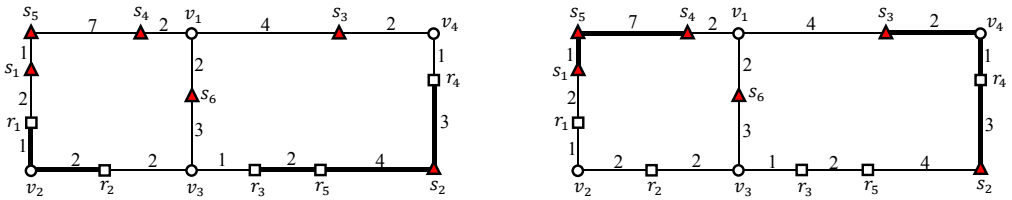

Figure 4. Grouping outer and inner objects (**a**) $\overline{R} = \{\overline{r_1r_2}, \overline{r_3r_5r_4}\}$ and $|\overline{R}| = 2$; (**b**) $\overline{S} = \{\overline{s_1s_5s_4}, \overline{s_2s_3}, s_6\}$ and $|\overline{S}| = 3$.

4.2. Shared Execution Processing of Grouped Outer Objects

Our shared execution strategy for processing ε-distance join queries in road networks is motivated by the observation that at most two range queries are sufficient for retrieving inner objects within the threshold distance ε from each outer object in an outer segment. This observation is formalized in Lemma 1, which states a simple but important fact regarding the shared execution of ε-distance join queries in road networks—if we evaluate two range queries at outer objects r_i and r_j, which correspond to the two end points of an outer segment $\overline{r_ir_{i+1}\ldots r_j}$, then we can retrieve inner objects within distance ε from outer objects $r_{i+1}, \cdots, r_{j-1}$ without duplicating the network traversal. Thus, the two range queries at r_i and r_j are sufficient for retrieving inner objects within distance ε from the other outer objects $r_{i+1}, \cdots, r_{j-1}$.

Lemma 1. *For every outer object $r \in \overline{r_ir_j}$, it holds that $S_r^\varepsilon \subseteq S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon \cup S(\overline{r_ir_j})$, where $S_{r_i}^\varepsilon$ ($S_{r_j}^\varepsilon$) is the set of inner objects within distance ε from an outer object r_i (r_j), and $S(\overline{r_ir_j})$ is the set of inner objects located in the outer segment $\overline{r_ir_j}$ (e.g., $s_2 \in \overline{r_3r_5r_4}$ in Figure 4).*

Proof. We prove Lemma 1 by contradiction. We assume that $S_r^\varepsilon \subseteq S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon \cup S(\overline{r_ir_j})$ does not hold, i.e., $S_r^\varepsilon \nsubseteq S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon \cup S(\overline{r_ir_j})$. Then this implies that there is an inner object $s^+ \in S_r^\varepsilon$ such that $s^+ \notin S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon \cup S(\overline{r_ir_j})$, i.e., $s^+ \notin S_{r_i}^\varepsilon$, $s^+ \notin S_{r_j}^\varepsilon$, and $s^+ \notin S(\overline{r_ir_j})$. Clearly, $s^+ \notin S_{r_i}^\varepsilon$ means that $dist(r_i, s^+) > \varepsilon$. Similarly, $s^+ \notin S_{r_j}^\varepsilon$ means that $dist(r_j, s^+) > \varepsilon$. Clearly, $s^+ \notin S(\overline{r_ir_j})$ means that s^+ is not located in $\overline{r_ir_j}$. Therefore,

the shortest path from r to s^+ passes through either r_i or r_j and the distance from r to s^+ is determined by $dist(r,s^+) = min\{len(r,r_i) + dist(r_i,s^+), len(r,r_j) + dist(r_j,s^+)\}$. From the previous condition, both $dist(r_i,s^+) > \varepsilon$ and $dist(r_j,s^+) > \varepsilon$, which leads to $dist(r,s^+) > \varepsilon$. Thus, s^+ cannot belong to S_r^ε, which contradicts the assumption that an inner object $s^+ \in S_r^\varepsilon$ exists such that $S_r^\varepsilon \nsubseteq S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon \cup S(\overline{r_i r_j})$. Consequently, it holds that $S_r^\varepsilon \subseteq S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon \cup S(\overline{r_i r_j})$ for $\forall r \in \overline{r_i r_j}$. □

We determine the inner objects within distance ε from an outer object $r \in \overline{r_i r_j}$ among inner objects in $S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon \cup S(\overline{r_i r_j})$. First, we compute the distance from an outer object $r \in \overline{r_i r_j}$ to an inner object $s \in S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon \cup S(\overline{r_i r_j})$. If there exists a path $r \to r_i \to s$ (i.e., $s \in S_{r_i}^\varepsilon$), then the distance from r to s is $dist(r,s) = len(r,r_i) + dist(r_i,s)$, as shown in Figure 5a. Similarly, if there exists a path $r \to r_j \to s$ (i.e., $s \in S_{r_j}^\varepsilon$), then the distance from r to s is $dist(r,s) = len(r,r_j) + dist(r_j,s)$, as shown in Figure 5b. If the inner object s is located in $\overline{r_i r_j}$ (i.e., $s \in \overline{r_i r_j}$), then the distance from r to s is $dist(r,s) = len(r,s)$, as shown in Figure 5c. Because $dist(r,s)$ is the length of the shortest path among multiple paths between r and s, $dist(r,s)$ is computed as follows.

$$dist(r,s) = \begin{cases} min\{len(r,r_i) + dist(r_i,s), len(r,r_j) + dist(r_j,s)\} & \text{if } s \notin \overline{r_i r_j} \\ min\{len(r,r_i) + dist(r_i,s), len(r,r_j) + dist(r_j,s), len(r,s)\} & \text{otherwise} \end{cases}$$

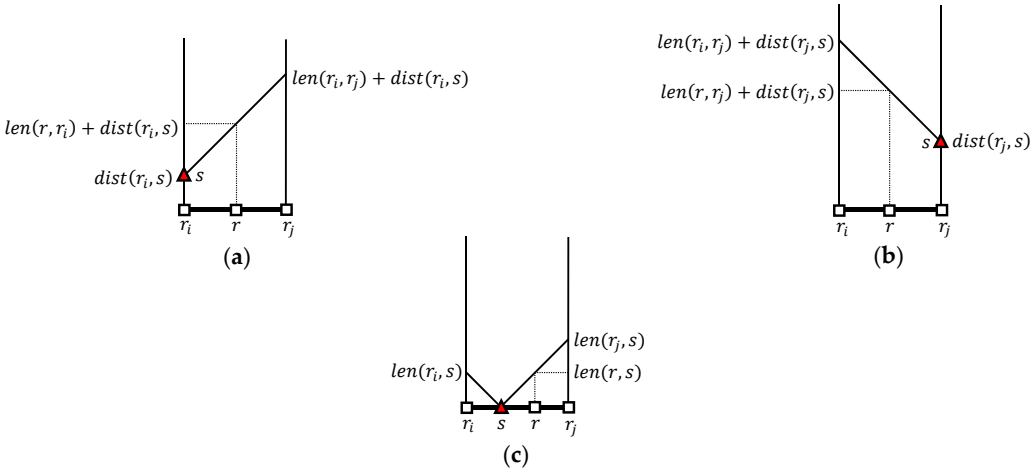

Figure 5. Determination of the distance from r to s (**a**) If $s \in S_{r_i}^\varepsilon$, then $dist(r,s) = len(r,r_i) + dist(r_i,s)$; (**b**) If $s \in S_{r_j}^\varepsilon$, then $dist(r,s) = len\left(r,r_j\right) + dist\left(r_j,s\right)$; (**c**) If $s \in \overline{r_i r_j}$, then $dist(r,s) = len(r,s)$.

Table 2 explains how to compute the distance from r to s, where $r \in \overline{r_i r_j}$ and $s \in S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon \cup S(\overline{r_i r_j})$. The inner object s belongs to a combination of $S_{r_i}^\varepsilon$, $S_{r_j}^\varepsilon$, and $S(\overline{r_i r_j})$, so seven possible cases are considered in total. Clearly, it is trivial to retrieve a set $S(\overline{r_i r_j})$ of inner objects that are located in $\overline{r_i r_j}$ compared with retrieving a set $S_{r_i}^\varepsilon$ ($S_{r_j}^\varepsilon$) of inner objects within distance ε from an outer object r_i (r_j).

Table 2. Computation of $dist(r,s)$ for $r \in \overline{r_i r_j}$ and $s \in S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon \cup S\left(\overline{r_i r_j}\right)$.

Condition	$dist(r,s)$
$s \in S_{r_i}^\varepsilon \cap S_{r_j}^\varepsilon \cap S\left(\overline{r_i r_j}\right)$	$dist(r,s) = min\left\{len(r,r_i) + dist(r_i,s),\ len\left(r,r_j\right) + dist\left(r_j,s\right), len(r,s)\right\}$
$s \in S_{r_i}^\varepsilon \cap S_{r_j}^\varepsilon - S\left(\overline{r_i r_j}\right)$	$dist(r,s) = min\left\{len(r,r_i) + dist(r_i,s),\ len\left(r,r_j\right) + dist\left(r_j,s\right)\right\}$
$s \in S_{r_i}^\varepsilon \cap S\left(\overline{r_i r_j}\right) - S_{r_j}^\varepsilon$	$dist(r,s) = min\{len(r,r_i) + dist(r_i,s), len(r,s)\}$
$s \in S_{r_j}^\varepsilon \cap S\left(\overline{r_i r_j}\right) - S_{r_i}^\varepsilon$	$dist(r,s) = min\left\{len\left(r,r_j\right) + dist\left(r_j,s\right), len(r,s)\right\}$
$s \in S_{r_i}^\varepsilon - \left(S_{r_j}^\varepsilon \cup S\left(\overline{r_i r_j}\right)\right)$	$dist(r,s) = len(r,r_i) + dist(r_i,s)$
$s \in S_{r_j}^\varepsilon - \left(S_{r_i}^\varepsilon \cup S\left(\overline{r_i r_j}\right)\right)$	$dist(r,s) = len\left(r,r_j\right) + dist\left(r_j,s\right)$
$s \in S\left(\overline{r_i r_j}\right) - \left(S_{r_i}^\varepsilon \cup S_{r_j}^\varepsilon\right)$	$dist(r,s) = len(r,s)$

4.3. Naive EDISON Algorithm

Algorithm 1 describes the naive EDISON algorithm, which employs the grouping of outer objects and shared execution to reduce query processing time. This algorithm involves two steps. In the first step, adjacent outer objects in a vertex sequence are grouped into an outer segment. Similarly, adjacent inner objects in a vertex sequence are also grouped into an inner segment. In other words, R and S are transformed into $\overline{R}$ and $\overline{S}$, respectively. For simplicity, we assume that $|\overline{R}| \leq |\overline{S}|$. If $|\overline{R}| > |\overline{S}|$, we simply evaluate $S \bowtie_\varepsilon R$, because $R \bowtie_\varepsilon S = S \bowtie_\varepsilon R$. In the second step, the naive EDISON algorithm explores each outer segment $\overline{r_i r_j}$ sequentially to retrieve inner objects that are within distance ε from each outer object in the outer segment $\overline{r_i r_j}$. Depending on the number of outer objects in $\overline{r_i r_j}$, there are three possible cases, which the naive EDISON algorithm handles differently: (1) $|\overline{r_i r_j}| = 1$; (2) $|\overline{r_i r_j}| = 2$; and (3) $|\overline{r_i r_j}| > 2$. If $|\overline{r_i r_j}| = 1$, a range query $RNQ(\varepsilon, r_i)$ is evaluated at r_i because $\overline{r_i r_j}$ includes an outer object r_i ($= r_j$) only. Then, a partial join result $\Phi(r_i)$ is simply obtained from the range query result $S^\varepsilon_{r_i}$ at r_i, i.e., $\Phi(r_i) = \left\{(r_i, s) \middle| s \in S^\varepsilon_{r_i}\right\}$. The partial join result $\Phi(r_i)$ is added to the ε-distance join query result $\Phi(R)$, i.e., $\Phi(R) = \Phi(R) \cup \Phi(r_i)$ (lines 8-10). If $|\overline{r_i r_j}| = 2$, two range queries, $RNQ(\varepsilon, r_i)$ and $RNQ(\varepsilon, r_j)$, are evaluated at r_i and r_j, respectively, because $\overline{r_i r_j}$ includes only two outer objects r_i and r_j. Then, two partial join results $\Phi(r_i)$ and $\Phi(r_j)$ are simply obtained from the range query results $S^\varepsilon_{r_i}$ and $S^\varepsilon_{r_j}$ at r_i and r_j, respectively, i.e., $\Phi(r_i) = \left\{(r_i, s) \middle| s \in S^\varepsilon_{r_i}\right\}$ and $\Phi(r_j) = \left\{(r_j, s) \middle| s \in S^\varepsilon_{r_j}\right\}$. These partial join results $\Phi(r_i)$ and $\Phi(r_j)$ are added to the ε-distance join query result $\Phi(R)$, i.e., $\Phi(R) = \Phi(R) \cup \Phi(r_i) \cup \Phi(r_j)$ (lines 11-14). Finally, if $|\overline{r_i r_j}| > 2$, only two range queries, $RNQ(\varepsilon, r_i)$ and $RNQ(\varepsilon, r_j)$, are evaluated at r_i and r_j, respectively, similar to the case of $|\overline{r_i r_j}| = 2$. According to Lemma 1, a partial join result $\Phi(\overline{r_i r_j})$ can be obtained from $S^\varepsilon_{r_i} \cup S^\varepsilon_{r_j} \cup S(\overline{r_i r_j})$ using the shared execution method (lines 15-19), which is detailed in Lemma 2. This is because the partial join result $\Phi(\overline{r_i r_j})$ is the set of object pairs (r, s) that are within distance ε, where $r \in \overline{r_i r_j}$ and $s \in S^\varepsilon_{r_i} \cup S^\varepsilon_{r_j} \cup S(\overline{r_i r_j})$, i.e., $\Phi(\overline{r_i r_j}) = \left\{(r, s) \middle| dist(r, s) \leq \varepsilon \text{ for } r \in \overline{r_i r_j} \text{ and } s \in S^\varepsilon_{r_i} \cup S^\varepsilon_{r_j} \cup S(\overline{r_i r_j})\right\}$. Finally, the ε-distance join query result $\Phi(R)$ is returned after all outer segments have been processed (line 20), i.e., $\Phi(R) = \bigcup \Phi(\overline{r_i r_j})$ for each outer segment $\overline{r_i r_j} \in \overline{R}$. In Lemma 2, we prove the correctness of the naive EDISON algorithm.

Algorithm 1: Naive_EDISON (ε, R, S)

Input: ε: threshold distance, R: set of outer objects, S: set of inner objects
Output: $R \bowtie_\varepsilon S$: set of object pairs $(r, s) \in R \times S$ such that $dist(r, s) \leq \varepsilon$

1 $\Phi(R) \leftarrow \varnothing$ // $\Phi(R)$ is the set of object pairs $(r, s) \in R \times S$ such that $dist(r, s) \leq \varepsilon$.
2 // Step 1: neighboring outer objects are grouped and neighboring inner objects are also grouped.
3 $\overline{R} \leftarrow group\_objects(R)$ // Outer objects in a vertex sequence are grouped into an outer segment.
4 $\overline{S} \leftarrow group\_objects(S)$ // Inner objects in a vertex sequence are grouped into an inner segment.
5 // For simplicity, we assume that $|\overline{R}| \leq |\overline{S}|$. if $|\overline{R}| > |\overline{S}|$, we simply evaluate $S \bowtie_\varepsilon R$.
6 // Step 2: $\overline{r_i r_j} \bowtie_\varepsilon S$ is evaluated for each outer segment $\overline{r_i r_j} \in \overline{R}$.
7 **for** each outer segment $\overline{r_i r_j} \in \overline{R}$ **do**
8 **if** $\left|\overline{r_i r_j}\right| = 1$ **then** // $\left|\overline{r_i r_j}\right| = 1$ means that $\overline{r_i r_j}$ contains an outer object r_i only.
9 $S^\varepsilon_{r_i} \leftarrow RNQ(\varepsilon, r_i)$ // A range query is evaluated at r_i and its result is $S^\varepsilon_{r_i}$.
10 $\Phi(R) \leftarrow \Phi(R) \cup \Phi(r_i)$ // Note that $\Phi(r_i) = \{(r_i, s) | s \in S^\varepsilon_{r_i}\}$.
11 **else if** $\left|\overline{r_i r_j}\right| = 2$ **then** // $\left|\overline{r_i r_j}\right| = 2$ means that $\overline{r_i r_j}$ contains two outer objects r_i and r_j.
12 $S^\varepsilon_{r_i} \leftarrow RNQ(\varepsilon, r_i)$ // A range query is evaluated at r_i and its result is $S^\varepsilon_{r_i}$.
13 $S^\varepsilon_{r_j} \leftarrow RNQ\left(\varepsilon, r_j\right)$ // Another range query is evaluated at r_j and its result is $S^\varepsilon_{r_j}$.
14 $\Phi(R) \leftarrow \Phi(R) \cup \Phi(r_i) \cup \Phi\left(r_j\right)$ // Note that $\Phi(r_i) = \{(r_i, s) | s \in S^\varepsilon_{r_i}\}$ and $\Phi\left(r_j\right) = \left\{\left(r_j, s\right) \middle| s \in S^\varepsilon_{r_j}\right\}$.
15 **else if** $\left|\overline{r_i r_j}\right| > 2$ **then** // $\left|\overline{r_i r_j}\right| \geq 2$ means that $\overline{r_i r_j}$ contains more than two outer objects.
16 $S^\varepsilon_{r_i} \leftarrow RNQ(\varepsilon, r_i)$ // A range query is evaluated at r_i and its result is $S^\varepsilon_{r_i}$.
17 $S^\varepsilon_{r_j} \leftarrow RNQ\left(\varepsilon, r_j\right)$ // Another range query is evaluated at r_j and its result is $S^\varepsilon_{r_j}$.
18 $\Phi\left(\overline{r_i r_j}\right) \leftarrow DJQ\left(\varepsilon, \overline{r_i r_j}, S^\varepsilon_{r_i} \cup S^\varepsilon_{r_j} \cup S\left(\overline{r_i r_j}\right)\right)$ // DJQ is explained in Algorithm 3.
19 $\Phi(R) \leftarrow \Phi(R) \cup \Phi\left(\overline{r_i r_j}\right)$ // $\Phi\left(\overline{r_i r_j}\right) = \left\{(r, s) \middle| dist(r, s) \leq \varepsilon \text{ for } r \in \overline{r_i r_j},\ s \in S^\varepsilon_{r_i} \cup S^\varepsilon_{r_j} \cup S\left(\overline{r_i r_j}\right)\right\}$.
20 **return** $\Phi(R)$ // $\Phi(R)$ stores the result of $R \bowtie_\varepsilon S$.

Lemma 2. *The naive EDISON algorithm is correct.*

Proof. We prove the correctness of the naive EDISON algorithm by cases. As shown in Algorithm 1, the naive EDISON algorithm handles the following three cases differently depending on the number of outer objects in an outer segment $\overline{r_i r_j}$, namely, (1) $|\overline{r_i r_j}| = 1$, (2) $|\overline{r_i r_j}| = 2$, and (3) $|\overline{r_i r_j}| > 2$. In the first two cases (i.e., $|\overline{r_i r_j}| = 1$ and $|\overline{r_i r_j}| = 2$), a range query is used to retrieve inner objects s within distance ε from each outer object $r \in \overline{r_i r_j}$. This is same as a straightforward method used to answer ε-distance join queries by issuing a range query for each outer object $r \in R$. For $|\overline{r_i r_j}| = 1$, the naive EDISON algorithm evaluates a range query $RNQ(\varepsilon, r_i)$ for only an outer object r_i, and adds an object pair (r_i, s) to the ε-distance join query result for each inner object $s \in S^{\varepsilon}_{r_i}$. Therefore, the naive EDISON algorithm is correct for $|\overline{r_i r_j}| = 1$ because $RNQ(\varepsilon, r_i)$ simply retrieves inner objects s within distance ε from r_i. Similarly, for $|\overline{r_i r_j}| = 2$, the naive EDISON algorithm evaluates two range queries, $RNQ(\varepsilon, r_i)$ and $RNQ(\varepsilon, r_j)$, for two outer objects r_i and r_j, respectively. The naive EDISON algorithm then adds an object pair (r_i, s) to the ε-distance join query result for each inner object $s \in S^{\varepsilon}_{r_i}$, and adds an object pair (r_j, s) to the ε-distance join query result for each inner object $s \in S^{\varepsilon}_{r_j}$. Therefore, the naive EDISON algorithm is correct for $|\overline{r_i r_j}| = 2$ because $RNQ(\varepsilon, r_i)$ and $RNQ(\varepsilon, r_j)$ retrieve inner objects s within distance ε from r_i and r_j, respectively.

The proof for the correctness of the naive EDISON algorithm for $|\overline{r_i r_j}| > 2$ differs from that for $|\overline{r_i r_j}| = 1$ and $|\overline{r_i r_j}| = 2$ because the naive EDISON algorithm exploits the shared execution strategy for $|\overline{r_i r_j}| > 2$. Note that the naive EDISON algorithm evaluates only two range queries $RNQ(\varepsilon, r_i)$ and $RNQ(\varepsilon, r_j)$ for $|\overline{r_i r_j}| > 2$. The key idea of the proof of the correctness for $|\overline{r_i r_j}| > 2$ is to compute the distance between an outer object r and each candidate inner object s without evaluating a range query for r, where $r \in \overline{r_i r_j} - \{r_i, r_j\}$ and $s \in S^{\varepsilon}_{r_i} \cup S^{\varepsilon}_{r_j} \cup S(\overline{r_i r_j})$. According to Lemma 1, we can retrieve inner objects within distance ε from every outer object $r \in \overline{r_i r_j}$ among the candidate inner objects $s \in S^{\varepsilon}_{r_i} \cup S^{\varepsilon}_{r_j} \cup S(\overline{r_i r_j})$. The distance between r and s, $dist(r, s)$, determines whether an object pair (r, s) is included in the ε-distance join query result. Let us compute the distance between an outer object r and candidate inner object s. To do so, we consider the two cases, $s \notin \overline{r_i r_j}$ and $s \in \overline{r_i r_j}$, as shown in Figure 6a,b, respectively. If $s \notin \overline{r_i r_j}$, the distance from r to s is evaluated as $dist(r, s) = min\{len(r, r_i) + dist(r_i, s), len(r, r_j) + dist(r_j, s)\}$ because there are two possible paths from r to s, i.e., $r \to r_i \to s$ and $r \to r_j \to s$; otherwise, the distance from r to s is evaluated as $dist(r, s) = min\{len(r, r_i) + dist(r_i, s), len(r, r_j) + dist(r_j, s), len(r, s)\}$ because there are three possible paths from r to s, i.e., $r \to r_i \to s$, $r \to r_j \to s$, and $r \to s$. If $dist(r, s) \leq \varepsilon$, an object pair (r, s) is included in the ε-distance join query result; otherwise, the object pair (r, s) is not included. Therefore, the naive EDISON algorithm is correct for $|\overline{r_i r_j}| > 2$. Consequently, the naive EDISON algorithm is correct for $|\overline{r_i r_j}| = 1$, $|\overline{r_i r_j}| = 2$, and $|\overline{r_i r_j}| > 2$. □

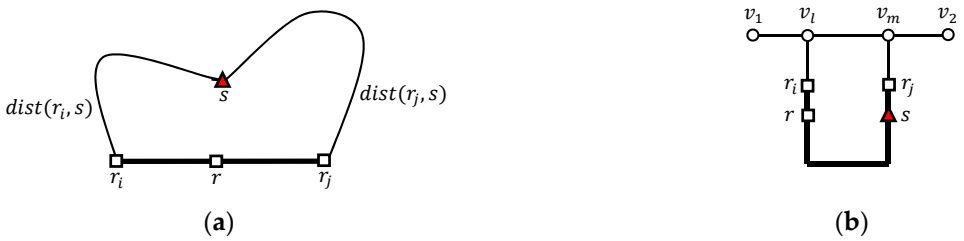

Figure 6. Computation of the distance from $r \in \overline{r_i r_j}$ to s (**a**) If $s \notin \overline{r_i r_j}$, $dist(r, s) = min\left\{len(r, r_i) + dist(r_i, s), len\left(r, r_j\right) + dist\left(r_j, s\right)\right\}$; (**b**) If $s \in \overline{r_i r_j}$, $dist(r, s) = min\left\{len(r, r_i) + dist(r_i, s), len\left(r, r_j\right) + dist\left(r_j, s\right), len(r, s)\right\}$.

4.4. Evaluation of an Example ε-Distance Join Query Using the Naive EDISON Algorithm

We discuss how to evaluate the ε-distance join query in Figure 3 using the naive EDISON algorithm. Recall that $\varepsilon = 5$, $R = \{r_1, r_2, r_3, r_4, r_5\}$, and $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ are given and that outer objects r_1 through r_5 are grouped into outer segments $\overline{r_1 r_2}$ and $\overline{r_3 r_5 r_4}$, as shown in Figure 4. Table 3 summarizes the computation of $R \bowtie_\varepsilon S$ for the naive EDISON algorithm.

Table 3. Computation of $R \bowtie_\varepsilon S$ using the naive EDISON algorithm.

$\overline{r_i r_j}$	r_i	r_j	S''_{r_i}	S''_{r_j}	$S(\overline{r_i r_j})$	$\{r_{i+1}, \ldots, r_{j-1}\}$	$S''_{r_i} \cup S''_{r_j} \cup S(\overline{r_i r_j})$
$\overline{r_1 r_2}$	r_1	r_2	$S^\varepsilon_{r_1} = \{s_1, s_5\}$	$S^\varepsilon_{r_2} = \{s_1, s_6\}$	$S(\overline{r_1 r_2}) = \varnothing$	$\varnothing$	$\{s_1, s_5, s_6\}$
$\overline{r_3 r_5 r_4}$	r_3	r_4	$S^\varepsilon_{r_3} = \{s_6\}$	$S^\varepsilon_{r_4} = \{s_2, s_3\}$	$S(\overline{r_3 r_5 r_4}) = \{s_2\}$	$\{r_5\}$	$\{s_2, s_3, s_6\}$

Because Algorithm 1 processes outer segments one by one, we determine inner objects within the threshold distance ε from each outer object $r \in \overline{r_1 r_2}$ followed by $\overline{r_3 r_5 r_4}$. Two range queries $RNQ(5, r_1)$ and $RNQ(5, r_2)$ are evaluated for an outer segment $\overline{r_1 r_2}$, whose results are $S^\varepsilon_{r_1} = \{s_1, s_5\}$ and $S^\varepsilon_{r_2} = \{s_1, s_6\}$, respectively. Because $|\overline{r_1 r_2}| = 2$ holds, we can simply obtain the partial join result $\Phi(\overline{r_1 r_2})$ for $\overline{r_1 r_2}$, i.e., $\Phi(\overline{r_1 r_2}) = \Phi(r_1) \cup \Phi(r_2) = \{(r_1, s_1), (r_1, s_5), (r_2, s_1), (r_2, s_6)\}$.

Similarly, two range queries $RNQ(5, r_3)$ and $RNQ(5, r_4)$ are evaluated for an outer segment $\overline{r_3 r_5 r_4}$, whose results are $S^\varepsilon_{r_3} = \{s_6\}$ and $S^\varepsilon_{r_4} = \{s_2, s_3\}$, respectively. Because we have $S^\varepsilon_{r_3} = \{s_6\}$, $S^\varepsilon_{r_4} = \{s_2, s_3\}$, and $S(\overline{r_3 r_5 r_4}) = \{s_2\}$, we can compute the distance ε from $r_5 \in \overline{r_3 r_5 r_4}$ to each candidate inner object $s \in S^\varepsilon_{r_3} \cup S^\varepsilon_{r_4} \cup S(\overline{r_3 r_5 r_4})$ and can retrieve inner objects within distance ε from r_5. For this, we need to compute the distance from r_5 to each candidate inner object $s \in \{s_2, s_3, s_6\}$. Because $s_2 \in S^\varepsilon_{r_4} \cap S(\overline{r_3 r_5 r_4}) - S^\varepsilon_{r_3}$ according to Table 3, the distance from r_5 to s_2 is $dist(r_5, s_2) = min\{len(r_5, r_4) + dist(r_4, s_2), len(r_5, s_2)\} = \min\{10, 4\} = 4$, as shown in Figure 7a. Similarly, because $s_3 \in S^\varepsilon_{r_4} - \left(S^\varepsilon_{r_3} \cup S(\overline{r_3 r_5 r_4})\right)$ according to Table 3, the distance from r_5 to s_3 is $dist(r_5, s_3) = len(r_5, r_4) + dist(r_4, s_3) = 10$, as shown in Figure 7b. Finally, because $s_6 \in S^\varepsilon_{r_3} - \left(S^\varepsilon_{r_4} \cup S(\overline{r_1 r_5 r_4})\right)$ according to Table 3, the distance from r_5 to s_6 is $dist(r_5, s_6) = len(r_5, r_3) + dist(r_3, s_6) = 6$, as shown in Figure 7c. Thus, $S^\varepsilon_{r_5} = \{s_2\}$ given that $dist(r_5, s_2) = 4$, $dist(r_5, s_3) = 10$, and $dist(r_5, s_6) = 6$. The partial join result $\Phi(\overline{r_3 r_5 r_4})$ for $\overline{r_3 r_5 r_4}$ is obtained by taking the union of the three range query results at r_3, r_4, and r_5, i.e., $\Phi(\overline{r_3 r_5 r_4}) = \Phi(r_3) \cup \Phi(r_4) \cup \Phi(r_5) = \{(r_3, s_6), (r_4, s_2), (r_4, s_3), (r_5, s_2)\}$ where $\Phi(r_3) = \{(r_3, s_6)\}$, $\Phi(r_4) = \{(r_4, s_2), (r_4, s_3)\}$, and $\Phi(r_5) = \{(r_5, s_2)\}$. Consequently, we have the result $\Phi(R)$ of the ε-distance join query, i.e., $\Phi(R) = \Phi(\overline{r_1 r_2}) \cup \Phi(\overline{r_3 r_5 r_4}) = \{(r_1, s_1), (r_1, s_5), (r_2, s_1), (r_2, s_6), (r_3, s_6), (r_4, s_2), (r_4, s_3), (r_5, s_2)\}$.

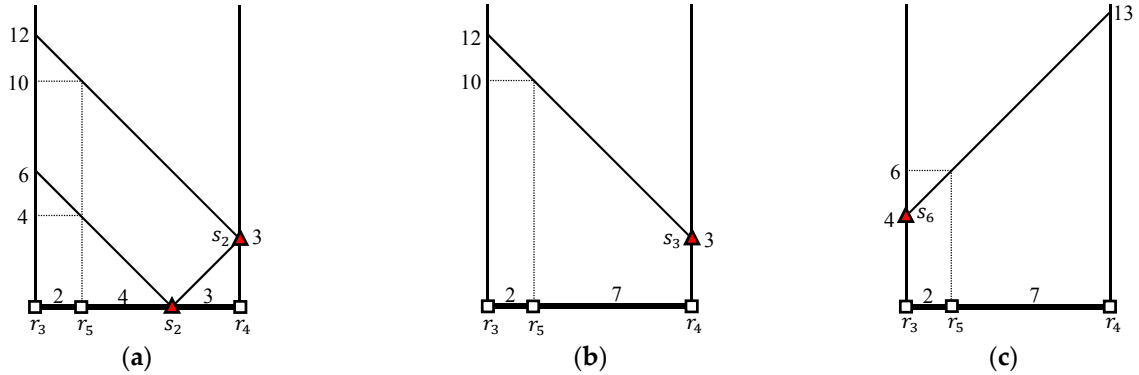

Figure 7. Computation of the distance from r_5 to $s \in \{s_2, s_3, s_6\}$ (**a**) $dist(r_5, s_2) = 4$; (**b**) $dist(r_5, s_3) = 10$; (**c**) $dist(r_5, s_6) = 6$.

5. Optimal Shared Execution for ε-Distance Join Queries in Road Networks

5.1. Extending Shared Execution Processing to Adjacent Outer Segments

We can extend the shared execution processing to adjacent outer segments. We call this shared execution processing the EDISON method. To this end, we investigate the number of outer segments that belong to adjacent vertex sequences of an intersection vertex. Let $\Omega(v_x)$ be the number of outer segments in vertex sequences adjacent to an intersection vertex v_x. If $\Omega(v_x) = 0$, as shown in Figure 8a, no range query is issued at v_x. If $\Omega(v_x) = 1$, as shown in Figure 8b, a range query $RNQ(\varepsilon, r_i)$ is issued at r_i. If $\Omega(v_x) \geq 2$, as shown in Figure 8c, a range query $RNQ(\varepsilon_{v_x}, v_x)$ is issued at v_x, where $\Omega(v_x) = n$ and $\varepsilon_{v_x} = \varepsilon - min\{len(v_x, r_{i_1}), len(v_x, r_{i_2}), \cdots, len(v_x, r_{i_n})\}$. Naturally, if $\varepsilon_{v_x} \leq 0$, no range query at v_x is issued. In summary, if $\Omega(v_x) = 1$, the same shared execution as the naive EDISON algorithm is applied to an outer segment, and if $\Omega(v_x) \geq 2$, an extended shared execution is applied to outer segments adjacent to v_x.

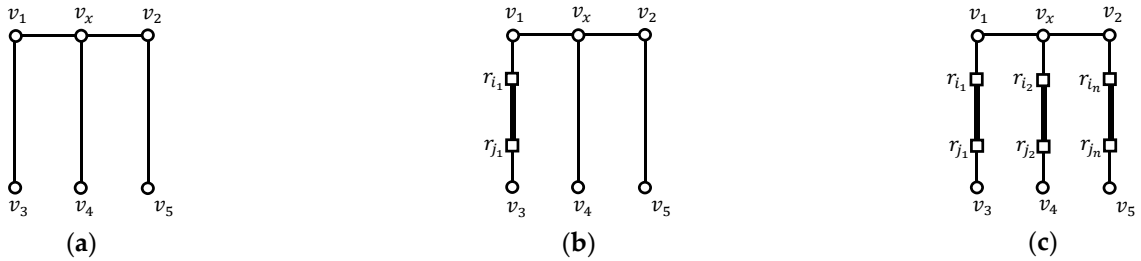

Figure 8. Investigating the number of outer segments adjacent to an intersection vertex v_x (**a**) $\Omega(v_x) = 0$; (**b**) $\Omega(v_x) = 1$; (**c**) $\Omega(v_x) \geq 2$.

Returning to the example in Figure 3, we reevaluate the ε-distance join query to demonstrate the effectiveness of considering the shared execution of adjacent outer segments. To answer the ε-distance join query, the naive EDISON method would evaluate four range queries $RNQ(5, r_1)$, $RNQ(5, r_2)$, $RNQ(5, r_3)$, and $RNQ(5, r_4)$ whereas the EDISON method only evaluates the range query $RNQ(4, v_3)$ at the intersection vertex v_3. This occurs because there are two intersection vertices v_1 and v_3, as shown in Figure 4a and we have $\Omega(v_1) = 2$, $\Omega(v_3) = 2$, $\varepsilon_{v_1} = \varepsilon - min\{len(v_1, r_1), len(v_1, r_4)\} = 5 - min\{12, 7\} = -2$, so $\varepsilon_{v_3} = \varepsilon - min\{len(v_3, r_2), len(v_3, r_3)\} = 5 - min\{2, 1\} = 4$.

We present a simple heuristic where no range queries close to terminal vertices are issued. As shown in Figure 9, the example network has one intersection vertex v_x and three terminal vertices v_3, v_4, and v_5. In this figure, the naive EDISON method evaluates two range queries $RNQ(\varepsilon, r_i)$ and $RNQ(\varepsilon, r_j)$ to answer the ε-distance join query. However, the EDISON method evaluates only the range query $RNQ(\varepsilon, r_i)$ to answer the same ε-distance join query. This is because $S^{\varepsilon}_{r_i} \cup S^{\varepsilon}_{r_j} \cup S(\overline{r_i r_j}) = S^{\varepsilon}_{r_i} \cup S(\overline{r_i r_j v_3})$ holds, and it is sufficient to evaluate the range query $RNQ(\varepsilon, r_i)$ rather than the two range queries $RNQ(\varepsilon, r_i)$ and $RNQ(\varepsilon, r_j)$.

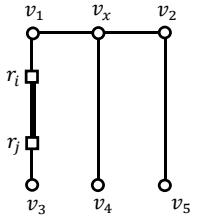

Figure 9. Simple heuristic $S^{\varepsilon}_{r_i} \cup S^{\varepsilon}_{r_j} \cup S(\overline{r_i r_j}) = S^{\varepsilon}_{r_i} \cup S(\overline{r_i v_3})$.

5.2. EDISON Algorithm

Algorithm 2 describes the EDISON algorithm based on shared execution to avoid redundant range queries while processing ε-distance join queries. For simplicity, we assume that $|\overline{R}| \leq |\overline{S}|$, because $R \bowtie_\varepsilon S = S \bowtie_\varepsilon R$. The EDISON algorithm examines $\Omega(v_x)$, the number of outer segments adjacent to each intersection vertex v_x, and applies the extended shared execution to outer segments adjacent to v_x if $\Omega(v_x) \geq 2$. If $\Omega(v_x) < 2$, then no range query is issued at v_x. If $\Omega(v_x) \geq 2$ and $\varepsilon_{v_x} > 0$, then a range query $RNQ(\varepsilon_{v_x}, v_x)$ is issued and its result is saved to $S_{v_x}^{\varepsilon_{v_x}}$, where $\varepsilon_{v_x} = \varepsilon - min\{len(v_x, r_{i_1}), len(v_x, r_{i_2}), \cdots, len(v_x, r_{i_n})\}$, assuming that outer segments $\overline{r_{i_1} r_{j_1}}, \overline{r_{i_2} r_{j_2}}, \cdots, \overline{r_{i_n} r_{j_n}}$ are adjacent to v_x and that r_{ik} is closer to v_x than r_{jk} for $1 \leq k \leq n$ (lines 8-11).

Next, for each outer segment $\overline{r_i r_j} \in \overline{R}$, we retrieve a set of object pairs (r, s) within distance ε from each outer object $r \in \overline{r_i r_j}$. We assume that r_i (r_j) is close to v_l (v_m) for an outer segment $\overline{r_i r_j} \in \overline{v_l v_m}$. We consider the following four cases depending on the values of $\Omega(v_l)$ and $\Omega(v_m)$, where $\overline{v_l v_m}$ is a vertex sequence containing an outer segment $\overline{r_i r_j}$: (1) $\Omega(v_l) = \Omega(v_m) = 1$; (2) $\Omega(v_l) = 1$ and $\Omega(v_m) \geq 2$; (3) $\Omega(v_l) \geq 2$ and $\Omega(v_m) = 1$; and (4) $\Omega(v_l) \geq 2$ and $\Omega(v_m) \geq 2$. If $\Omega(v_l) = 1$ and $\Omega(v_m) = 1$, then inner objects within distance ε from each outer object $r \in \overline{r_i r_j}$ are retrieved among the candidate inner objects in $S_{r_i}^{\varepsilon} \cup S_{r_j}^{\varepsilon} \cup S(\overline{r_i r_j})$ (lines 13-16). If $\Omega(v_l) = 1$ and $\Omega(v_m) \geq 2$, then inner objects within distance ε from each outer object $r \in \overline{r_i r_j}$ are retrieved among the candidate inner objects in $S_{r_i}^{\varepsilon} \cup S_{v_m}^{\varepsilon_{v_m}} \cup S(\overline{r_i v_m})$ (lines 17-19). If $\Omega(v_l) \geq 2$ and $\Omega(v_m) = 1$, then inner objects within distance ε from each outer object $r \in \overline{r_i r_j}$ are retrieved among the candidate inner objects in $S_{v_l}^{\varepsilon_{v_l}} \cup S_{r_j}^{\varepsilon} \cup S(\overline{v_l r_j})$ (lines 20-22). Finally, if $\Omega(v_l) \geq 2$ and $\Omega(v_m) \geq 2$, then inner objects within distance ε from each outer object $r \in \overline{r_i r_j}$ are retrieved among the candidate inner objects in $S_{v_l}^{\varepsilon_{v_l}} \cup S_{v_m}^{\varepsilon_{v_m}} \cup S(\overline{v_l v_m})$ (lines 23-24). A partial join result $\Phi(\overline{r_i r_j})$ for $\overline{r_i r_j}$ is added to the query result $\Phi(R)$. Finally, the ε-distance join query result $\Phi(R)$ is returned after all outer segments have been processed (line 26), i.e., $\Phi(R) = \cup\Phi(\overline{r_i r_j})$ for each outer segment $\overline{r_i r_j} \in \overline{R}$.

Algorithm 2: EDISON (ε, R, S)

Input: R: set of outer objects, S: set of inner objects
Output: $R \bowtie_\varepsilon S$: set of object pairs $(r, s) \in R \times S$ such that $dist(r, s) \leq \varepsilon$

1 $\Phi(R) \leftarrow \varnothing$ // $\Phi(R)$ is the set of object pairs $(r, s) \in R \times S$ such that $dist(r, s) \leq \varepsilon$.
2 // Step 1: neighboring outer objects are grouped and neighboring inner objects are also grouped.
3 $\overline{R} \leftarrow$ group_objects(R) // Outer objects in a vertex sequence are grouped into an outer segment.
4 $\overline{S} \leftarrow$ group_objects(S) // Inner objects in a vertex sequence are grouped into an inner segment.
5 // For simplicity, we assume that $|\overline{R}| \leq |\overline{S}|$. if $|\overline{R}| > |\overline{S}|$, we simply evaluate $S \bowtie_\varepsilon R$.
6 // Step 2: $\overline{r_i r_j} \bowtie_\varepsilon S$ is evaluated by extending shared execution processing to adjacent outer segments.
7 for each intersection vertex v_x do
8 if $\Omega(v_x) \geq 2$ then // $\Omega(v_x) \geq 2$ means that more than two outer segments are adjacent to v_x.
9 $\varepsilon_{v_x} \leftarrow \varepsilon - \min\{len(v_x, r_{i_1}), len(v_x, r_{i_2}), \cdots, len(v_x, r_{i_n})\}$ // assume that $\Omega(v_x) = n$.
10 if $\varepsilon_{v_x} > 0$ then // If $\varepsilon_{v_x} \leq 0$, then no range query is evaluated at v_x.
11 $S_{v_x}^{\varepsilon_{v_x}} \leftarrow RNQ(\varepsilon_{v_x}, v_x)$ // Otherwise, a range query $RNQ(\varepsilon_{v_x}, v_x)$ is evaluated at v_x.
12 for each outer segment $\overline{r_i r_j} \in \overline{v_l v_m}$ do // Assume that r_i (r_j) is close to v_l (v_m).
13 if $\Omega(v_l) = 1$ and $\Omega(v_m) = 1$ then
14 $S_{r_i}^{\varepsilon} \leftarrow RNQ(\varepsilon, r_i)$ // A range query is evaluated at r_i because no range query is issued at v_l.
15 $S_{r_j}^{\varepsilon} \leftarrow RNQ\left(\varepsilon, r_j\right)$ // A range query is evaluated at r_j because no range query is issued at v_m.
16 $\Phi\left(\overline{r_i r_j}\right) \leftarrow DJQ\left(\varepsilon, \overline{r_i r_j}, S_{r_i}^{\varepsilon} \cup S_{r_j}^{\varepsilon} \cup S\left(\overline{r_i r_j}\right)\right)$
17 else if $\Omega(v_l) = 1$ and $\Omega(v_m) \geq 2$ then
18 $S_{r_i}^{\varepsilon} \leftarrow RNQ(\varepsilon, r_i)$ // A range query is evaluated at r_i because no range query is issued at v_l.
19 $\Phi\left(\overline{r_i r_j}\right) \leftarrow DJQ\left(\varepsilon, \overline{r_i r_j}, S_{r_i}^{\varepsilon} \cup S_{v_m}^{\varepsilon_{v_m}} \cup S(\overline{r_i v_m})\right)$ // $S_{v_m}^{\varepsilon_{v_m}}$ is reused by outer segments adjacent to v_m.
20 else if $\Omega(v_l) \geq 2$ and $\Omega(v_m) = 1$ then
21 $S_{r_j}^{\varepsilon} \leftarrow RNQ\left(\varepsilon, r_j\right)$ // A range query is evaluated at r_j because no range query is issued at v_m.
22 $\Phi\left(\overline{r_i r_j}\right) \leftarrow DJQ\left(\varepsilon, \overline{r_i r_j}, S_{v_l}^{\varepsilon_{v_l}} \cup S_{r_j}^{\varepsilon} \cup S\left(\overline{v_l r_j}\right)\right)$ // $S_{v_l}^{\varepsilon_{v_l}}$ is reused by outer segments adjacent to v_l.
23 else if $\Omega(v_l) \geq 2$ and $\Omega(v_m) \geq 2$ then
24 $\Phi\left(\overline{r_i r_j}\right) \leftarrow DJQ\left(\varepsilon, \overline{r_i r_j}, S_{v_l}^{\varepsilon_{v_l}} \cup S_{v_m}^{\varepsilon_{v_m}} \cup S(\overline{v_l v_m})\right)$ // $S_{v_l}^{\varepsilon_{v_l}}$ ($S_{v_m}^{\varepsilon_{v_m}}$) is reused by outer segments adjacent to v_l (v_m).
25 $\Phi(R) \leftarrow \Phi(R) \cup \Phi\left(\overline{r_i r_j}\right)$ // A partial join result $\Phi\left(\overline{r_i r_j}\right)$ for $\overline{r_i r_j}$ is added to $\Phi(R)$.
26 return $\Phi(R)$ // $\Phi(R)$ stores the result of $R \bowtie_\varepsilon S$.

Algorithm 3 retrieves a set $\Phi(\overline{r_i r_j})$ of object pairs (r, s) within distance ε, where $r \in \overline{r_i r_j}$ and $s \in S_\alpha^{\varepsilon_\alpha} \cup S_\beta^{\varepsilon_\beta} \cup S(\overline{\alpha\beta})$. First, $\Phi(\overline{r_i r_j})$ is initialized to the empty set. According to the condition of an

inner object $s \in S_{\alpha}^{\varepsilon_{\alpha}} \cup S_{\beta}^{\varepsilon_{\beta}} \cup S(\overline{\alpha\beta})$, the distance from r to s is computed (lines 4-11). Note that $dist(r,s)$ in Algorithm 3 may not necessarily be the length of the shortest path from r to s, as discussed in Section 5.3. If $dist(r,s) \leq \varepsilon$, then the object pair (r,s) is added to the partial join result $\Phi(\overline{r_i r_j})$ (lines 12-14). Clearly, $\Phi(\overline{r_i r_j})$ is returned after all candidate object pairs (r,s) have been examined (line 15). In Lemma 3, we prove the correctness of the EDISON algorithm.

Algorithm 3: $DJQ\left(\varepsilon, \overline{r_i r_j}, S_{\alpha}^{\varepsilon_{\alpha}} \cup S_{\beta}^{\varepsilon_{\beta}} \cup S(\overline{\alpha\beta})\right)$

Input: ε: threshold distance, $\overline{r_i r_j}$: outer segment,
$S_{\alpha}^{\varepsilon_{\alpha}} \cup S_{\beta}^{\varepsilon_{\beta}} \cup S(\overline{\alpha\beta})$: set of candidate inner objects for outer objects in $\overline{r_i r_j}$
Output: $\Phi(\overline{r_i r_j})$: set of object pairs (r,s) such that $dist(r,s) \leq \varepsilon$ for $r \in \overline{r_i r_j}$ and $s \in S_{\alpha}^{\varepsilon_{\alpha}} \cup S_{\beta}^{\varepsilon_{\beta}} \cup S(\overline{\alpha\beta})$

1 $\Phi(\overline{r_i r_j}) \leftarrow \emptyset$ // $\Phi(\overline{r_i r_j})$ is the set of object pairs (r,s) within distance ε where $r \in \overline{r_i r_j}$ and $s \in S_{\alpha}^{\varepsilon_{\alpha}} \cup S_{\beta}^{\varepsilon_{\beta}} \cup S(\overline{\alpha\beta})$.
2 for each outer object $r \in \overline{r_i r_j}$ do
3 for each inner object $s \in S_{\alpha}^{\varepsilon_{\alpha}} \cup S_{\beta}^{\varepsilon_{\beta}} \cup S(\overline{\alpha\beta})$ do
4 // $dist(r,s)$ is computed according to the condition of s.
5 if $s \in S_{\alpha}^{\varepsilon_{\alpha}} \cap S_{\beta}^{\varepsilon_{\beta}} \cap S(\overline{\alpha\beta})$ then $dist(r,s) \leftarrow min\{len(r,\alpha) + dist(\alpha,s),\ len(r,\beta) + dist(\beta,s), len(r,s)\}$
6 else if $s \in S_{\alpha}^{\varepsilon_{\alpha}} \cap S_{\beta}^{\varepsilon_{\beta}} - S(\overline{\alpha\beta})$ then $dist(r,s) \leftarrow min\{len(r,\alpha) + dist(\alpha,s),\ len(r,\beta) + dist(\beta,s)\}$
7 else if $s \in S_{\alpha}^{\varepsilon_{\alpha}} \cap S(\overline{\alpha\beta}) - S_{\beta}^{\varepsilon_{\beta}}$ then $dist(r,s) \leftarrow min\{len(r,\alpha) + dist(\alpha,s), len(r,s)\}$
8 else if $s \in S_{\beta}^{\varepsilon_{\beta}} \cap S(\overline{\alpha\beta}) - S_{\alpha}^{\varepsilon_{\alpha}}$ then $dist(r,s) \leftarrow min\{len(r,\beta) + dist(\beta,s), len(r,s)\}$
9 else if $s \in S_{\alpha}^{\varepsilon_{\alpha}} - \left(S_{\beta}^{\varepsilon_{\beta}} \cup S(\overline{\alpha\beta})\right)$ then $dist(r,s) \leftarrow len(r,\alpha) + dist(\alpha,s)$
10 else if $s \in S_{\beta}^{\varepsilon_{\beta}} - \left(S_{\alpha}^{\varepsilon_{\alpha}} \cup S(\overline{\alpha\beta})\right)$ then $dist(r,s) \leftarrow len(r,\beta) + dist(\beta,s)$
11 else if $s \in S(\overline{\alpha\beta}) - \left(S_{\alpha}^{\varepsilon_{\alpha}} \cup S_{\beta}^{\varepsilon_{\beta}}\right)$ then $dist(r,s) \leftarrow len(r,s)$
12 // If $dist(r,s) \leq \varepsilon$ then an object pair (r,s) is involved in the query result.
13 if $dist(r,s) \leq \varepsilon$ then
14 $\Phi(\overline{r_i r_j}) \leftarrow \Phi(\overline{r_i r_j}) \cup \{(r,s)\}$
15 return $\Phi(\overline{r_i r_j})$ // A partial join result $\Phi(\overline{r_i r_j})$ for $\overline{r_i r_j}$ is returned.

Lemma 3. *The EDISON algorithm is correct.*

Proof. We prove the correctness of the EDISON algorithm by cases. As shown in Algorithm 2, the EDISON algorithm handles the following four cases differently depending on the values of $\Omega(v_l)$ and $\Omega(v_m)$ of a vertex sequence $\overline{v_l v_m}$ containing an outer segment $\overline{r_i r_j}$, namely, (1) $\Omega(v_l) = \Omega(v_m) = 1$; (2) $\Omega(v_l) = 1$ and $\Omega(v_m) \geq 2$; (3) $\Omega(v_l) \geq 2$ and $\Omega(v_m) = 1$; and (4) $\Omega(v_l) \geq 2$ and $\Omega(v_m) \geq 2$. For $\Omega(v_l) = \Omega(v_m) = 1$, two range queries, $RNQ(\varepsilon, r_i)$ and $RNQ(\varepsilon, r_j)$, are evaluated at r_i and r_j, respectively. According to Lemma 1, we can retrieve the inner objects within distance ε from every outer object $r \in \overline{r_i r_j}$ among the candidate inner objects $s \in S_{r_i}^{\varepsilon} \cup S_{r_j}^{\varepsilon} \cup S(\overline{r_i r_j})$. The EDISON algorithm computes the distance $dist(r,s)$ from an outer object r to each candidate inner object $s \in S_{r_i}^{\varepsilon} \cup S_{r_j}^{\varepsilon} \cup S(\overline{r_i r_j})$. Because $dist(r,s)$ is the length of the shortest path among three possible paths (i.e., $r \rightarrow r_i \rightarrow s$, $r \rightarrow r_j \rightarrow s$, and $r \rightarrow s$ if $s \in \overline{r_i r_j}$), it is determined simply depending on the conditions listed in Algorithm 3. Specifically, if $s \notin \overline{r_i r_j}$, then $dist(r,s) = min\{len(r,r_i) + dist(r_i,s), len(r,r_j) + dist(r_j,s)\}$; otherwise, $dist(r,s) = min\{len(r,r_i) + dist(r_i,s), len(r,r_j) + dist(r_j,s), len(r,s)\}$. If $dist(r,s) \leq \varepsilon$, an object pair (r,s) is included in the ε-distance join query result; otherwise, the object pair (r,s) is not included. Therefore, the EDISON algorithm is correct for $\Omega(v_l) = \Omega(v_m) = 1$.

For $\Omega(v_l) = 1$ and $\Omega(v_m) \geq 2$, two range queries, $RNQ(\varepsilon, r_i)$ and $RNQ(\varepsilon_{v_m}, v_m)$, are evaluated at r_i and v_m, respectively, where $\varepsilon_{v_m} = \varepsilon - min\{len(v_m, r_{j_1}), len(v_m, r_{j_2}), \cdots, len(v_m, r_{j_n})\}$, assuming that the outer segments $\overline{r_{i_1} r_{j_1}}, \overline{r_{i_2} r_{j_2}}, \cdots, \overline{r_{i_n} r_{j_n}}$ are adjacent to v_m, and that r_{jk} is closer to v_m than r_{i_k} for $1 \leq k \leq \Omega(v_m)$. Because $\overline{r_i r_j} \in \{\overline{r_{i_1} r_{j_1}}, \overline{r_{i_2} r_{j_2}}, \cdots, \overline{r_{i_n} r_{j_n}}\}$ and $\varepsilon_{v_m} \geq \varepsilon'$, set $S_{v_m}^{\varepsilon_{v_m}}$ of the inner objects within distance ε_{v_m} from v_m contains set $S_{v_m}^{\varepsilon'}$ of the inner objects within distance ε' from v_m, where $\varepsilon' = \varepsilon - len(v_m, r_j)$, i.e., $S_{v_m}^{\varepsilon'} \subseteq S_{v_m}^{\varepsilon_{v_m}}$. According to Lemma 1, we can retrieve inner objects within distance ε from every outer object $r \in \overline{r_i r_j}$ among the candidate inner objects $s \in S_{r_i}^{\varepsilon} \cup S_{v_m}^{\varepsilon_{v_m}} \cup S(\overline{r_i v_m})$ because $S_{r_i}^{\varepsilon} \cup S_{v_m}^{\varepsilon_{v_m}} \cup S(\overline{r_i v_m})$ contains $S_{r_i}^{\varepsilon} \cup S_{r_j}^{\varepsilon} \cup S(\overline{r_i r_j})$. The EDISON algorithm computes the distance

$dist(r,s)$ from an outer object r to each candidate inner object $s \in S_{r_i}^{\varepsilon} \cup S_{v_m}^{\varepsilon_{v_m}} \cup S(\overline{r_i v_m})$. Because $dist(r,s)$ is the length of the shortest path among three possible paths (i.e., $r \to r_i \to s$, $r \to v_m \to s$, and $r \to s$ if $s \in \overline{r_i v_m}$), it is determined simply depending on the conditions listed in Algorithm 3. Specifically, if $s \notin \overline{r_i v_m}$, then $dist(r,s) = min\{len(r,r_i) + dist(r_i,s), len(r,v_m) + dist(v_m,s)\}$; otherwise, $dist(r,s) = min\{len(r,r_i) + dist(r_i,s), len(r,v_m) + dist(v_m,s), len(r,s)\}$. If $dist(r,s) \le \varepsilon$, an object pair (r,s) is included in the ε-distance join query result; otherwise, the object pair (r,s) is not included. Therefore, the EDISON algorithm is correct for $\Omega(v_l) = 1$ and $\Omega(v_m) \ge 2$. Without loss of generality, the proof of the correctness of the EDISON algorithm for $\Omega(v_l) \ge 2$ and $\Omega(v_m) = 1$ can be simply obtained by interchanging the roles of v_l and v_m, as well as the roles of r_i and r_j, in the proof for $\Omega(v_l) = 1$ and $\Omega(v_m) \ge 2$, which is omitted.

Finally, for $\Omega(v_l) \ge 2$ and $\Omega(v_m) \ge 2$, two range queries, $RNQ(\varepsilon_{v_l}, v_l)$ and $RNQ(\varepsilon_{v_m}, v_m)$, are evaluated at v_l and v_m, respectively, where $\varepsilon_{v_l} = \varepsilon - min\{len(v_l,r_{a_1}), \cdots, len(v_l,r_{a_n})\}$ and $\varepsilon_{v_m} = \varepsilon - \min\{len(v_m,r_{d_1}), \cdots, len(v_m,r_{d_n})\}$. It is assumed that the outer segments $\overline{r_{a_1}r_{b_1}}, \cdots, \overline{r_{a_n}r_{b_n}}$ $(\overline{r_{c_1}r_{d_1}}, \cdots, \overline{r_{c_n}r_{d_n}})$ are adjacent to v_l (v_m), and that r_{ak} (r_{d_k}) is closer to v_l (v_m) than r_{b_k} (r_{c_k}) for $1 \le k \le \Omega(v_l)$ $(1 \le k \le \Omega(v_m))$. Because $\overline{r_i r_j} \in \{\overline{r_{a_1}r_{b_1}}, \cdots, \overline{r_{a_n}r_{b_n}}\}$ $(\overline{r_i r_j} \in \{\overline{r_{c_1}r_{d_1}}, \cdots, \overline{r_{c_n}r_{d_n}}\})$ and $\varepsilon_{v_l} \ge \varepsilon - len(v_l,r_i)$ $(\varepsilon_{v_m} \ge \varepsilon - len(v_m,r_j))$, set $S_{v_l}^{\varepsilon_{v_l}}$ $\left(S_{v_m}^{\varepsilon_{v_m}}\right)$ of the inner objects within distance ε_{v_l} (ε_{v_m}) from v_l (v_m) contains set $S_{v_l}^{\varepsilon''}$ $\left(S_{v_m}^{\varepsilon'}\right)$ of the inner objects within distance ε'' (ε') from v_l (v_m), where $\varepsilon'' = \varepsilon - len(v_l,r_i)$ $(\varepsilon' = \varepsilon - len(v_m,r_j))$, i.e., $S_{v_l}^{\varepsilon''} \subseteq S_{v_l}^{\varepsilon_{v_l}}$ $\left(S_{v_m}^{\varepsilon'} \subseteq S_{v_m}^{\varepsilon_{v_m}}\right)$. According to Lemma 1, we can retrieve the inner objects within distance ε from every outer object $r \in \overline{r_i r_j}$ among the candidate inner objects $s \in S_{v_l}^{\varepsilon_{v_l}} \cup S_{v_m}^{\varepsilon_{v_m}} \cup S(\overline{v_l v_m})$ because $S_{v_l}^{\varepsilon_{v_l}} \cup S_{v_m}^{\varepsilon_{v_m}} \cup S(\overline{v_l v_m})$ contains $S_{r_i}^{\varepsilon} \cup S_{r_j}^{\varepsilon} \cup S(\overline{r_i r_j})$. The EDISON algorithm computes the distance $dist(r,s)$ from an outer object $r \in \overline{r_i r_j}$ to each candidate inner object $s \in S_{v_l}^{\varepsilon_{v_l}} \cup S_{v_m}^{\varepsilon_{v_m}} \cup S(\overline{v_l v_m})$. Because $dist(r,s)$ is the length of the shortest path among three possible paths (i.e., $r \to v_l \to s$, $r \to v_m \to s$, and $r \to s$ if $s \in \overline{v_l v_m}$), it is determined simply depending on the conditions listed in Algorithm 3. Specifically, if $s \notin \overline{v_l v_m}$, then $dist(r,s) = min\{len(r,v_l) + dist(v_l,s), len(r,v_m) + dist(v_m,s)\}$; otherwise, $dist(r,s) = min\{len(r,v_l) + dist(v_l,s), len(r,v_m) + dist(v_m,s), len(r,s)\}$. If $dist(r,s) \le \varepsilon$, an object pair (r,s) is included in the ε-distance join query result; otherwise, the object pair (r,s) is not included. Therefore, the EDISON algorithm is correct for $\Omega(v_l) \ge 2$ and $\Omega(v_m) \ge 2$. Consequently, the EDISON algorithm is correct for $\Omega(v_l) = \Omega(v_m) = 1$, $\Omega(v_l) = 1$ and $\Omega(v_m) \ge 2$, $\Omega(v_l) \ge 2$ and $\Omega(v_m) = 1$, and $\Omega(v_l) \ge 2$ and $\Omega(v_m) \ge 2$. □

5.3. Evaluation of an Example ε-Distance Join Query Using the EDISON Algorithm

We discuss how to evaluate the ε-distance join query in Figure 3 using the EDISON algorithm. As shown in Figure 4, R and S are grouped into $\overline{R} = \{\overline{r_1 r_2}, \overline{r_3 r_5 r_4}\}$ and $\overline{S} = \{\overline{s_1 s_5 s_4}, \overline{s_2 s_3}, s_6\}$, respectively. Because $|\overline{R}| < |\overline{S}|$, we evaluate $R \bowtie_{\varepsilon} S$. There are two intersection vertices, v_1 and v_3, both of which are adjacent to $\overline{r_1 r_2}$ and $\overline{r_3 r_5 r_4}$. Therefore, to determine whether range queries at v_1 and v_3 are evaluated, the EDISON algorithm computes the distances ε_{v_1} and ε_{v_3} for the range queries at v_1 and v_3, respectively. Because $\varepsilon_{v_1} = \varepsilon - min\{len(v_1,r_1), len(v_1,r_4)\} = 5 - min\{12,7\} = -2$ and $\varepsilon_{v_3} = \varepsilon - min\{len(v_3,r_2), len(v_3,r_3)\} = 5 - min\{2,1\} = 4$, the EDISON algorithm evaluates the range query $RNQ(4,v_3)$ only. Clearly, the range query $RNQ(-2,v_1)$ returns the empty set. Table 4 summarizes the computation of $R \bowtie_{\varepsilon} S$ for the EDISON algorithm.

Table 4. Computation of $R \bowtie_{\varepsilon} S$ using the EDISON algorithm.

$\overline{r_i r_j}$	$\overline{\mathbf{fffi}}$	**ff**	**fi**	$S''_{\mathbf{ff}}$	$S''_{\mathbf{fi}}$	$S\left(\overline{\mathbf{fffi}}\right)$	$S''_{\mathbf{ff}} \cup S''_{\mathbf{fi}} \cup S\left(\overline{\mathbf{fffi}}\right)$
$\overline{r_1 r_2}$	$\overline{v_1 v_2 v_3}$	v_1	v_3	$S_{v_1}^{-2} = \varnothing$	$S_{v_3}^{4} = \{s_6\}$	$S(\overline{v_1 v_2 v_3}) = \{s_1, s_4, s_5\}$	$\{s_1, s_4, s_5, s_6\}$
$\overline{r_3 r_5 r_4}$	$\overline{v_1 v_4 v_3}$	v_1	v_3	$S_{v_1}^{-2} = \varnothing$	$S_{v_3}^{4} = \{s_6\}$	$S(\overline{v_1 v_4 v_3}) = \{s_2, s_3\}$	$\{s_2, s_3, s_6\}$

We retrieve inner objects within distance ε from each outer object $r \in \overline{r_1 r_2}$ among the candidate inner objects, followed by inner objects within distance ε from each outer object $r \in \overline{r_3 r_5 r_4}$. As shown in Table 4, $\{s_1, s_4, s_5, s_6\}$ is the set of candidate inner objects for $\overline{r_1 r_2}$, and $\{s_2, s_3, s_6\}$ is the set of candidate inner objects for $\overline{r_3 r_5 r_4}$. The EDISON algorithm computes the distance between an outer object r and each candidate inner object s and finds all qualifying object pairs (r, s) such that $dist(r, s) \leq \varepsilon$.

We compute the distance from r_1 to each candidate inner object $s \in \{s_1, s_4, s_5, s_6\}$. Because $s_1 \in S(\overline{v_1 v_2 v_3}) - (S^{-2}_{v_1} \cup S^{4}_{v_3})$ according to Table 4, the distance from r_1 to s_1 is $dist(r_1, s_1) = len(r_1, s_1) = 2$, as shown in Figure 10a. Because $s_4 \in S(\overline{v_1 v_2 v_3}) - \left(S^{-2}_{v_1} \cup S^{4}_{v_3}\right)$, the distance from r_1 to s_4 is $dist(r_1, s_4) = len(r_1, s_4) = 10$, as shown in Figure 10b. Similarly, because $s_5 \in S(\overline{v_1 v_2 v_3}) - \left(S^{-2}_{v_1} \cup S^{4}_{v_3}\right)$, the distance from r_1 to s_5 is $dist(r_1, s_5) = len(r_1, s_5) = 3$, as shown in Figure 10c. Finally, because $s_6 \in S^{4}_{v_3} - \left(S^{-2}_{v_1} \cup S(\overline{v_1 v_2 v_3})\right)$ according to Table 4, the distance from r_1 to s_6 is $dist(r_1, s_6) = len(r_1, v_3) + dist(v_3, s_6) = 5 + 3 = 8$, as shown in Figure 10d. Consequently, $S^{\varepsilon}_{r_1} = \{s_1, s_5\}$ given that $dist(r_1, s_1) = 2$, $dist(r_1, s_4) = 10$, $dist(r_1, s_5) = 3$, and $dist(r_1, s_6) = 8$, and the generated partial join result is $\Phi(r_1) = \{(r_1, s_1), (r_1, s_5)\}$.

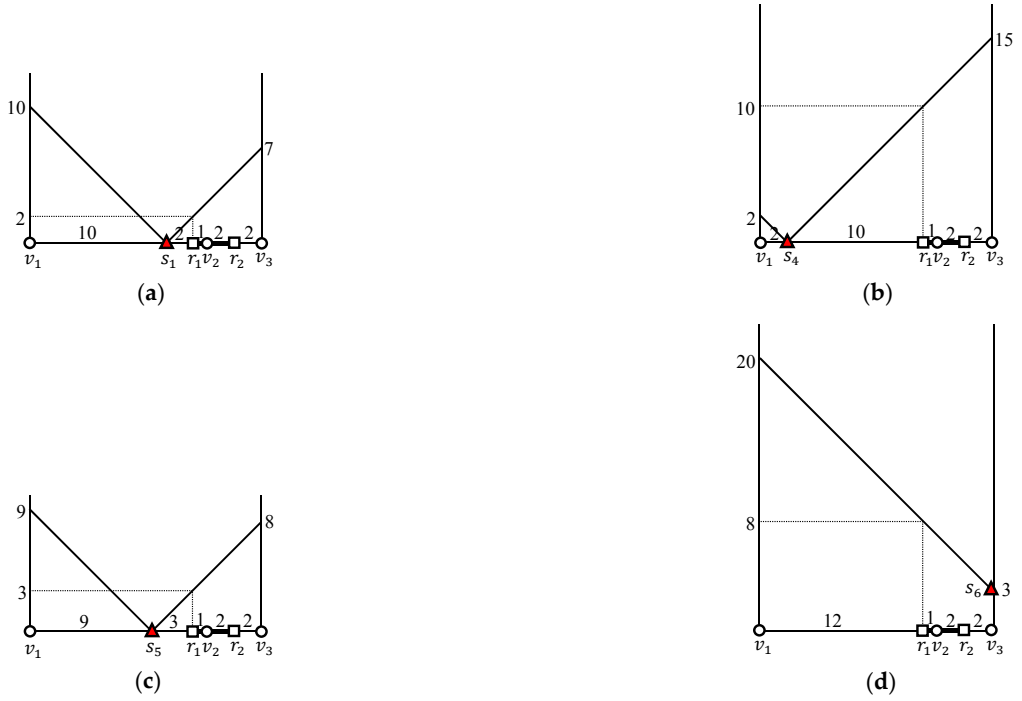

Figure 10. Computation of the distance from r_1 to $s \in \{s_1, s_4, s_5, s_6\}$ **(a)** $dist(r_1, s_1) = 2$; **(b)** $dist(r_1, s_4) = 10$; **(c)** $dist(r_1, s_5) = 3$; **(d)** $dist(r_1, s_6) = 8$.

We compute the distance from r_2 to each candidate inner object $s \in \{s_1, s_4, s_5, s_6\}$. Because $s_1 \in S(\overline{v_1 v_2 v_3}) - (S^{-2}_{v_1} \cup S^{4}_{v_3})$ according to Table 4, the distance from r_2 to s_1 is $dist(r_2, s_1) = len(r_2, s_1) = 5$, as shown in Figure 11a. Because $s_4 \in S(\overline{v_1 v_2 v_3}) - (S^{-2}_{v_1} \cup S^{4}_{v_3})$, the distance from r_2 to s_4 is $dist(r_2, s_4) = len(r_2, s_4) = 13$, as shown in Figure 11b. In fact, the shortest path from r_2 to s_4 is $r_2 \rightarrow v_3 \rightarrow v_1 \rightarrow s_4$, whose length is $dist(r_2, s_4) = 9$. However, this deviation from the shortest distance does not affect the query result. Similarly, because $s_5 \in S(\overline{v_1 v_2 v_3}) - (S^{-2}_{v_1} \cup S^{4}_{v_3})$, this distance from r_2 to s_5 is $dist(r_2, s_5) = len(r_2, s_5) = 6$, as shown in Figure 11c. Finally, because $s_6 \in S^{4}_{v_3} - \left(S^{-2}_{v_1} \cup S(\overline{v_1 v_2 v_3})\right)$ according to Table 4, the distance from r_2 to s_6 is $dist(r_2, s_6) = len(r_2, v_3) + dist(v_3, s_6) = 2 + 3 = 5$, as shown in Figure 11d. Consequently, $S^{\varepsilon}_{r_2} = \{s_1, s_6\}$ given that $dist(r_2, s_1) = 5$, $dist(r_2, s_4) = 13$, $dist(r_2, s_5) = 6$, and $dist(r_2, s_6) = 5$, and the generated partial join result is $\Phi(r_2) = \{(r_2, s_1), (r_2, s_6)\}$.

We compute the distance from r_3 to each candidate inner object $s \in \{s_2, s_3, s_6\}$. Because $s_2 \in S(\overline{v_1 v_4 v_3}) - \left(S^{-2}_{v_1} \cup S^{4}_{v_3}\right)$ according to Table 4, the distance from r_3 to s_2 is $dist(r_3, s_2) = len(r_3, s_2) = 6$,

as shown in Figure 12a. Similarly, because $s_3 \in S(\overline{v_1v_4v_3}) - (S^{-2}_{v_1} \cup S^{4}_{v_3})$, the distance from r_3 to s_3 is $dist(r_3, s_3) = len(r_3, s_3) = 12$, as shown in Figure 12b. In fact, the shortest path from r_3 to s_3 is $r_3 \rightarrow v_3 \rightarrow v_1 \rightarrow s_3$, whose length is $dist(r_3, s_3) = 10$. However, this deviation from the shortest distance does not affect the query result. Finally, because $s_6 \in S^{4}_{v_3} - \left(S^{-2}_{v_1} \cup (\overline{v_1v_4v_3})\right)$, the distance from r_3 to s_6 is $dist(r_3, s_6) = len(r_3, v_3) + dist(v_3, s_6) = 1 + 3 = 4$, as shown in Figure 12c. Consequently, $S^{\varepsilon}_{r_3} = \{s_6\}$ given that $dist(r_3, s_2) = 6$, $dist(r_3, s_3) = 12$, and $dist(r_3, s_6) = 4$, and the generated partial join result is $\Phi(r_3) = \{(r_3, s_6)\}$.

We compute the distance from r_4 to each candidate inner object $s \in \{s_2, s_3, s_6\}$. Because $s_2 \in S(\overline{v_1v_4v_3}) - \left(S^{-2}_{v_1} \cup S^{4}_{v_3}\right)$ according to Table 4, the distance from r_4 to s_2 is $dist(r_4, s_2) = len(r_4, s_2) = 3$, as shown in Figure 13a. Similarly, because $s_3 \in S(\overline{v_1v_4v_3}) - \left(S^{-2}_{v_1} \cup S^{4}_{v_3}\right)$, the distance from r_4 to s_3 is $dist(r_4, s_3) = len(r_4, s_3) = 3$, as shown in Figure 13b. Finally, because $s_6 \in S^{4}_{v_3} - \left(S^{-2}_{v_1} \cup S(\overline{v_1v_4v_3})\right)$, the distance from r_4 to s_6 is $dist(r_4, s_6) = len(r_4, v_3) + dist(v_3, s_6) = 10 + 3 = 13$, as shown in Figure 13c. In fact, the shortest path from r_4 to s_6 is $r_4 \rightarrow v_4 \rightarrow v_1 \rightarrow s_6$, whose length is $dist(r_4, s_6) = 9$. However, this deviation from the shortest distance does not affect the query result. Consequently, $S^{\varepsilon}_{r_4} = \{s_2, s_3\}$ given that $dist(r_4, s_2) = 3$, $dist(r_4, s_3) = 3$, and $dist(r_4, s_6) = 13$, and the generated partial join result is $\Phi(r_4) = \{(r_4, s_2), (r_4, s_3)\}$.

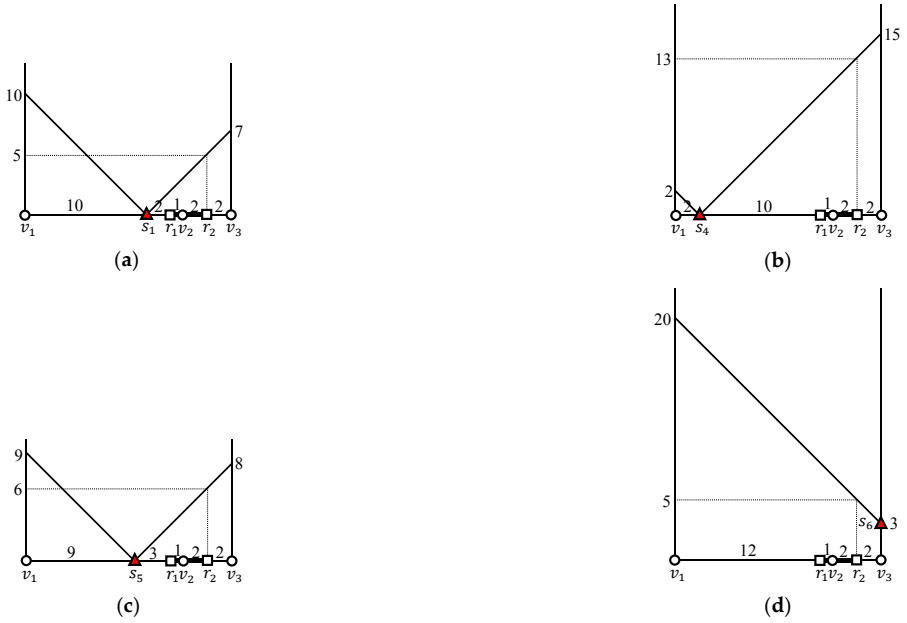

Figure 11. Computation of the distance from r_2 to $s \in \{s_1, s_4, s_5, s_6\}$ (**a**) $dist(r_2, s_1) = 5$; (**b**) $dist(r_2, s_4) = 13$; (**c**) $dist(r_2, s_5) = 6$; (**d**) $dist(r_2, s_6) = 5$.

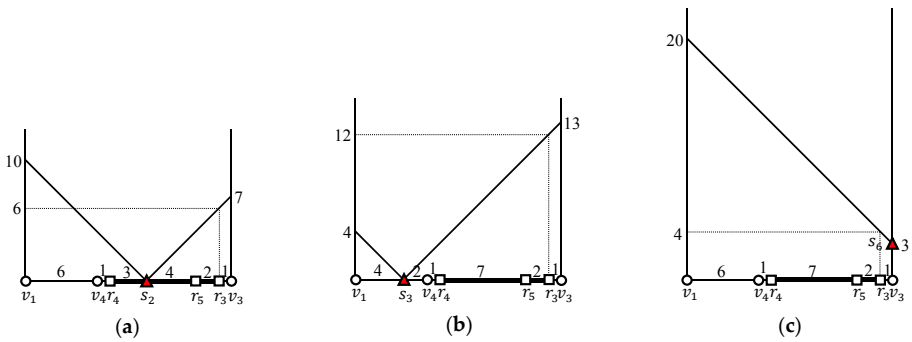

Figure 12. Computation of the distance from r_3 to $s \in \{s_2, s_3, s_6\}$ (**a**) $dist(r_3, s_2) = 6$; (**b**) $dist(r_3, s_3) = 12$; (**c**) $dist(r_3, s_6) = 4$.

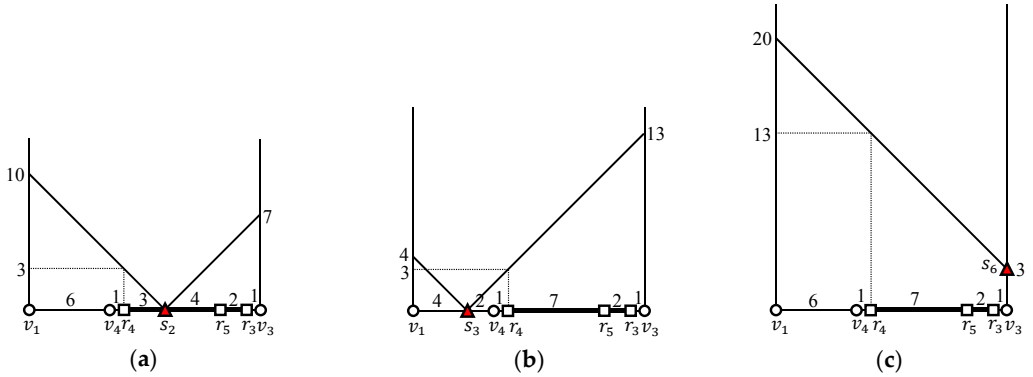

Figure 13. Computation of the distance from r_4 to $s \in \{s_2, s_3, s_6\}$ (**a**) $dist(r_4, s_2) = 3$; (**b**) $dist(r_4, s_3) = 3$; (**c**) $dist(r_4, s_6) = 13$.

We compute the distance from r_5 to each candidate inner object $s \in \{s_2, s_3, s_6\}$. Because $s_2 \in S(\overline{v_1 v_4 v_3}) - \left(S^{-2}_{v_1} \cup S^{4}_{v_3}\right)$ according to Table 4, the distance from r_5 to s_2 is $dist(r_5, s_2) = len(r_5, s_2) = 4$, as shown in Figure 14a. Similarly, because $s_3 \in S(\overline{v_1 v_4 v_3}) - \left(S^{-2}_{v_1} \cup S^{4}_{v_3}\right)$, the distance from r_5 to s_3 is $dist(r_5, s_3) = len(r_5, s_3) = 10$, as shown in Figure 14b. Finally, because $s_6 \in S^{4}_{v_3} - \left(S^{-2}_{v_1} \cup S(\overline{v_1 v_4 v_3})\right)$, the distance from r_5 to s_6 is $dist(r_5, s_6) = len(r_5, v_3) + dist(v_3, s_6) = 3 + 3 = 6$, as shown in Figure 14c. Consequently, $S^{\varepsilon}_{r_5} = \{s_2\}$ given that $dist(r_5, s_2) = 4$, $dist(r_5, s_3) = 10$, and $dist(r_5, s_6) = 6$, and the generated partial join result is $\Phi(r_5) = \{(r_5, s_2)\}$. Finally, we obtain the complete query result $\Phi(R) = \Phi(r_1) \cup \Phi(r_2) \cup \Phi(r_3) \cup \Phi(r_4) \cup \Phi(r_5) = \{(r_1, s_1), (r_1, s_5), (r_2, s_1), (r_2, s_6), (r_3, s_6), (r_4, s_2), (r_4, s_3), (r_5, s_2)\}$.

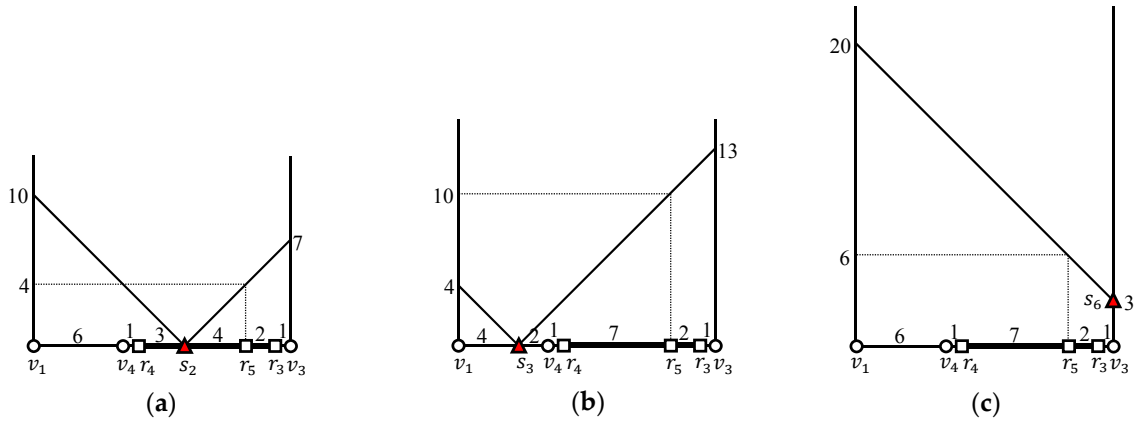

Figure 14. Computation of the distance from r_5 to $s \in \{s_2, s_3, s_6\}$ (**a**) $dist(r_5, s_2) = 4$; (**b**) $dist(r_5, s_3) = 10$; (**c**) $dist(r_5, s_6) = 6$.

6. Performance Study

In this section, we report on an empirical analysis of our proposed solution. We present our experimental settings in Section 6.1, followed by our experimental results in Section 6.2.

6.1. Experimental Settings

For the performance study, we use three real-world road networks [43], which are described in Table 5. These real-world road networks have different sizes and are part of the US's road network. Table 6 shows the range of each variable used in the experiments with defaults indicated in bold. For convenience, each dimension of the data universe is normalized independently to unit length, such that the threshold distance ε is in the range of $[0, 1]$. The positions of both the outer objects and the inner

objects follow either centroid or uniform distributions. The centroid dataset is generated to resemble the real-world data. First, 10 centroids are selected randomly. The objects around each centroid follow a normal distribution, where the mean is set to the centroid and the standard deviation is set to 1% of the side length of the data universe. In each experiment, we vary one or two of the parameters within the range shown in Table 6, while keeping other parameters at default values. The outer objects and the inner objects follow the centroid distribution unless stated otherwise.

Table 5. Real-world roadmaps.

Name	Description	Number of Vertices	Number of Edges	Number of Vertex Sequences
CAL	California and Nevada	1,890,815	2,315,222	1,794,708
FLA	Florida	1,070,376	1,343,951	1,100,675
COL	Colorado	435,666	521,200	374,355

Table 6. Experimental parameter settings.

Parameter	Range
Threshold distance (ε)	0.005, 0.01, **0.03**, 0.05, 0.1
Numbers of outer objects ($\|R\|$) and inner objects ($\|S\|$)	1, 5, **10**, 20, 30 ($\times 10^3$) for CAL and FLA 1, 3, **5**, 7, 10 ($\times 10^3$) for COL
Distributions of outer objects	**(C)entroid**, (U)niform
Distributions of inner objects	(C)entroid, **(U)niform**
Real-world roadmaps	CAL, FLA, COL

We implement and evaluate two versions of our proposed solution, i.e., the naive EDISON and EDISON methods. As a benchmark for our proposed method, we use a baseline method that computes the range query of every outer object using the RNE algorithm [10]. A comparison with the pre-computed distance-based solution [23] and the Euclidean distance-based solution (e.g., JER [10]) is beyond the scope of this study, because these methods cannot support frequent network traffic updates.

All algorithms are implemented in C++ in Microsoft Visual Studio 2015, and they use common subroutines for similar tasks. We conduct experiments on a desktop computer running Windows 10 with a 4 GHz processor and 32 GB of memory. We believe that indexing structures of all techniques should be memory resident to ensure responsive query processing, which is assumed in many recent studies [5,11] and is crucial to online map services and commercial navigation systems. We determine the average values based on 10 repetitions of the experiments for each algorithm.

6.2. Experimental Results

Figure 15 compares the query processing times using the baseline, naive EDISON, and EDISON methods to evaluate ε-distance join queries in the CAL roadmap, where each chart illustrates the effect of changing one or two of the parameters in Table 6. The first and the second values in parentheses indicate the number of range queries that are evaluated by the naive EDISON and EDISON methods, respectively. The numbers of range queries evaluated by the baseline method are omitted, because these numbers become $min\{|R|, |S|\}$, i.e., the cardinality of the smaller dataset between $|R|$ and $|S|$. Figure 15a shows the query processing time as a function of the threshold distance ε. Although the EDISON method shows the worst performance for $\varepsilon = 0.005$, the processing times using the baseline and the naive EDISON methods increase more rapidly with the value of ε than those using the EDISON method. This implies that the shared execution of the EDISON method is more effective for a large threshold distance ε. The baseline, naive EDISON, and EDISON methods evaluate a total of 10,000 range queries, 959 range queries, and 448 range queries, respectively. Figure 15b shows the query processing time as a function of the number $|R|$ of outer objects, while the number of inner objects is fixed at $|S| = 10,000$. The EDISON method is less sensitive to variations in $|R|$ than the other methods, although it shows the worst performance at $|R| = 1000$. Owing to the benefit of the shared execution

processing, the numbers of range queries evaluated by the naive EDISON and EDISON methods increase slightly with $|R|$.

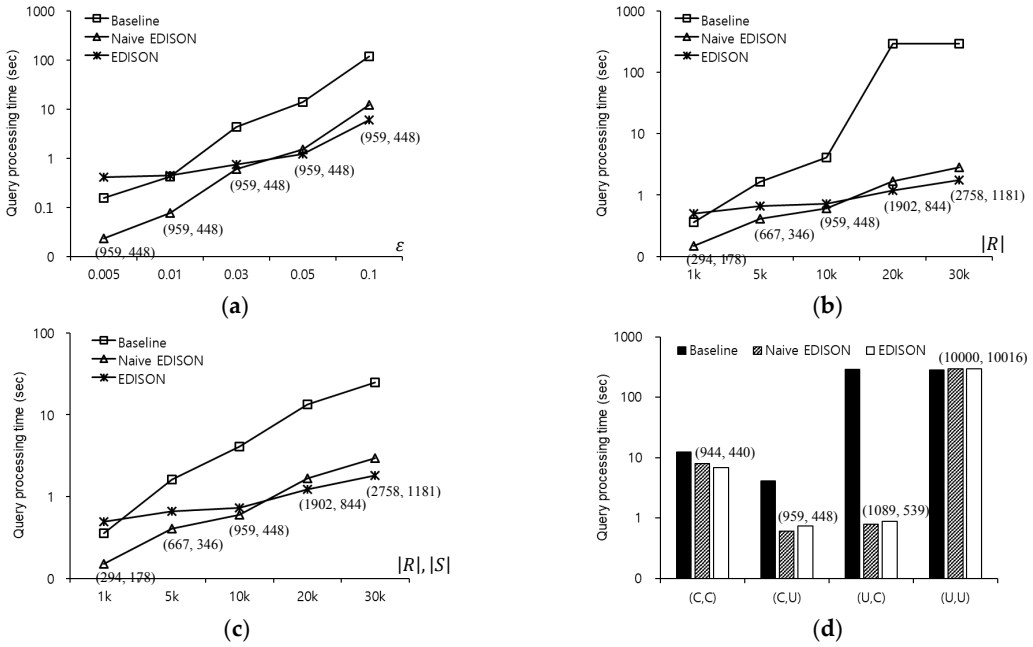

Figure 15. Comparison of the baseline, naive EDISON, and EDISON methods for CAL (**a**) varying ε; (**b**) varying $|R|$; (**c**) varying $|R|$ and $|S|$; (**d**) varying the distributions of objects.

Figure 15c shows the query processing time as a function of both the number $|R|$ of outer objects and the number $|S|$ of inner objects. The naive EDISON method outperforms the EDISON method for $1000 \leq |R|, |S| \leq 10,000$, whereas the latter outperforms the former for $20,000 \leq |R|, |S| \leq 30,000$. This indicates that the EDISON method optimizes the shared execution processing more effectively than the naive EDISON method for large datasets. Clearly, the baseline method shows the worst performance in most cases. Figure 15d shows the query processing time for various distributions of outer objects and inner objects, where each ordered pair (i.e., (C,C), (C,U), (U,C), and (U,U)) denotes a combination of the distributions of outer objects and inner objects. Because shared execution processing is favorable for non-uniform distributions of objects, the naive EDISON and EDISON methods significantly outperform the baseline method for (C,C), (C,U), and (U,C). However, the processing times of the naive EDISON and EDISON methods for (U,U) are very similar to the baseline method. This is because both the outer objects and the inner objects are widely scattered, which hinders shared execution processing.

Figure 16 compares the query processing times using the baseline, naive EDISON, and EDISON methods to evaluate ε-distance join queries in the FLA roadmap. Figure 16a shows the query processing time as a function of ε, when ε varies between 0.005 and 0.1. The EDISON method achieves the best performance for $0.01 \leq \varepsilon \leq 0.1$, because it evaluates the smallest number of range queries among the three methods. The baseline, naive EDISON, and EDISON methods evaluate 10,000 range queries, 1320 range queries, and 665 range queries, respectively. Figure 16b shows the query processing time as a function of $|R|$, when $|R|$ varies between 1000 and 30,000. The EDISON method clearly outperforms the other methods for $5000 \leq |R| \leq 30,000$. Due to shared execution processing, the naive EDISON and EDISON methods are less sensitive to changes in $|R|$ than the baseline method. Figure 16c shows the query processing time as a function of $|R|$ and $|S|$, when $|R|$ ($|S|$) varies between 1000 and 30,000. The EDISON method clearly outperforms the other methods for $5000 \leq |R|, |S| \leq 30,000$. Figure 16d shows the query processing time for various distributions of outer objects and inner objects. The naive EDISON and EDISON methods significantly outperform the baseline method for (C,C), (C,U), and

(U, C), whereas the former methods show similar performance to the latter method for (U, U) for the same reason as in the CAL case.

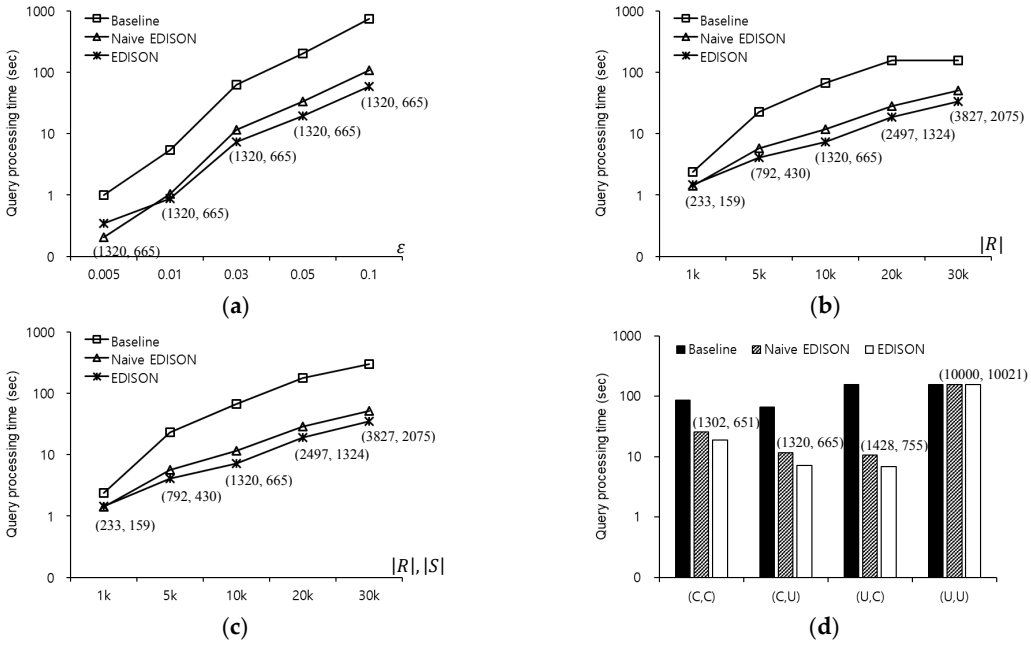

Figure 16. Comparison of the baseline, naive EDISON, and EDISON methods for FLA (**a**) varying ε; (**b**) varying $|R|$; (**c**) varying $|R|$ and $|S|$; (**d**) varying the distributions of objects.

Figure 17 compares the query processing times using the baseline, naive EDISON, and EDISON methods to evaluate ε-distance join queries in the COL roadmap. Figure 17a shows the query processing time as a function of ε, when ε varies between 0.005 and 0.1. The naive EDISON and EDISON methods significantly outperform the baseline method in all cases. Specifically, the query processing time of the EDISON method is up to 11 times shorter than the baseline method at $\varepsilon = 0.1$. The naive EDISON method significantly outperforms the EDISON method for $0.005 \leq \varepsilon \leq 0.01$, whereas the latter outperforms the former for $0.03 \leq \varepsilon \leq 0.1$. This indicates that the EDISON method is less sensitive to changes in ε than the naive EDISON method. Figure 17b shows the query processing time as a function of $|R|$, when $|R|$ varies between 1000 and 10,000. The naive EDISON and EDISON methods significantly outperform the baseline method in all cases and the former methods are less sensitive to changes in $|R|$ than the latter method. This indicates that the performance difference between the EDISON method and the baseline method increases rapidly with $|R|$. Specifically, the query processing time of the EDISON method is up to 155 times shorter than the baseline method at $|R| = 10,000$. Figure 17c shows the query processing time as a function of $|R|$ and $|S|$, when $|R|$ ($|S|$) varies between 1000 and 10,000. The naive EDISON method outperforms the EDISON method at $|R| = |S| = 1000$, whereas the latter outperforms the former for $5000 \leq |R|, |S| \leq 10,000$, and the performance difference between the two methods increases with $|R|$. and $|S|$. This implies that the EDISON method scales better with $|R|$ and $|S|$ than the naive EDISON method. Figure 17d shows the query processing time for various distributions of outer objects and inner objects. The naive EDISON and EDISON methods significantly outperform the baseline method for (C, C), (C, U), and (U, C). However, all methods show similar performance when the outer objects and the inner objects follow uniform distributions (U, U). This is expected because both the outer objects and inner objects are widely scattered for (U, U), which obstructs shared execution processing of the naive EDISON and EDISON methods.

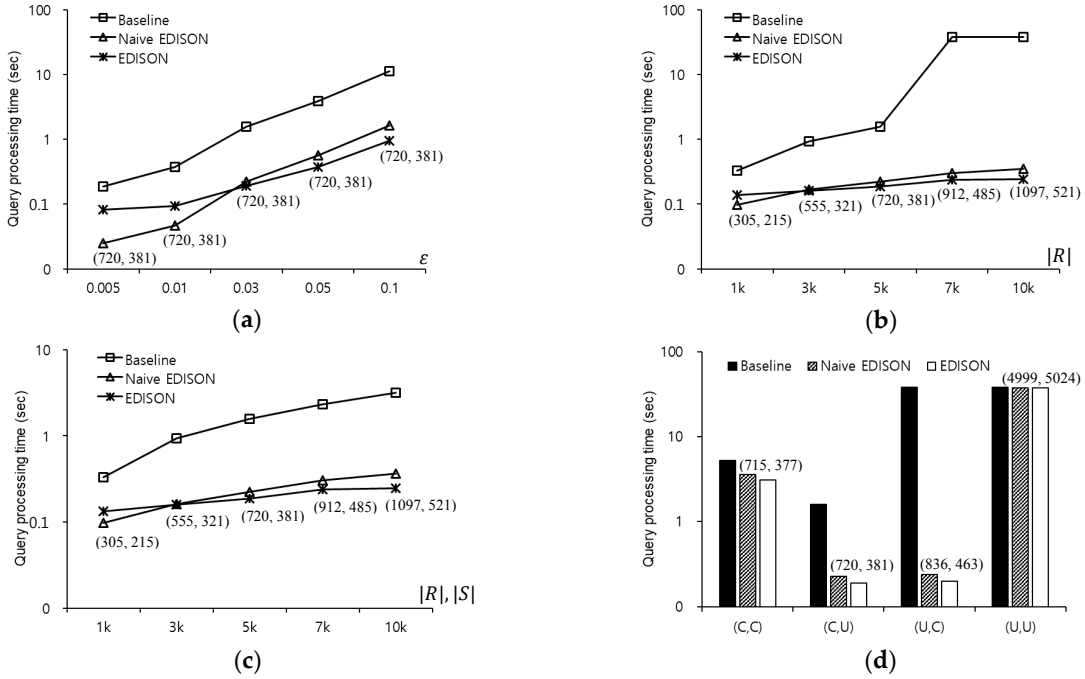

Figure 17. Comparison of the baseline, naive EDISON, and EDISON methods for COL (**a**) varying ε; (**b**) varying $|R|$; (**c**) varying $|R|$ and $|S|$; (**d**) varying the distributions of objects.

7. Conclusions

In this study, we investigated the efficient processing of ε-distance join queries in dynamic road networks. We proposed a cost-effective solution called EDISON that optimizes the shared execution method to avoid redundant network traversal. We implemented and evaluated a baseline method and two versions of EDISON, i.e., the naive EDISON and EDISON methods. The experiments are based on several real-world roadmaps and involve a wide range of parameter values. The experimental results are summarized as follows: (1) the naive EDISON and EDISON methods significantly outperform the baseline method; (2) the naive EDISON and EDISON methods are typically comparable in terms of query processing time; (3) the EDISON method scales better with increasing number of objects and threshold distance than the naive EDISON method. In future work, we plan to extend the shared execution approach used here to the problems of processing sophisticated spatial queries over road networks, such as multi-way distance join queries [44] and aggregate k-farthest neighbor queries [45,46]. These problems have not been adequately addressed with regard to road networks despite their importance.

Author Contributions: H.-J.C. designed the study, performed the experiments, and wrote the paper.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIP) (NRF-2016R1A2B4009793).

Conflicts of Interest: The author declares no conflict of interest.

References

1. Delling, D.; Goldberg, A.V.; Pajor, T.; Werneck, R.F. Customizable route planning in road networks. *Transp. Sci.* **2017**, *51*, 566–591. [CrossRef]
2. Samet, H.; Sankaranarayanan, J.; Alborzi, H. Scalable network distance browsing in spatial databases. In Proceedings of the International Conference on Management of Data, Vancouver, BC, Canada, 10–12 June 2008.

3. Sankaranarayanan, J.; Alborzi, H.; Samet, H. Efficient query processing on spatial networks. In Proceedings of the International Workshop on Geographic Information Systems, Bremen, Germany, 4–5 November 2005.
4. Sankaranarayanan, J.; Samet, H.; Alborzi, H. Path oracles for spatial networks. *PVLDB* **2009**, *2*, 1210–1221. [CrossRef]
5. Wu, L.; Xiao, X.; Deng, D.; Cong, G.; Zhu, A.D.; Zhou, S. Shortest path and distance queries on road networks: An experimental evaluation. *PVLDB* **2012**, *5*, 406–417. [CrossRef]
6. Zhang, D.; Yang, D.; Wang, Y.; Tan, K.-L.; Cao, J.; Shen, H.T. Distributed shortest path query processing on dynamic road networks. *VLDB J.* **2017**, *26*, 399–419. [CrossRef]
7. 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]
8. D'Angelo, G.; D'Emidio, M.; Frigioni, D. Distance queries in large-scale fully dynamic complex networks. In Proceedings of the International Workshop on Combinatorial Algorithms, Helsinki, Finland, 17–19 August 2016.
9. Sankaranarayanan, J.; Samet, H. Query processing using distance oracles for spatial networks. *IEEE Trans. Knowl. Data Eng.* **2010**, *22*, 1158–1175. [CrossRef]
10. Papadias, D.; Zhang, J.; Mamoulis, N.; Tao, Y. Query processing in road network databases. In Proceedings of the International Conference on Very Large Data Bases, Berlin, Germany, 9–12 September 2003.
11. Abeywickrama, T.; Cheema, M.A.; Taniar, D. k-Nearest neighbors on road networks: A journey in experimentation and in-memory implementation. *PVLDB* **2016**, *9*, 492–503. [CrossRef]
12. 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. *PVLDB* **2018**, *11*, 594–606.
13. Chaudhuri, S.; Ganti, V.; Kaushik, R. A primitive operator for similarity joins in data cleaning. In Proceedings of the International Conference on Data Engineering, Atlanta, GA, USA, 3–8 April 2006.
14. Deng, D.; Li, G.; Hao, S.; Wang, J.; Feng, J. MassJoin: A mapreduce-based method for scalable string similarity joins. In Proceedings of the International Conference on Data Engineering, Chicago, IL, USA, 31 March–4 April 2014.
15. Metwally, A.; Faloutsos, C. V-SMART-Join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. *PVLDB* **2012**, *5*, 704–715. [CrossRef]
16. Sarma, A.D.; He, Y.; Chaudhuri, S. ClusterJoin: A similarity joins framework using map-reduce. *PVLDB* **2014**, *7*, 1059–1070.
17. Vernica, R.; Carey, M.J.; Li, C. Efficient parallel set-similarity joins using mapreduce. In Proceedings of the International Conference on Management of Data, Indianapolis, IN, USA, 6–10 June 2010.
18. Wang, Y.; Metwally, A.; Parthasarathy, S. Scalable all-pairs similarity search in metric spaces. In Proceedings of the International Conference on Knowledge Discovery and Data Mining, Chicago, IL, USA, 11–14 August 2013.
19. Hjaltason, G.R.; Samet, H. Incremental distance join algorithms for spatial databases. In Proceedings of the International Conference on Management of Data, Seattle, WA, USA, 2–4 June 1998.
20. Shin, H.; Moon, B.; Lee, S. Adaptive and incremental processing for distance join queries. *IEEE Trans. Knowl. Data Eng.* **2003**, *15*, 1561–1578. [CrossRef]
21. 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 Spatial and Temporal Databases, Munich, Germany, 21–23 August 2013.
22. 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.
23. Sankaranarayanan, J.; Alborzi, H.; Samet, H. Distance join queries on spatial networks. In Proceedings of the International Symposium on Geographic Information Systems, Arlington, VA, USA, 10–11 November 2006.
24. Arain, Q.A.; Deng, Z.; Memon, I.; Zubedi, A.; Jiao, J.; Ashraf, A.; Khan, M.S. Privacy protection with dynamic pseudonym-based multiple mix-zones over road networks. *China Commun.* **2017**, *14*, 89–100. [CrossRef]
25. Arain, Q.A.; Memon, I.; Deng, Z.; Memon, M.H.; Mangi, F.A.; Zubedi, A. Location monitoring approach: Multiple mix-zones with location privacy protection based on traffic flow over road networks. *Multimedia Tools Appl.* **2018**, *77*, 5563–5607. [CrossRef]

26. Arain, Q.A.; Uqaili, M.A.; Deng, Z.; Memon, I.; Jiao, J.; Shaikh, M.A.; Zubedi, A.; Ashraf, A.; Arain, U.A. Clustering based energy efficient and communication protocol for multiple mix-zones over road networks. *Wirel. Pers. Commun.* **2017**, *95*, 411–428. [CrossRef]
27. Domenic, M.K.; Wang, Y.; Zhang, F.; Memon, I.; Gustav, Y.H. Preserving users' privacy for continuous query services in road networks. In Proceedings of the International Conference on Information Management, Innovation Management and Industrial Engineering, Xi'an, China, 23–24 November 2013.
28. Gustav, Y.H.; Wang, Y.; Domenic, M.K.; Zhang, F.; Memon, I. Velocity similarity anonymization for continuous query Location based services. In Proceedings of the International Conference on Computational Problem-Solving, Jiuzhai, China, 26–28 October 2013.
29. Kamenyi, D.M.; Wang, Y.; Zhang, F.; Memon, I. Authenticated privacy preserving for continuous query in location based services. *J. Comput. Inform. Syst.* **2013**, *9*, 9857–9864.
30. Memon, I. Distance and clustering-based energy-efficient pseudonyms changing strategy over road network. *Int. J. Commun. Syst.* **2018**, *31*, 1–22. [CrossRef]
31. Memon, I. Authentication user's privacy: An integrating location privacy protection algorithm for secure moving objects in location based services. *Wirel. Pers. Commun.* **2015**, *82*, 1585–1600. [CrossRef]
32. Memon, I.; Arain, Q.A. Dynamic path privacy protection framework for continuous query service over road networks. *World Wide Web* **2017**, *20*, 639–672. [CrossRef]
33. Memon, I.; Arain, Q.A.; Zubedi, A.; Mangi, F.A. DPMM: Dynamic pseudonym-based multiple mix-zones generation for mobile traveler. *Multimedia Tools Appl.* **2017**, *76*, 24359–24388. [CrossRef]
34. Memon, I.; Chen, L.; Arain, Q.A.; Memon, H.; Chen, G. Pseudonym changing strategy with multiple mix zones for trajectory privacy protection in road networks. *Int. J. Commun. Syst.* **2018**, *31*, 1–44. [CrossRef]
35. Ali, M.E.; Tanin, E.; Zhang, R.; Kulik, L. A motion-aware approach for efficient evaluation of continuous queries on 3D object databases. *VLDB J.* **2010**, *19*, 603–632. [CrossRef]
36. Giannikis, G.; Alonso, G.; Kossmann, D. SharedDB: Killing one thousand queries with one stone. *PVLDB* **2012**, *5*, 526–537. [CrossRef]
37. 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.
38. Zhang, D.; Chow, C.-Y.; Li, Q.; Zhang, X.; Xu, Y. SMashQ: Spatial mashup framework for k-nn queries in time-dependent road networks. *Distrib. Parall. Databases* **2013**, *31*, 259–287. [CrossRef]
39. Brinkhoff, T.; Kriegel, H.-P.; Seeger, B. Efficient processing of spatial joins using r-trees. In Proceedings of the International Conference on Management of Data, Washington, DC, USA, 26–28 May 1993.
40. Chen, C.; Sun, W.; Zheng, B.; Mao, D.; Liu, W. An incremental approach to closest pair queries in spatial networks using best-first search. In Proceedings of the International Conference on Database and Expert Systems Applications, Toulouse, France, 29 August–2 September 2011.
41. Koudas, N.; Sevcik, K.C. High dimensional similarity joins: Algorithms and performance evaluation. *IEEE Trans. Knowl. Data Eng.* **2000**, *12*, 3–18. [CrossRef]
42. Makreshanski, D.; Giannikis, G.; Alonso, G.; Kossmann, D. MQJoin: Efficient shared execution of main-memory joins. *PVLDB* **2016**, *9*, 480–491. [CrossRef]
43. 9th DIMACS Implementation Challenge: Shortest Paths. Available online: http://www.dis.uniroma1.it/challenge9/download.shtml (accessed on 15 June 2018).
44. Corral, A.; Manolopoulos, Y.; Theodoridis, Y.; Vassilakopoulos, M. Multi-way distance join queries in spatial databases. *GeoInformatica* **2004**, *8*, 373–402. [CrossRef]
45. Gao, Y.; Shou, L.; Chen, K.; Chen, G. Aggregate farthest-neighbor queries over spatial data. In Proceedings of the International Conference on Database Systems for Advanced Applications, Hong Kong, China, 22–25 April 2011.
46. Wang, H.; Zheng, K.; Su, H.; Wang, J.; Sadiq, S.; Zhou, X. Efficient aggregate farthest neighbour query processing on road networks. In Proceedings of the Australasian Database Conference, Brisbane, Australia, 14–16 July 2014.

© 2018 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).