Hindawi
Mobile Information Systems
Volume 2018, Article ID 1243289, 17 pages
https://doi.org/10.1155/2018/1243289

Research Article

Efficient Shared Execution Processing of *k*-Nearest Neighbor Joins in Road Networks

Hyung-Ju Cho

Department of Software, Kyungpook National University, 2559, Gyeongsang-daero, Sangju-si, Gyeongsangbuk-do 37224, Republic of Korea

Correspondence should be addressed to Hyung-Ju Cho; hyungju@knu.ac.kr

Received 22 September 2017; Revised 12 January 2018; Accepted 12 February 2018; Published 12 April 2018

Academic Editor: Jinglan Zhang

Copyright © 2018 Hyung-Ju Cho. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We investigate the *k*-nearest neighbor (*k*NN) join in road networks to determine the *k*-nearest neighbors (NNs) from a dataset *S* to every object in another dataset *R*. The *k*NN join is a primitive operation and is widely used in many data mining applications. However, it is an expensive operation because it combines the *k*NN query and the join operation, whereas most existing methods assume the use of the Euclidean distance metric. We alternatively consider the problem of processing *k*NN joins in road networks where the distance between two points is the length of the shortest path connecting them. We propose a shared execution-based approach called the group-nested loop (GNL) method that can efficiently evaluate *k*NN joins in road networks by exploiting grouping and shared execution. The GNL method can be easily implemented using existing *k*NN query algorithms. Extensive experiments using several real-life roadmaps confirm the superior performance and effectiveness of the proposed method in a wide range of problem settings.

1. Introduction

Road networks are often represented as weighted undirected graphs by placing a graph vertex at each road intersection or terminus and connecting vertices by edges that represent each segment of a road between two vertices [1–5]. The distance between two points in a road network is the length of the shortest path between them. In this study, we investigate the *k*-nearest neighbor (*k*NN) join in road networks, which combines each object in a dataset with the *k* objects in another dataset that are closest to it [6–8]. The *k*NN join is a primitive operation, which is widely used in many data mining and analytic applications, such as *k*NN classification, *k*-means clustering, sample assessment and sample post processing, missing value imputation, and *k*-distance diagrams [6–12].

Figure 1 presents an example of a *k*NN join, which we denote as $R \ltimes_{kNN} S$, in a road network where $R = \{r_1, r_2, r_3, r_4\}$ and $S = \{s_1, s_2, s_3, s_4, s_5\}$. For convenience, we assume that outer objects r_1 through r_4 denote hotels represented by rectangles and inner objects s_1 through s_5 denote tourist attractions represented by triangles. In this example, we can consider a *k*NN join query for tourists, which could be "find ordered pairs of two tourist attractions closest to each hotel." Outer objects and inner objects typically correspond to points of interest in different categories such as hotels, tourist attractions, and hospitals.

The *k*NN join is an expensive operation because it combines the *k*NN query and the join operation. A simple solution to the *k*NN join $R \ltimes_{kNN} S$ of datasets *R* and *S* requires scanning *S* once for each object in *R* while computing the distance between each pair of objects from *R* and *S*. This simple solution is not achievable for large datasets due to the complexity of $O(|R| \times |S|)$. Therefore, many studies have been performed to improve the efficiency of the *k*NN join [6–12]. Most of these previous studies focused on processing the *k*NN join in the Euclidean space, mainly by designing elegant indexing techniques to avoid scanning the entire dataset repeatedly and to prune as many distance computations as possible. However, few studies have considered processing the *k*NN join in road networks.

In this study, we address the problem of implementing the *k*NN join operator in road networks by proposing a shared execution-based approach called the group-nested loop (GNL) method, which comprises the following three steps.

In the first step, the outer objects in a dataset R are grouped and then converted into a set of outer segments, where each outer segment connects adjacent outer objects. A method for grouping adjacent outer objects into an outer segment is presented in Section 4.1. In the second step, at most two kNN queries are evaluated for the outer segment. In the last step, the kNN join operation based on shared execution is performed for the outer objects in the outer segment. The GNL method is efficient for the following reasons. (1) It inherits the strength of shared execution processing by avoiding the evaluation of redundant kNN queries. (2) It can effectively group neighboring outer objects and consider them as a whole while pruning away unpromising network traversals. (3) It does not require any materialized structures, such as precomputing network Voronoi polygons [13, 14] or precomputing the network distance between every pair of vertices [15]. (4) It can be implemented easily using existing kNN query algorithms (e.g., INE [16], DisBrw [17], ROAD [2], and G-tree [5]), which is highly desirable in practice.

We summarize the main contributions of this study as follows:

(i) We propose the GNL method for processing kNN joins efficiently in road networks. To the best of our knowledge, this is the first attempt to evaluate kNN joins efficiently in road networks.

(ii) The GNL method is intuitive and straightforward to implement, thereby allowing its simple integration with existing kNN query processing methods [1, 2, 5, 13, 16–19]. The GNL method employs an optimized number of kNN queries to evaluate kNN joins while exploiting grouping and shared execution.

(iii) We conduct extensive experiments with different setups to demonstrate the superior performance of the GNL method compared with conventional solutions.

The remainder of this paper is organized as follows: In Section 2, we review related research. In Section 3, we provide some background knowledge. In Section 4, we present the basic GNL method for processing kNN joins in road networks. In Section 5, we present the GNL method to avoid redundant kNN queries for processing kNN joins efficiently in road networks. In Section 6, we present empirical comparisons of the GNL method and conventional solutions with different setups. Finally, we discuss our conclusions in Section 7.

2. Related Work

2.1. kNN Search in Road Networks. The processing of kNN queries in road networks has been studied extensively [2, 3, 5, 16, 17, 19, 20]. Papadias et al. [16] introduced incremental Euclidean restriction (IER) and incremental network expansion (INE). IER exploits the Euclidean restriction principle in road networks to achieve better performance. INE conducts network expansion from the query location in a similar manner to Dijkstra's algorithm and examines the data objects in the sequence in which they are encountered. Shahabi et al. [3] developed an embedding technique for transforming a road network to a constraint-free high-dimensional Euclidean space to approximately retrieve the nearest objects using traditional Euclidean-based algorithms. Kolahdouzan and Shahabi [13, 14] utilized the first degree network Voronoi diagrams to partition the spatial network into network Voronoi polygons (NVP), with one for each data object. They indexed the NVPs by using a spatial access method to reduce the problem to a point location problem in the Euclidean space. This minimizes the online network distance computation by allowing precomputing of the NVPs. Huang et al. [18] addressed the same problem using the *island* approach where each vertex is associated with all the data points. These islands are considered centers with a given radius r covering the vertex. In their approach, they utilized restricted network expansion from the query point using the precomputed islands.

Huang et al. [20] and Samet et al. [17] proposed two different algorithms to address the drawbacks of data object-dependent precomputation. Huang et al. [20] introduced S-GRID for partitioning the spatial network into disjoint subnetworks and precomputing the shortest path for each pair of connected border points. To find the k-nearest neighbors (NNs), network expansion is first performed within the subnetworks before outer expansion between the border points using the precomputed information. Samet et al. [17] proposed a distance browsing (DisBrw) method where they associate a label with each edge to represent all of the vertices with a shortest path starting with a particular edge. They use these labels to traverse the shortest path quadtrees that facilitate geometric pruning to find the network distance between objects.

Lee et al. [2] proposed a new system framework called ROAD for processing location-dependent spatial queries. ROAD organizes a large road network as a hierarchy of interconnected regional subnetworks, each of which is augmented with shortcuts and object abstracts to accelerate network traversal and facilitate rapid object lookups, respectively. Inspired by the R-tree [21], Zhong et al. [5] proposed a height-balanced index called G-tree in road networks, which employs graph partitioning to efficiently compute the network distances through a hierarchy of subgraphs. Abeywickrama et al. [1] performed a thorough experimental evaluation of several kNN algorithms for road networks, where they showed that G-tree [5] typically outperformed INE [16], DisBrw [17], and ROAD [2], but not in all cases, thereby demonstrating the impact of an efficient implementation. Finally, other previous studies [15, 16] also processed the distance joins in road networks. For example, an ordered distance join operation returns a set of object pairs, which are reported in increasing order of the distance between the pairs. However, it is not appropriate to extend the existing solutions to solve our problem due to differences in problem definition.

2.2. kNN Join in the Euclidean Space. The processing of kNN join queries in the Euclidean space has been studied extensively [6–10, 12]. Böhm and Krebs [6] developed an

R-tree-based method called a multipage index (MuX) to evaluate kNN joins in the Euclidean space. The MuX method organizes the input datasets with large-sized pages in order to reduce the I/O cost. The computational cost is further reduced by carefully designing a secondary structure with a much smaller size within the pages. Xia et al. [7] developed a grid partitioning-based approach called Gorder (an order based on a grid) to evaluate kNN joins in the Euclidean space. Gorder is a block-nested loop join method that employs sorting, join scheduling, and distance computation filtering and reduction to decrease both the I/O and CPU costs. Yu et al. [8] developed an index-based kNN join method called iJoin by using iDistance [22] as the underlying index structure. By splitting the two input datasets into individual sets of partitions, the iJoin method employs a B^+-tree to hold the objects in each dataset using the iDistance technique and it evaluates the kNN joins based on the properties of the B^+-tree. Yao et al. [10] developed z-kNN as a Z-order-based method to evaluate kNN joins in large relational databases without changing the database engine, thereby allowing the query optimizer to understand and generate the best query plan. The z-kNN method transforms the kNN join operation into a set of kNN search operations where each object in R is a query point. Recently, Lu et al. [9] and Zhang et al. [12] developed novel algorithms using MapReduce to efficiently perform parallel kNN joins on large datasets. However, due to the different problem environments, it is not possible to apply these solutions based on the Euclidean distance to the kNN join problem in road networks. Finally, Li and Taniar [23] presented a taxonomy for distance-based spatial join queries, which are divided into the following three main categories: (1) all-range join (e.g., [24, 25]), (2) all-kNN join (e.g., [6, 26]), and (3) all-reverse NN join. Our study belongs to the all-kNN join category, and to the best of our knowledge, our current study represents the first attempt to evaluate kNN joins efficiently in road networks.

3. Preliminaries

In Section 3.1, we formally define the kNN join problem in road networks and explain its properties. In Section 3.2, we define the terms and notations used in this study.

3.1. kNN Join. We consider outer objects and inner objects in a road network G where the outer objects and inner objects typically correspond to points of interest in different categories as shown in Figure 1. Given two points p and q, $\mathrm{dist}(p, q)$ denotes the network distance between p and q, which is the length of the shortest path connecting them in G.

Definition 1. (kNN search): Given an integer k, an outer object r, and a set of inner objects S, the kNNs of r from S, denoted as S_k^r, constitute a set of k inner objects from S such that $\mathrm{dist}(r, s^+) \leq \mathrm{dist}(r, s^-)$ holds for $\forall s^+ \in S_k^r$ and $\forall s^- \in S - S_k^r$.

Definition 2. (kNN join): Given an integer k and datasets R and S, the kNN join of R and S, denoted as $R \ltimes_{k\mathrm{NN}} S$, returns ordered pairs of two objects r and s such that s is one of the kNNs of r where $r \in R$ and $s \in S$. For simplicity, $R \ltimes_{k\mathrm{NN}} S$ is abbreviated as $R \ltimes S$. Formally,

$$R \ltimes S = \left\{ \langle r, s \rangle \middle| \forall r \in R, \forall s \in S_k^r \right\}. \quad (1)$$

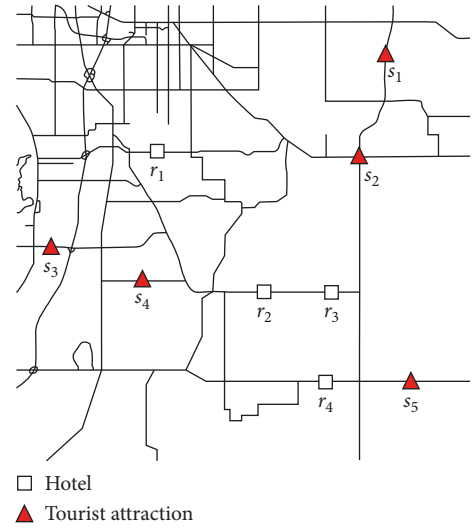


Figure 1: Example of kNN join $R \ltimes_{k\mathrm{NN}} S$ in a road network where $R = \{r_1, r_2, r_3, r_4\}$ and $S = \{s_1, s_2, s_3, s_4, s_5\}$.

The kNN join has the following properties:

(i) The kNN join is not commutative, that is, $R \ltimes S \neq S \ltimes R$. Without loss of generality, we assume that $R \ltimes S$ and $k \leq |S|$ in this study.

(ii) The cardinality of the result set of a kNN join is $|R \ltimes S| = k \times |R|$ because the kNN join returns the k inner objects that are closest to each outer object in R.

(iii) The traversal distances from an outer object to its kNNs are not known in advance, in contrast to the range join.

3.2. Definition of Terms and Notations

3.2.1. Road Network. A road network can be modeled using a weighted undirected graph $G = \langle V, E, W \rangle$, where V, E, and W indicate the vertex set, edge set, and edge distance matrix, respectively. Each edge has a nonnegative weight to represent the network distance.

3.2.2. Classification of Vertices. Vertices can be divided 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.

3.2.3. Vertex Sequence and Outer Segment. A vertex sequence $\overline{v_l v_{l+1} \cdots v_m}$ denotes a path between two vertices, v_l and v_m, such that v_l and v_m are either an intersection vertex or a terminal vertex, and the other vertices in the path, $v_{l+1}, \ldots, v_{m-1}$, are intermediate vertices. The length of

Table 1: Symbols and their meanings.

Symbol	Definition
k	Number of requested NNs
r	Outer object $r \in R$
s	Inner object $s \in S$
$\overline{v_l v_{l+1} \cdots v_m}$	Vertex sequence where v_l and v_m are either an intersection vertex or a terminal vertex and the other vertices, $v_{l+1}, \ldots, v_{m-1}$, are intermediate vertices
$\overline{r_i r_{i+1} \cdots r_j}$	Outer segment generated from outer objects $r_i, r_{i+1}, \ldots, r_j$ in the same vertex sequence
S_k^r	Set of k inner objects closest to outer object r from S
$S(\overline{r_i r_j})$	Set of inner objects located in outer segment $\overline{r_i r_j}$, that is, $S(\overline{r_i r_j}) = \{s \mid s \in \overline{r_i r_j}\}$
$\mathrm{dist}(p, q)$	Length of the shortest path connecting two points p and q
$\mathrm{len}(p, q)$	Length of the segment connecting two points p and q such that p and q are located in the same vertex sequence

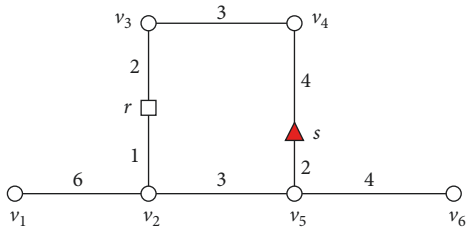


Figure 2: $\mathrm{dist}(r, s) = 6$ and $\mathrm{len}(r, s) = 9$.

a vertex sequence is the total weight of the edges in the vertex sequence. An outer segment $\overline{r_i r_{i+1} \cdots r_j}$ denotes a path that connects outer objects $r_i, r_{i+1}, \ldots, r_j$ in the same vertex sequence. Table 1 summarizes the notations used in the study. To simplify the presentation, we use $\overline{r_i r_j}$ to denote $\overline{r_i r_{i+1} \cdots r_j}$ if it causes no confusion, where $r_i, r_{i+1}, \ldots, r_j$ are outer objects in the same vertex sequence.

Figure 2 shows the difference in the distance and segment length between two objects r and s in a road network, where the numbers on the edges indicate the distance between two adjacent points (e.g., $\mathrm{dist}(v_1, v_2) = 6$). The shortest path from r to s is $r \to v_2 \to v_5 \to s$, and thus the distance between them is $\mathrm{dist}(r, s) = 6$, whereas the segment connecting r and s in the same vertex sequence $\overline{v_2 v_3 v_4 v_5}$ becomes $\overline{r v_3 v_4 s}$, and thus its length is $\mathrm{len}(r, s) = 9$. We recall that $\mathrm{len}(r, s)$ is defined if and only if the two objects are located in the same vertex sequence.

4. Basic GNL Method for kNN Joins in Road Networks

In Section 4.1, we describe the grouping of outer objects in a vertex sequence. In Section 4.2, we explain the shared execution processing of outer objects in a segment. In Section 4.3, we provide algorithms for processing kNN joins efficiently in road networks. Finally, in Section 4.4, we discuss the evaluation of an example kNN join in a road network.

4.1. Grouping of Outer Objects in a Vertex Sequence. Figure 3 presents an example of the kNN join $R \ltimes S$ in a road network that we consider in this section, where there are six outer objects, r_1 through r_6, and four inner objects, s_1 through s_4, in a road network, with three vertex sequences, $\overline{v_1 v_2 v_3}$, $\overline{v_1 v_3}$, and $\overline{v_1 v_4 v_3}$. For simplicity, we consider a 2-NN join $R \ltimes S$ where $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ and $S = \{s_1, s_2, s_3, s_4\}$.

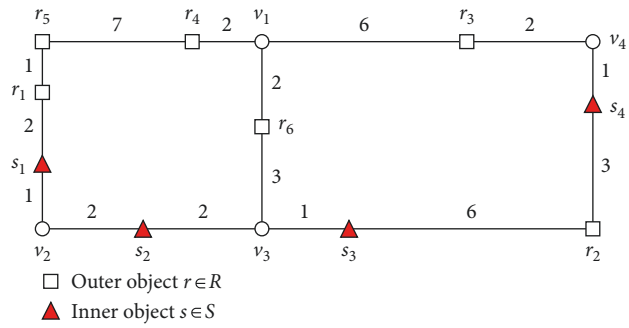


Figure 3: Example of kNN join $R \ltimes S$ in a road network.

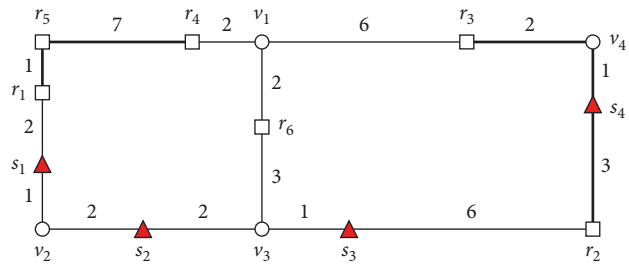


Figure 4: Grouping outer objects in a vertex sequence.

Figure 4 shows a sample grouping of outer objects in a vertex sequence, where the outer objects r_1, r_4, and r_5 in vertex sequence $\overline{v_1 v_2 v_3}$ are grouped into $\overline{r_1 r_5 r_4}$ and the outer objects r_2 and r_3 in vertex sequence $\overline{v_1 v_4 v_3}$ are grouped into $\overline{r_2 r_3}$. Note that the outer segments (e.g., $\overline{r_1 r_5 r_4}$ and $\overline{r_2 r_3}$ in Figure 4) are marked by bold lines. Therefore, a set of outer objects $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ is transformed into $\overline{R} = \{\overline{r_1 r_5 r_4}, \overline{r_2 r_3}, r_6\}$, where $\overline{R}$ denotes the set of outer segments generated from the outer objects in R.

Algorithm 1 describes how to group neighboring outer objects into an outer segment in a vertex sequence. This algorithm comprises two steps. In the first step, a set of vertex sequences $\overline{V}$ is generated from a set of vertices V and a set of edges E. In the second step, a set of outer segments $\overline{R}$ is generated from a set of outer objects R and $\overline{V}$. We first examine each vertex to find either an intersection vertex or a terminal vertex. If v_l is either an intersection vertex or a terminal vertex, for each edge $\overline{v_l v_{l+1}} \in E$ adjacent to v_l, we explore the path from v_l toward v_{l+1} until a nonintermediate

Input: R: set of outer objects, V: set of vertices, E: set of edges
Output: $\overline{R}$: set of outer segments

(1) $\overline{V} \leftarrow \varnothing$, $\overline{R} \leftarrow \varnothing$ // $\overline{V}$ and $\overline{R}$ are initialized to the empty sets, where $\overline{V}$ is a set of vertex sequences
(2) // step 1: $\overline{V}$ is generated from V and E
(3) **for** each vertex $v_l \in V$ **do**
(4) **if** v_l is either an intersection vertex or a terminal vertex **then**
(5) **for** each edge $\overline{v_l v_{l+1}} \in E$ adjacent to v_l **do**
(6) $\overline{v_l v_{l+1} \cdots v_m} \leftarrow$ find_vertex_sequence(v_l, v_{l+1}, V) // it explores the path from v_l toward v_{l+1} until v_m is found
(7) $\overline{V} \leftarrow \overline{V} \cup \{\overline{v_l v_{l+1} \cdots v_m}\}$ // v_m is either an intersection vertex or a terminal vertex
(8) // step 2: $\overline{R}$ is generated from R and $\overline{V}$
(9) **for** each vertex sequence $\overline{v_l v_{l+1} \cdots v_m} \in \overline{V}$ **do**
(10) $\{r_i, r_{i+1}, \ldots, r_j\} \leftarrow$ find_outer_objects_in_vertex_sequence$(\overline{v_l v_{l+1} \cdots v_m})$
(11) $\overline{R} \leftarrow \overline{R} \cup \{\overline{r_i r_{i+1} \cdots r_j}\}$ // $\overline{r_i r_{i+1} \cdots r_j}$ is generated from outer objects $r_i, r_{i+1}, \ldots, r_j$
(12) return $\overline{R}$

Algorithm 1: Group_outer_objects (R, V, E).

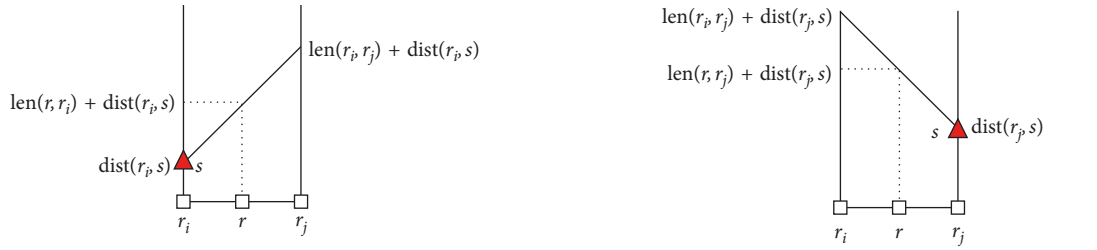


Figure 5: $\text{dist}(r,s) = \begin{cases} \min\{\text{len}(r,r_i) + \text{dist}(r_i,s), \text{len}(r,r_j) + \text{dist}(r_j,s)\}, & \text{if } s \notin \overline{r_i r_j} \\ \min\{\text{len}(r,r_i) + \text{dist}(r_i,s), \text{len}(r,r_j) + \text{dist}(r_j,s), \text{len}(r,s)\}, & \text{otherwise.} \end{cases}$

vertex v_m is found. Then, a new vertex sequence $\overline{v_l v_{l+1} \cdots v_m}$ is added to $\overline{V}$. We search for outer objects $r_i, r_{i+1}, \ldots, r_j$ in each vertex sequence $\overline{v_l v_{l+1} \cdots v_m}$. The outer objects $r_i, r_{i+1}, \ldots, r_j$ are grouped into an outer segment $\overline{r_i r_{i+1} \cdots r_j}$, which is added to $\overline{R}$.

4.2. Shared Execution Processing of Outer Objects in a Segment. The shared execution strategies in the basic GNL method are motivated by the observation that two kNN queries at most are sufficient to retrieve the kNNs of all outer objects in an outer segment. This observation is formalized in Lemma 1, which states a simple but important fact regarding the shared execution processing of outer objects in an outer segment. If we evaluate two kNN queries for outer objects r_i and r_j, which are located at the ends of an outer segment $\overline{r_i r_{i+1} \cdots r_j}$, then we can determine the kNNs of the other outer objects $r_{i+1}, \ldots, r_{j-1}$ without evaluating additional kNN queries for them. Thus, the two kNN queries for r_i and r_j are sufficient to retrieve the kNNs of the other outer objects $r_{i+1}, \ldots, r_{j-1}$ in a segment.

Lemma 1. *For every outer object $r \in \overline{r_i r_j}$, it holds that $S_k^r \subseteq S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$, where $S_k^{r_i}$ ($S_k^{r_j}$) is the set of k inner objects closest to outer object r_i (r_j) and $S(\overline{r_i r_j})$ is the set of inner objects located in the outer segment $\overline{r_i r_j}$* (e.g., $s_4 \in \overline{r_2 r_3}$ *in Figure* 4).

Proof. We prove Lemma 1 by contradiction. If we assume that $S_k^r \subseteq S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$ does not hold, i.e., $S_k^r \nsubseteq$

Table 2: Computation of $\mathrm{dist}(r, s)$ for $r \in \overline{r_i r_j}$ and $s \in S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$.

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

Input: k: number of inner objects retrieved for an outer object, R: set of outer objects, S: set of inner objects
Output: $R \ltimes S$: set of ordered pairs of outer object $r \in R$ and inner object $s \in S_k^r$
(1) $\Psi \leftarrow \varnothing$ // kNN join result set Ψ is initialized to the empty set
(2) // step 1: outer objects $r_i, r_{i+1}, \ldots, r_j$ in a vertex sequence are grouped into an outer segment $\overline{r_i r_j}$
(3) $\overline{R} \leftarrow$ group_outer_objects(R, V, E) // group_outer_objects is explained in Algorithm 1
(4) // step 2: kNN join is evaluated for each outer segment $\overline{r_i r_j} \in \overline{R}$
(5) **for** each outer segment $\overline{r_i r_j} \in \overline{R}$ **do**
(6) $\Psi(\overline{r_i r_j}) \leftarrow kNN_join(k, \overline{r_i r_j}, S)$ // kNN_join is explained in Algorithm 3
(7) $\Psi \leftarrow \Psi \cup \Psi(\overline{r_i r_j})$ // $\Psi(\overline{r_i r_j}) = \{\langle r, s\rangle | \forall r \in \overline{r_i r_j}, \forall s \in S_k^r\}$
(8) **return** Ψ // $\Psi = \{\langle r, s\rangle | \forall r \in R, \forall s \in S_k^r\}$ is returned after all outer segments have been processed

Algorithm 2: Basic_GNL (k, R, S).

$S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$, then this implies that there is an inner object $s^+ \in S_k^r$ such that $s^+ \notin S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$. Clearly, $s^+ \notin S_k^{r_i}$, so s^+ is more distant from r_i than its kth closest inner object $s_{k\mathrm{th}}^{r_i}$, that is, $\mathrm{dist}(r_i, s_{k\mathrm{th}}^{r_i}) < \mathrm{dist}(r_i, s^+)$. Similarly, $s^+ \notin S_k^{r_j}$, so s^+ is also more distant from r_j than its kth closest inner object $s_{k\mathrm{th}}^{r_j}$, that is, $\mathrm{dist}(r_j, s_{k\mathrm{th}}^{r_j}) < \mathrm{dist}(r_j, s^+)$. However, $s^+ \notin S(\overline{r_i r_j})$, so s^+ is not located in $\overline{r_i r_j}$. 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 $\mathrm{dist}(r, s^+) = \min\{\mathrm{len}(r, r_i) + \mathrm{dist}(r_i, s^+), \mathrm{len}(r, r_j) + \mathrm{dist}(r_j, s^+)\}$. From the former conditions, $\mathrm{dist}(r, s^+) > \min\{\mathrm{len}(r, r_i) + \mathrm{dist}(r_i, s_{k\mathrm{th}}^{r_i}), \mathrm{len}(r, r_j) + \mathrm{dist}(r_j, s_{k\mathrm{th}}^{r_j})\}$, so s^+ cannot belong to S_k^r, which contradicts the assumption that an inner object $s^+ \in S_k^r$ exists such that $s^+ \notin S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$.

We determine the k inner objects that are closest to an outer object $r \in \overline{r_i r_j}$ by using two kNN sets $S_k^{r_i}$ and $S_k^{r_j}$ of outer objects r_i and r_j, respectively. First, we investigate the distance from the outer object $r \in \overline{r_i r_j}$ to an inner object s. In Figure 5, we assume that r_i corresponds to the origin of the XY coordinate system. Then, the Y-axis represents $\mathrm{dist}(r, s)$ and the X-axis represents $\mathrm{len}(r_i, r)$. If there is a path of $r \to r_i \to s$, then the distance from r to s is computed as $\mathrm{dist}(r, s) = \mathrm{len}(r, r_i) + \mathrm{dist}(r_i, s)$, as shown in Figure 5(a). Similarly, if there is a path of $r \to r_j \to s$, then $\mathrm{dist}(r, s)$ is computed as $\mathrm{dist}(r, s) = \mathrm{len}(r, r_j) + \mathrm{dist}(r_j, s)$, as shown in Figure 5(b). If an inner object s is located in $\overline{r_i r_j}$, then $\mathrm{dist}(r, s)$ is computed as $\mathrm{dist}(r, s) = \mathrm{len}(r, s)$, as shown in Figure 5(c). Note that min returns the minimum of the values in the input array. Thus, $\mathrm{dist}(r, s)$ is the length of the shortest path among multiple paths between r and s, and it is computed as follows:

$$\mathrm{dist}(r,s) = \begin{cases} \min\{\mathrm{len}(r, r_i) + \mathrm{dist}(r_i, s), \mathrm{len}(r, r_j) + \mathrm{dist}(r_j, s)\}, & \text{if } s \notin \overline{r_i r_j} \\ \min\{\mathrm{len}(r, r_i) + \mathrm{dist}(r_i, s), \mathrm{len}(r, r_j) + \mathrm{dist}(r_j, s), \mathrm{len}(r, s)\}, & \text{otherwise.} \end{cases} \quad (2)$$

Table 2 explains how to compute the distance from r to s, where $r \in \overline{r_i r_j}$ and $s \in S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$. An inner object s belongs to a combination of $S_k^{r_i}$, $S_k^{r_j}$, 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 is located in $\overline{r_i r_j}$ compared with retrieving a set $S_k^{r_i}$ ($S_k^{r_j}$) of k inner objects closest to outer object r_i (r_j). Note that $\mathrm{dist}(r, s)$ in Table 2 may not necessarily be the length of the shortest path from r to s, as discussed in Section 4.4.

4.3. kNN Join Algorithm. Algorithm 2 describes the basic GNL algorithm, which employs the grouping of outer objects and shared execution to reduce the processing time.

Input: k: number of inner objects retrieved for an outer object, $\overline{r_i r_j}$: outer segment, S: set of inner objects
Output: $\overline{r_i r_j} \ltimes S$: set of ordered pairs of outer object $r \in \overline{r_i r_j}$ and inner object $s \in S_k^r$

(1) **if** $|\overline{r_i r_j}| = 1$ **then** // $|\overline{r_i r_j}| = 1$ denotes that $\overline{r_i r_j}$ contains only an outer object r_i, that is, $r_i = r_j$
(2) $S_k^{r_i} \leftarrow k\text{NN\_query}(k, r_i)$ // kNN query from r_i is performed and its result is saved to $S_k^{r_i}$
(3) $\Psi(r_i) \leftarrow \{\langle r_i, s\rangle | s \in S_k^{r_i}\}$ // a partial join result $\Psi(r_i)$ for r_i is generated from $S_k^{r_i}$
(4) **return** $\Psi(r_i)$ // a partial join result $\Psi(r_i)$ for r_i is returned
(5) **else if** $|\overline{r_i r_j}| = 2$ **then** // $|\overline{r_i r_j}| = 2$ denotes that $\overline{r_i r_j}$ contains only two outer objects r_i and r_j
(6) $S_k^{r_i} \leftarrow k\text{NN\_query}(k, r_i)$ // kNN query from r_i is performed, and its result is saved to $S_k^{r_i}$
(7) $\Psi(r_i) \leftarrow \{\langle r_i, s\rangle | s \in S_k^{r_i}\}$ // a partial join result $\Psi(r_i)$ for r_i is generated from $S_k^{r_i}$
(8) $S_k^{r_j} \leftarrow k\text{NN\_query}(k, r_j)$ // kNN query from r_j is performed, and its result is saved to $S_k^{r_j}$
(9) $\Psi(r_j) \leftarrow \{\langle r_j, s\rangle | s \in S_k^{r_j}\}$ // a partial join result $\Psi(r_j)$ for r_j is generated from $S_k^{r_j}$
(10) **return** $\Psi(r_i) \cup \Psi(r_j)$ // the union $\Psi(r_i) \cup \Psi(r_j)$ of the partial join results for r_i and r_j is returned
(11) **else if** $|\overline{r_i r_j}| \geq 3$ **then** // $|\overline{r_i r_j}| \geq 3$ denotes that $\overline{r_i r_j}$ contains more than three outer objects
(12) $S_k^{r_i} \leftarrow k\text{NN\_query}(k, r_i)$ // kNN query from r_i is performed, and its result is saved to $S_k^{r_i}$
(13) $\Psi(r_i) \leftarrow \{\langle r_i, s\rangle | s \in S_k^{r_i}\}$ // a partial join result $\Psi(r_i)$ for r_i is generated from $S_k^{r_i}$
(14) $S_k^{r_j} \leftarrow k\text{NN\_query}(k, r_j)$ // kNN query from r_j is performed, and its result is saved to $S_k^{r_j}$
(15) $\Psi(r_j) \leftarrow \{\langle r_j, s\rangle | s \in S_k^{r_j}\}$ // a partial join result $\Psi(r_j)$ for r_j is generated from $S_k^{r_j}$
(16) $S(\overline{r_i r_j}) \leftarrow \{s | s \in \overline{r_i r_j}\}$ // inner objects located in $\overline{r_i r_j}$ are retrieved
(17) **for** each outer object $r \in \{r_{i+1}, \ldots, r_{j-1}\}$ **do** // k NNs of outer objects $r_{i+1}, \ldots, r_{j-1}$ are retrieved from $S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$
(18) $S_k^r \leftarrow k\text{NN\_search}(k, r, S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j}))$ // kNN_search is explained in Algorithm 4
(19) $\Psi(r) \leftarrow \{\langle r, s\rangle | s \in S_k^r\}$ // a partial join result $\Psi(r)$ for $r \in \{r_{i+1}, \ldots, r_{j-1}\}$ is generated from S_k^r
(20) **return** $\Psi(r_i) \cup \Psi(r_{i+1}) \cup \cdots \cup \Psi(r_j)$ // the union of the partial join results for $r_i, r_{i+1}, \ldots, r_j$ is returned

Algorithm 3: kNN_join$(k, \overline{r_i r_j}, S)$.

Input: k: number of inner objects retrieved for an outer object, $r \in \overline{r_i r_j}$: outer object, $S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$: set of candidate inner objects
Output: S_k^r: set of k inner objects closest to r

(1) $S_k^r \leftarrow \varnothing$ // S_k^r is initialized to the empty set
(2) **for** each inner object $s \in S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$ **do**
(3) // step 1: $\text{dist}(r, s)$ is computed according to the condition of s, see Table 2 for details.
(4) **if** $s \in S_k^{r_i} \cap S_k^{r_j} \cap S(\overline{r_i r_j})$ **then** $\text{dist}(r, s) \leftarrow \min\{\text{len}(r, r_i) + \text{dist}(r_i, s), \text{len}(r, r_j) + \text{dist}(r_j, s), \text{len}(r, s)\}$
(5) **else if** $s \in S_k^{r_i} \cap S_k^{r_j} - S(\overline{r_i r_j})$ **then** $\text{dist}(r, s) \leftarrow \min\{\text{len}(r, r_i) + \text{dist}(r_i, s), \text{len}(r, r_j) + \text{dist}(r_j, s)\}$
(6) **else if** $s \in S_k^{r_i} \cap S(\overline{r_i r_j}) - S_k^{r_j}$ **then** $\text{dist}(r, s) \leftarrow \min\{\text{len}(r, r_i) + \text{dist}(r_i, s), \text{len}(r, s)\}$
(7) **else if** $s \in S_k^{r_j} \cap S(\overline{r_i r_j}) - S_k^{r_i}$ **then** $\text{dist}(r, s) \leftarrow \min\{\text{len}(r, r_j) + \text{dist}(r_j, s), \text{len}(r, s)\}$
(8) **else if** $s \in S_k^{r_i} - (S_k^{r_j} \cup S(\overline{r_i r_j}))$ **then** $\text{dist}(r, s) \leftarrow \text{len}(r, r_i) + \text{dist}(r_i, s)$
(9) **else if** $s \in S_k^{r_j} - (S_k^{r_i} \cup S(\overline{r_i r_j}))$ **then** $\text{dist}(r, s) \leftarrow \text{len}(r, r_j) + \text{dist}(r_j, s)$
(10) **else if** $s \in S(\overline{r_i r_j}) - (S_k^{r_i} \cup S_k^{r_j})$ **then** $\text{dist}(r, s) \leftarrow \text{len}(r, s)$
(11) // step 2: s is added to S_k^r if it satisfies either of the following two conditions:
(12) **if** $|S_k^r| < k$ **then**
(13) $S_k^r \leftarrow S_k^r \cup \{s\}$
(14) **else if** $|S_k^r| = k$ **and** $\text{dist}(r, s) < \text{dist}(r, s_{k\text{th}})$ **then**
(15) $S_k^r \leftarrow S_k^r \cup \{s\} - \{s_{k\text{th}}\}$
(16) **return** S_k^r

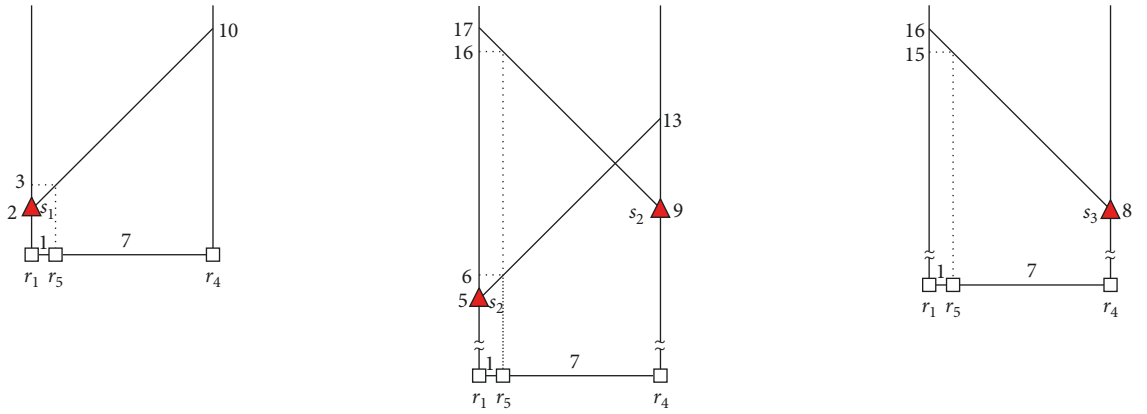
Algorithm 4: kNN_search$(k, r, S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j}))$.

This algorithm comprises two steps. In the first step, the outer objects $r_i, r_{i+1}, \ldots, r_j$ in a vertex sequence are grouped into an outer segment $\overline{r_i r_j}$ (line 3). In the second step, the kNNs of the outer objects in the outer segment $\overline{r_i r_j}$ are retrieved using the shared execution method (line 6), as explained in Algorithm 3. An ordered pair comprising an outer object $r \in \overline{r_i r_j}$ and inner object $s \in S_k^r$ is then added to the partial join result $\Psi(\overline{r_i r_j}) = \{\langle r, s\rangle | \forall r \in \overline{r_i r_j}, \forall s \in S_k^r\}$. Finally, the kNN join result set $\Psi = \{\langle r, s\rangle | \forall r \in R, \forall s \in S_k^r\}$ is returned after all the outer segments have been processed (line 8).

Algorithm 3 describes the kNN join for all outer objects $r_i, r_{i+1}, \ldots, r_j$ in $\overline{r_i r_j}$. Three general cases are formally

Table 3: Computation of the example kNN join $R \ltimes S$ using the basic GNL method.

$\overline{r_i r_j}$	r_i	r_j	$S_k^{r_i}$	$S_k^{r_j}$	$S(\overline{r_i r_j})$	$\{r_{i+1}, \ldots, r_{j-1}\}$	$S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$
$\overline{r_1 r_5 r_4}$	r_1	r_4	$S_k^{r_1} = \{s_1, s_2\}$	$S_k^{r_4} = \{s_2, s_3\}$	$S(\overline{r_1 r_5 r_4}) = \varnothing$	$\{r_5\}$	$\{s_1, s_2, s_3\}$
$\overline{r_2 r_3}$	r_2	r_3	$S_k^{r_2} = \{s_3, s_4\}$	$S_k^{r_3} = \{s_3, s_4\}$	$S(\overline{r_2 r_3}) = \{s_4\}$	$\varnothing$	Not applicable
r_6	r_6	r_6	$S_k^{r_6} = \{s_2, s_3\}$	$S_k^{r_6} = \{s_2, s_3\}$	$S(r_6) = \varnothing$	$\varnothing$	Not applicable



(a) $\text{dist}(r_5, s_1) = 3$ for $s_1 \in S_k^{r_1} - (S_k^{r_4} \cup S(\overline{r_1 r_5 r_4}))$

(b) $\text{dist}(r_5, s_2) = 6$ for $s_2 \in S_k^{r_1} \cap S_k^{r_4} - S(\overline{r_1 r_5 r_4})$

(c) $\text{dist}(r_5, s_3) = 15$ for $s_3 \in S_k^{r_4} - (S_k^{r_1} \cup S(\overline{r_1 r_5 r_4}))$

Figure 6: Computation of the distance from r_5 to $s \in \{s_1, s_2, s_3\}$.

considered depending on the number of outer objects in $\overline{r_i r_j}$, that is, $|\overline{r_i r_j}| = 1$, $|\overline{r_i r_j}| = 2$, and $|\overline{r_i r_j}| \geq 3$, where $|\overline{r_i r_j}|$ returns the number of outer objects in $\overline{r_i r_j}$. $|\overline{r_i r_j}| = 1$ denotes that $\overline{r_i r_j}$ contains only one outer object r_i, $|\overline{r_i r_j}| = 2$ denotes that $\overline{r_i r_j}$ contains only two outer objects r_i and r_j, and $|\overline{r_i r_j}| \geq 3$ denotes that $\overline{r_i r_j}$ contains more than three outer objects $r_i, r_{i+1}, \ldots, r_j$. If $|\overline{r_i r_j}| = 1$, then the kNN query from r_i is evaluated. In this study, INE [16] is employed to evaluate the kNN query. Naturally, INE can be replaced by other kNN algorithms [1, 2, 5, 13, 17–19]. A partial join result $\Psi(r_i)$ for r_i is generated from $S_k^{r_i}$ and returned (lines 1–4). Similarly, if $|\overline{r_i r_j}| = 2$, then two kNN queries from r_i and r_j are evaluated. Partial join results $\Psi(r_i)$ and $\Psi(r_j)$ for r_i and r_j are generated from $S_k^{r_i}$ and $S_k^{r_j}$, respectively; and their union is returned (lines 5–10). Finally, if $|\overline{r_i r_j}| \geq 3$, then two kNN queries from r_i and r_j are evaluated and a search for the inner objects located in $\overline{r_i r_j}$ is performed. The partial join results $\Psi(r_i)$ and $\Psi(r_j)$ for r_i and r_j are generated from $S_k^{r_i}$ and $S_k^{r_j}$, respectively. For each outer object $r \in \{r_{i+1}, \ldots, r_{j-1}\}$, the set of kNNs of r is retrieved from $S_k^{r_i}$, $S_k^{r_j}$, and $S(\overline{r_i r_j})$ (lines 17–19), as explained in Algorithm 4. Finally, the union of partial join results $\Psi(r_i), \Psi(r_{i+1}), \ldots, \Psi(r_j)$ is returned.

Algorithm 4 describes the kNN search of an outer object $r \in \{r_{i+1}, \ldots, r_{j-1}\}$. First, the set of kNNs of outer object r, S_k^r, is initialized to the empty set. The distance from r to a candidate inner object s is computed according to the condition of s (lines 3–10), as explained in Table 2. After computing $\text{dist}(r, s)$, we can determine whether s is added to the candidate set S_k^r. If $|S_k^r| < k$, then s is simply added to S_k^r (lines 12-13). If $|S_k^r| = k$ and $\text{dist}(r, s) < \text{dist}(r, s_{k\text{th}})$, then s is added to S_k^r and $s_{k\text{th}}$ is removed from S_k^r accordingly, where $s_{k\text{th}}$ denotes the kth closest inner object to r, that is, $S_k^r \leftarrow S_k^r \cup \{s\} - \{s_{k\text{th}}\}$ (lines 14-15). The set S_k^r of the kNNs of r is returned after all of the candidate inner objects in $S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j})$ have been considered (line 16).

4.4. Evaluation of an Example kNN Join Using the Basic GNL Method. We now discuss the evaluation of the kNN join example shown in Figure 3. We recall that $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$, $S = \{s_1, s_2, s_3, s_4\}$, and $k = 2$ are given. Table 3 summarizes the computation of the kNN join $R \ltimes S$.

As explained in Algorithm 2, the basic GNL method first groups the outer objects in a vertex sequence into an outer segment. Therefore, $\overline{R} = \{\overline{r_1 r_5 r_4}, \overline{r_2 r_3}, r_6\}$ is generated from $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$, as shown in Figure 4. In this example, we process $\overline{r_1 r_5 r_4}$, $\overline{r_2 r_3}$, and r_6 in this order. As explained in Algorithm 3, we process the outer segment differently according to the number of outer objects in the outer segment. Outer segment $\overline{r_1 r_5 r_4}$ contains three outer objects r_1, r_5, and r_4, so two kNN queries are issued from r_1 and r_4, and the inner objects located in $\overline{r_1 r_5 r_4}$ are retrieved. We have $S_k^{r_1} = \{s_1, s_2\}$, $S_k^{r_4} = \{s_2, s_3\}$, and $S(\overline{r_1 r_5 r_4}) = \varnothing$, as shown in Table 3. Thus, $S_k^{r_1} = \{s_1, s_2\}$ and $S_k^{r_4} = \{s_2, s_3\}$ simply generate the partial join results $\Psi(r_1) = \{\langle r_1, s_1\rangle, \langle r_1, s_2\rangle\}$ and $\Psi(r_4) = \{\langle r_4, s_2\rangle, \langle r_4, s_3\rangle\}$, respectively.

According to Lemma 1, we can retrieve the kNNs of outer object r_5 from a set of candidate inner objects $S_k^{r_1} \cup S_k^{r_4} \cup S(\overline{r_1 r_5 r_4}) = \{s_1, s_2, s_3\}$ without issuing a kNN query for r_5. Thus, we need to compute the distance

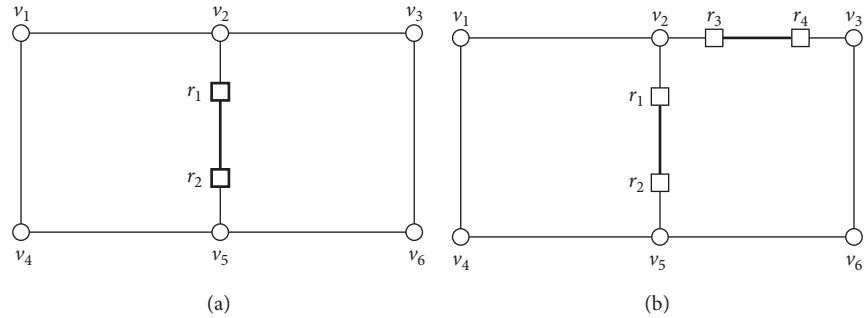


Figure 7: Example of shared kNN query processing of the GNL method: (a) evaluating two kNN queries at r_1 and r_2 and (b) evaluating two kNN queries at v_2 and v_5.

Input: k: number of inner objects retrieved for an outer object, R: set of outer objects, S: set of inner objects
Output: $R \ltimes S$: set of ordered pairs of outer object $r \in R$ and inner object $s \in S_k^r$

(1) $\Psi \leftarrow \varnothing$ // kNN join result set Ψ is initialized to the empty set
(2) // step 1: outer objects $r_i, r_{i+1}, \ldots, r_j$ in a vertex sequence are grouped into an outer segment $\overline{r_i r_j}$
(3) $\overline{R} \leftarrow$ group_outer_objects(R, V, E) // group_outer_objects is explained in Algorithm 1
(4) // step 2: kNN join is evaluated for each outer segment $\overline{r_i r_j} \in \overline{R}$
(5) for each outer segment $\overline{r_i r_j} \in \overline{R}$ **do**
(6) **if** $|\overline{r_i r_j}| = 1$ **then** // see Figures 8(a) to 8(d)
(7) $S_k^{r_i} \leftarrow k$NN_query(k, r_i) // kNN query from r_i is performed, and its result is saved to $S_k^{r_i}$
(8) $\Psi(r_i) \leftarrow \{\langle r_i, s\rangle | s \in S_k^{r_i}\}$ // a partial join result $\Psi(r_i)$ for r_i is generated from $S_k^{r_i}$
(9) $\Psi \leftarrow \Psi \cup \Psi(r_i)$
(10) **else if** $|\overline{r_i r_j}| \geq 2$ **then** // it is assumed that $\overline{r_i r_j}$ belongs to $\overline{v_l v_m}$ and that r_i (r_j) is close to v_l (v_m)
(11) **if** $\Omega(v_l) \geq 2$ **and** $\Omega(v_m) \geq 2$ **then** // see Figure 9(a)
(12) $S_k^{v_l} \leftarrow k$NN_query(k, v_l) // result of kNN query at v_l is reused for other outer segment(s)
(13) $S_k^{v_m} \leftarrow k$NN_query(k, v_m) // result of kNN query at v_m is reused for other outer segment(s)
(14) $\Psi(\overline{r_i r_j}) \leftarrow k$NN_search_of_outer_objects$(k, \overline{r_i r_j}, S_k^{v_l} \cup S_k^{v_m} \cup S(\overline{v_l v_m}))$ // refer to Algorithm 6
(15) $\Psi \leftarrow \Psi \cup \Psi(\overline{r_i r_j})$
(16) **else if** $\Omega(v_l) \geq 2$ **and** $\Omega(v_m) = 1$ **then** // see Figure 9(b)
(17) $S_k^{v_l} \leftarrow k$NN_query(k, v_l) // result of kNN query at v_l is reused for other outer segment(s)
(18) $S_k^{r_j} \leftarrow k$NN_query(k, r_j) // result of kNN query at r_j is not reused
(19) $\Psi(\overline{r_i r_j}) \leftarrow k$NN_search_of_outer_objects$(k, \overline{r_i r_j}, S_k^{v_l} \cup S_k^{r_j} \cup S(\overline{v_l r_j}))$ // refer to Algorithm 6
(20) $\Psi \leftarrow \Psi \cup \Psi(\overline{r_i r_j})$
(21) **else if** $\Omega(v_l) = 1$ **and** $\Omega(v_m) \geq 2$ **then** // see Figure 9(c)
(22) $S_k^{r_i} \leftarrow k$NN_query(k, r_i) // result of kNN query at r_i is not reused
(23) $S_k^{v_m} \leftarrow k$NN_query(k, v_m) // result of kNN query at v_m is reused for other outer segment(s)
(24) $\Psi(\overline{r_i r_j}) \leftarrow k$NN_search_of_outer_objects$(k, \overline{r_i r_j}, S_k^{r_i} \cup S_k^{v_m} \cup S(\overline{r_i v_m}))$ // refer to Algorithm 6
(25) $\Psi \leftarrow \Psi \cup \Psi(\overline{r_i r_j})$
(26) **else if** $\Omega(v_l) = 1$ **and** $\Omega(v_m) = 1$ **then** // see Figure 9(d)
(27) $S_k^{r_i} \leftarrow k$NN_query(k, r_i) // result of kNN query at r_i is not reused
(28) $S_k^{r_j} \leftarrow k$NN_query(k, r_j) // result of kNN query at r_j is not reused
(29) $\Psi(\overline{r_i r_j}) \leftarrow k$NN_search_of_outer_objects$(k, \overline{r_i r_j}, S_k^{r_i} \cup S_k^{r_j} \cup S(\overline{r_i r_j}))$ // refer to Algorithm 6
(30) $\Psi \leftarrow \Psi \cup \Psi(\overline{r_i r_j})$
(31) **return** Ψ // $\Psi = \{\langle r, s\rangle | \forall r \in R, \forall s \in S_k^r\}$ is returned after all outer segments have been processed

Algorithm 5: GNL (k, R, S).

from r_5 to each candidate inner object $s \in \{s_1, s_2, s_3\}$. Clearly, $s_1 \in S_k^{r_1} - (S_k^{r_4} \cup S(\overline{r_1 r_5 r_4}))$ according to Table 3, so the distance from r_5 to s_1 is $\text{dist}(r_5, s_1) = \text{len}(r_5, r_1) + \text{dist}(r_1, s_1) = 3$, as shown in Figure 6(a). Similarly, $s_2 \in S_k^{r_1} \cap S_k^{r_4} - S(\overline{r_1 r_5 r_4})$ according to Table 3, so the distance from r_5 to s_2 is $\text{dist}(r_5, s_2) = \min\{\text{len}(r_5, r_1) + \text{dist}(r_1, s_2),$

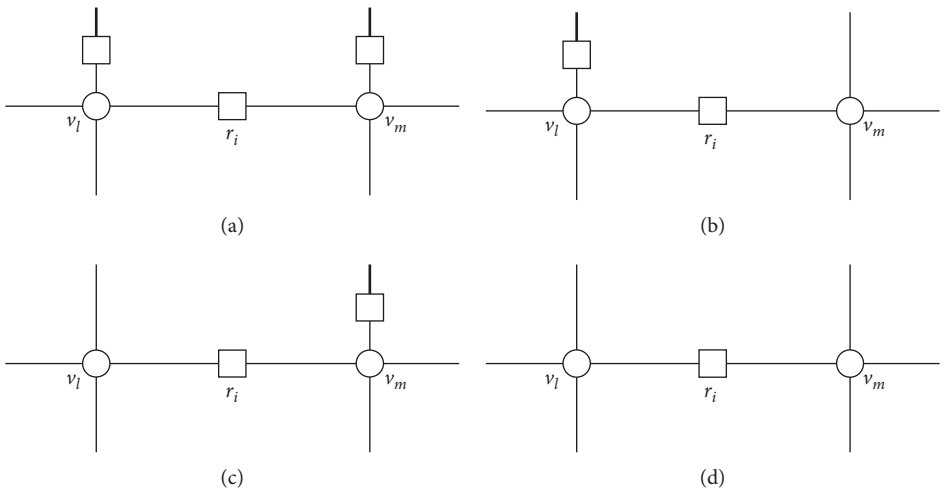


FIGURE 8: $|\overline{r_i r_j}| = 1$. (a) $\Omega(v_l) \geq 2$ and $\Omega(v_m) \geq 2$, (b) $\Omega(v_l) \geq 2$ and $\Omega(v_m) = 1$, (c) $\Omega(v_l) = 1$ and $\Omega(v_m) \geq 2$, and (d) $\Omega(v_l) = 1$ and $\Omega(v_m) = 1$.

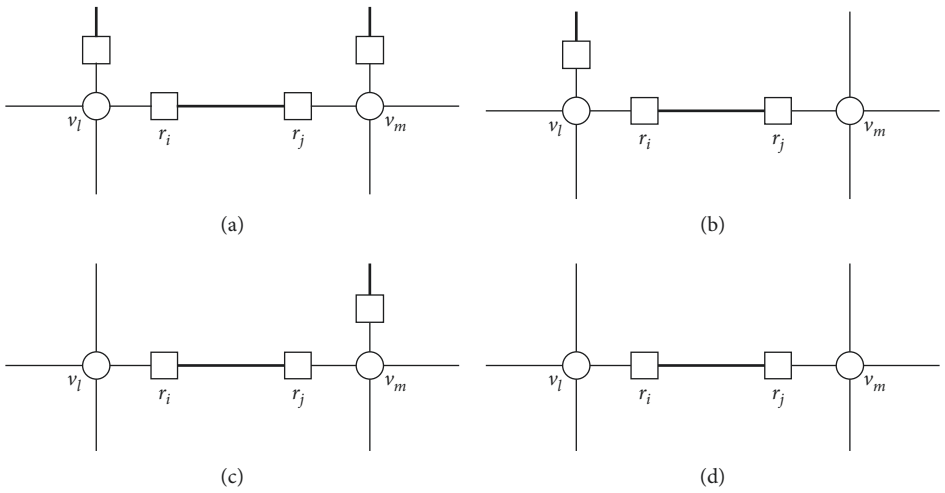


FIGURE 9: $|\overline{r_i r_j}| \geq 2$. (a) $\Omega(v_l) \geq 2$ and $\Omega(v_m) \geq 2$, (b) $\Omega(v_l) \geq 2$ and $\Omega(v_m) = 1$, (c) $\Omega(v_l) = 1$ and $\Omega(v_m) \geq 2$, and (d) $\Omega(v_l) = 1$ and $\Omega(v_m) = 1$.

$\text{len}(r_5, r_4) + \text{dist}(r_4, s_2)\} = \min\{6, 16\} = 6$, as shown in Figure 6(b). Finally, $s_3 \in S_k^{r_4} - (S_k^{r_1} \cup S(\overline{r_1 r_5 r_4}))$ according to Table 3, so the distance from r_5 to s_3 is $\text{dist}(r_5, s_3) = \text{len}(r_5, r_4) + \text{dist}(r_4, s_3) = 15$, as shown in Figure 6(c). It should be noted that the shortest path from r_5 to s_3 is $r_5 \to r_1 \to s_3$ rather than $r_5 \to r_4 \to s_3$, and thus the shortest distance from r_5 to s_3 is $\text{dist}(r_5, s_3) = \text{len}(r_5, r_1) + \text{dist}(r_1, s_3) = 9$. However, this does not affect the correctness of the set of the kNNs of r_5 because s_3 does not belong to $S_k^{r_1} = \{s_1, s_2\}$ so the path $r_5 \to r_1 \to s_3$ does not have to be considered. Consequently, we have $S_k^{r_5} = \{s_1, s_2\}$ from $\text{dist}(r_5, s_1) = 3$, $\text{dist}(r_5, s_2) = 6$, and $\text{dist}(r_5, s_3) = 15$, which generates the partial join result $\Psi(r_5) = \{\langle r_5, s_1\rangle, \langle r_5, s_2\rangle\}$.

Next, we compute the partial join results for the outer objects in $\overline{r_2 r_3}$. Outer segment $\overline{r_2 r_3}$ only contains two outer objects r_2 and r_3, so two kNN queries are issued from r_2 and r_3. Therefore, as shown in Table 3, we obtain $S_k^{r_2} = \{s_3, s_4\}$ and $S_k^{r_3} = \{s_3, s_4\}$, which directly generate the partial join results $\Psi(r_2) = \{\langle r_2, s_3\rangle, \langle r_2, s_4\rangle\}$ and $\Psi(r_3) = \{\langle r_3, s_3\rangle, \langle r_3, s_4\rangle\}$, respectively. Finally, we compute the partial join result for outer object r_6, where a single kNN query is simply issued from r_6. Therefore, as shown in Table 3, we obtain $S_k^{r_6} = \{s_2, s_3\}$, which generates the partial join result $\Psi(r_6) = \{\langle r_6, s_2\rangle, \langle r_6, s_3\rangle\}$. Finally, we obtain the kNN join result that is the union of the partial join results for outer objects r_1 to r_6, as follows:

$$\begin{aligned}\Psi &= \Psi(\overline{r_1 r_5 r_4}) \cup \Psi(\overline{r_2 r_3}) \cup \Psi(r_6) \\ &= \{\langle r_1, s_1\rangle, \langle r_1, s_2\rangle, \langle r_2, s_3\rangle, \langle r_2, s_4\rangle, \langle r_3, s_3\rangle, \langle r_3, s_4\rangle, \\ &\quad \langle r_4, s_2\rangle, \langle r_4, s_3\rangle, \langle r_5, s_1\rangle, \langle r_5, s_2\rangle, \langle r_6, s_2\rangle, \langle r_6, s_3\rangle\},\end{aligned} \tag{3}$$

where $\Psi(\overline{r_1 r_5 r_4}) = \Psi(r_1) \cup \Psi(r_5) \cup \Psi(r_4)$ and $\Psi(\overline{r_2 r_3}) = \Psi(r_2) \cup \Psi(r_3)$.

Input: k: number of inner objects retrieved for an outer object, $\overline{r_i r_j}$: outer segment, S_k^{cnd}: set of candidate inner objects
Output: $\Psi(\overline{r_i r_j})$: a partial join result for outer objects in $\overline{r_i r_j}$
(1) $\Psi(\overline{r_i r_j}) \leftarrow \varnothing$ // $\Psi(\overline{r_i r_j})$ is initialized to the empty set
(2) **for** each outer object $r \in \overline{r_i r_j}$ **do**
(3) $S_k^r \leftarrow k\text{NN\_search}(k, r, S_k^{\text{cnd}})$ // kNN_search in Algorithm 4 is performed
(4) $\Psi(r) \leftarrow \{\langle r, s\rangle | s \in S_k^r\}$ // a partial join result $\Psi(r)$ for r is generated from S_k^r
(5) $\Psi(\overline{r_i r_j}) \leftarrow \Psi(\overline{r_i r_j}) \cup \Psi(r)$
(6) **return** $\Psi(\overline{r_i r_j})$ // $\Psi(\overline{r_i r_j}) = \{\langle r, s\rangle | \forall r \in \overline{r_i r_j}, \forall s \in S_k^r\}$ is returned after all outer objects in $\overline{r_i r_j}$ have been processed

Algorithm 6: kNN_search_of_outer_objects$(k, \overline{r_i r_j}, S_k^{\text{cnd}})$.

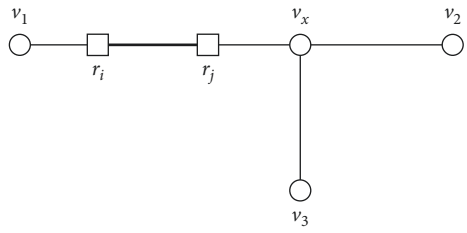


Figure 10: $S_k^r \subseteq S_k^{r_j} \cup S(\overline{v_1 r_j})$ for each outer object $r \in \overline{r_i r_j}$.

Table 4: Investigation of N_{GNL} and N_{OPT}.

Condition	N_{GNL} (query locations)	N_{OPT} (query locations)	Does N_{GNL} equal N_{OPT}?
Figure 8(a)	3 (v_l, v_m, r_i)	2 (v_l,v_m)	No
Figure 8(b)	2 (v_l, r_i)	2 (v_l, r_i)	Yes
Figure 8(c)	2 (r_i, v_m)	2 (r_i, v_m)	Yes
Figure 8(d)	1 (r_i)	1 (r_i)	Yes
Figure 9(a)	2 (v_l, v_m)	2 (v_l, v_m)	Yes
Figure 9(b)	2 (v_l, r_j)	2 (v_l, r_j)	Yes
Figure 9(c)	2 (r_i, v_m)	2 (r_i, v_m)	Yes
Figure 9(d)	2 (r_i, r_j)	2 (r_i, r_j)	Yes

5. GNL Method for kNN Joins in Road Networks

5.1. Avoiding Redundant kNN Queries. We present the GNL method to evaluate a smaller number of kNN queries than the basic GNL method. To this end, we investigate the number of outer segments in vertex sequences that are adjacent to an intersection vertex. Let $\Omega(v_x)$ be the number of outer segments in vertex sequences that are adjacent to an intersection vertex v_x. If $\Omega(v_x) = 0$, no kNN query is issued at v_x. If $\Omega(v_x) = 1$, a kNN is evaluated at r_i, where it is assumed that an outer segment $\overline{r_i r_j}$ is in a vertex sequence adjacent to v_x and that r_i is closer to v_x than r_j. If $\Omega(v_x) \geq 2$, a kNN query is evaluated at v_x instead of at r_i. Figure 7 illustrates a simple example of the shared query processing to reduce the number of kNN queries used to evaluate the kNN join. Observe that there are two intersection vertices v_2 and v_5, both of which are adjacent to three vertex sequences $\overline{v_2 v_1 v_4 v_5}$, $\overline{v_2 v_5}$, and $\overline{v_2 v_3 v_6 v_5}$. In Figure 7(a), an outer segment $\overline{r_1 r_2}$ is in a vertex sequence $\overline{v_2 v_5}$ and we have $\Omega(v_2) = 1$ and $\Omega(v_5) = 1$. Therefore, two kNN queries are evaluated at r_1 and r_2. However, in Figure 7(b), outer segments $\overline{r_1 r_2}$ and $\overline{r_3 r_4}$ are in vertex sequences $\overline{v_2 v_5}$ and $\overline{v_2 v_3 v_6 v_5}$, respectively, both of which are adjacent to v_2 and v_5, and we have $\Omega(v_2) = 2$ and $\Omega(v_5) = 2$. Therefore, the GNL method evaluates only two kNN queries at v_2 and v_5, instead of evaluating four kNN queries at r_1, r_2, r_3, and r_4.

Algorithm 5 describes the GNL algorithm that employs an optimized number of kNN queries to evaluate kNN joins by exploiting grouping and shared execution. Similar to the basic GNL algorithm, this algorithm comprises two steps. In the first step, the outer objects $r_i, r_{i+1}, \ldots, r_j$ in a vertex sequence are grouped into an outer segment $\overline{r_i r_j}$ (line 3). In the second step, the GNL algorithm considers $|\overline{r_i r_j}| = 1$ and $|\overline{r_i r_j}| \geq 2$ as shown in Figures 8 and 9, respectively, to use the shared execution for avoiding redundant kNN queries.

Figure 8 shows that $\overline{r_i r_j}$ contains an outer object only, that is, $|\overline{r_i r_j}| = 1$. For the case of $|\overline{r_i r_j}| = 1$, a kNN query is evaluated at r_i, whose result is not reused (lines 6–9).

Figure 9 shows that $\overline{r_i r_j}$ contains more than two outer objects, that is, $|\overline{r_i r_j}| \geq 2$, meaning that two kNN queries are required to retrieve the k inner objects closest to each outer object $r \in \overline{r_i r_j}$. As shown in Figure 9(a), for the case of $\Omega(v_l) \geq 2$ and $\Omega(v_m) \geq 2$, two kNN queries at v_l and v_m are evaluated, whose results are reused for the other outer segments adjacent to v_l or v_m (lines 11–15). As shown in Figure 9(b), for the case of $\Omega(v_l) \geq 2$ and $\Omega(v_m) = 1$, two kNN queries at v_l and r_j are evaluated, and the query result at v_l is reused for the other outer segment(s) adjacent to v_l (lines 16–20). As shown in Figure 9(c), for the case of $\Omega(v_l) = 1$ and $\Omega(v_m) \geq 2$, two kNN queries at r_i and v_m are evaluated, and the query result at v_m is reused for the other outer segment(s) adjacent to v_m (lines 21–25). Finally, as shown in Figure 9(d), for the case of $\Omega(v_l) = 1$ and $\Omega(v_m) = 1$, two kNN queries at r_i and r_j are evaluated, whose results are not reused (lines 26–30).

Algorithm 6 describes how to find the k inner objects closest to each outer object $r \in \overline{r_i r_j}$, where the kNN search of the outer object is performed using Algorithm 4.

To avoid evaluating redundant kNN queries, we present a simple heuristic so that the redundant kNN queries are not evaluated at locations close to terminal vertices. For example, as shown in Figure 10, the graph has one intersection vertex v_x and three terminal vertices v_1, v_2, and v_3. In this example, the kNN query at r_i is redundant because it holds that $S_k^r \subseteq S_k^{r_j} \cup S(\overline{v_1 r_j})$ for each outer object $r \in \overline{r_i r_j}$.

Table 5: Computation of the example kNN join $R \ltimes S$ using the GNL method.

$\overline{r_i r_j}$	v_l	v_m	$S_k^{v_l}$	$S_k^{v_m}$	$S(\overline{v_l \cdots v_m})$	$\{r_i, r_{i+1}, \ldots, r_j\}$	$S_k^{v_l} \cup S_k^{v_m} \cup S(\overline{v_l \ldots v_m})$
$\overline{r_1 r_5 r_4}$	v_1	v_3	$S_k^{v_1} = \{s_2, s_3\}$	$S_k^{v_3} = \{s_2, s_3\}$	$S(\overline{v_1 v_2 v_3}) = \{s_1, s_2\}$	$\{r_1, r_5, r_4\}$	$S_k^{v_1} \cup S_k^{v_3} \cup S(\overline{v_1 v_2 v_3}) = \{s_1, s_2, s_3\}$
$\overline{r_2 r_3}$	v_1	v_3	$S_k^{v_1} = \{s_2, s_3\}$	$S_k^{v_3} = \{s_2, s_3\}$	$S(\overline{v_1 v_4 v_3}) = \{s_3, s_4\}$	$\{r_2, r_3\}$	$S_k^{v_1} \cup S_k^{v_3} \cup S(\overline{v_1 v_4 v_3}) = \{s_2, s_3, s_4\}$

Table 6: Summary of computing the example kNN join $R \ltimes S$.

$\overline{r_i r_j}$	$r \in \overline{r_i r_j}$	s	Condition of s	$\text{dist}(r, s)$	S_k^r	$\Psi(r)$
$\overline{r_1 r_5 r_4}$	r_1	s_1	$s_1 \in S(\overline{v_1 v_2 v_3}) - (S_k^{v_1} \cup S_k^{v_3})$	$\text{dist}(r_1, s_1) = 2$	$S_k^{r_1} = \{s_1, s_2\}$	$\Psi(r_1) = \{\langle r_1, s_1\rangle, \langle r_1, s_2\rangle\}$
		s_2	$s_2 \in S_k^{v_1} \cap S_k^{v_3} \cap S(\overline{v_1 v_2 v_3})$	$\text{dist}(r_1, s_2) = 5$		
		s_3	$s_3 \in S_k^{v_1} \cap S_k^{v_3} - S(\overline{v_1 v_2 v_3})$	$\text{dist}(r_1, s_3) = 8$		
	r_5	s_1	$s_1 \in S(\overline{v_1 v_2 v_3}) - (S_k^{v_1} \cup S_k^{v_3})$	$\text{dist}(r_5, s_1) = 3$	$S_k^{r_5} = \{s_1, s_2\}$	$\Psi(r_5) = \{\langle r_5, s_1\rangle, \langle r_5, s_2\rangle\}$
		s_2	$s_2 \in S_k^{v_1} \cap S_k^{v_3} \cap S(\overline{v_1 v_2 v_3})$	$\text{dist}(r_5, s_2) = 6$		
		s_3	$s_3 \in S_k^{v_1} \cap S_k^{v_3} - S(\overline{v_1 v_2 v_3})$	$\text{dist}(r_5, s_3) = 9$		
	r_4	s_1	$s_1 \in S(\overline{v_1 v_2 v_3}) - (S_k^{v_1} \cup S_k^{v_3})$	$\text{dist}(r_4, s_1) = 10$	$S_k^{r_4} = \{s_2, s_3\}$	$\Psi(r_4) = \{\langle r_4, s_2\rangle, \langle r_4, s_3\rangle\}$
		s_2	$s_2 \in S_k^{v_1} \cap S_k^{v_3} \cap S(\overline{v_1 v_2 v_3})$	$\text{dist}(r_4, s_2) = 9$		
		s_3	$s_3 \in S_k^{v_1} \cap S_k^{v_3} - S(\overline{v_1 v_2 v_3})$	$\text{dist}(r_4, s_3) = 8$		
$\overline{r_2 r_3}$	r_2	s_2	$s_2 \in S_k^{v_1} \cap S_k^{v_3} - S(\overline{v_1 v_2 v_3})$	$\text{dist}(r_2, s_2) = 9$	$S_k^{r_2} = \{s_3, s_4\}$	$\Psi(r_2) = \{\langle r_2, s_3\rangle, \langle r_2, s_4\rangle\}$
		s_3	$s_3 \in S_k^{v_1} \cap S_k^{v_3} \cap S(\overline{v_1 v_2 v_3})$	$\text{dist}(r_2, s_3) = 6$		
		s_4	$s_4 \in S(\overline{v_1 v_2 v_3}) - (S_k^{v_1} \cap S_k^{v_3})$	$\text{dist}(r_2, s_4) = 3$		
	r_3	s_2	$s_2 \in S_k^{v_1} \cap S_k^{v_3} - S(\overline{v_1 v_2 v_3})$	$\text{dist}(r_3, s_2) = 13$	$S_k^{r_3} = \{s_3, s_4\}$	$\Psi(r_3) = \{\langle r_3, s_3\rangle, \langle r_3, s_4\rangle\}$
		s_3	$s_3 \in S_k^{v_1} \cap S_k^{v_3} \cap S(\overline{v_1 v_2 v_3})$	$\text{dist}(r_3, s_3) = 12$		
		s_4	$s_4 \in S(\overline{v_1 v_4 v_3}) - (S_k^{v_1} \cap S_k^{v_3})$	$\text{dist}(r_3, s_4) = 3$		
r_6	r_6	s_2	$s_2 \in S_k^{r_6}$	$\text{dist}(r_6, s_2) = 5$	$S_k^{r_6} = \{s_2, s_3\}$	$\Psi(r_6) = \{\langle r_6, s_2\rangle, \langle r_6, s_3\rangle\}$
		s_3	$s_3 \in S_k^{r_6}$	$\text{dist}(r_6, s_3) = 4$		

Table 7: Experimental parameter settings.

Parameter	Range
Number of requested NNs (k)	1, 10, **30**, 50, 100 for NA and SF 1, 5, **10**, 30, 50 for SJ
Numbers of outer objects ($\lvert R\rvert$) and inner objects ($\lvert S\rvert$)	1, 3, **5**, 7, 10 ($\times 10^4$) for NA and SF 1, 3, **5**, 7, 10 ($\times 10^3$) for SJ
Distributions of outer objects and inner objects	**(C)entroid**, (U)niform
Real-life roadmaps	NA, SF, SJ

The GNL method employs an optimized number of kNN queries to evaluate kNN joins while exploiting grouping and shared execution. For this, we consider the eight cases in Figures 8 and 9. For each case, Table 4 investigates the number N_{GNL} of kNN queries evaluated by the GNL method and the minimum number N_{OPT} of kNN queries evaluated by the virtual optimal method. The values in parentheses indicate locations of kNN queries evaluated by the GNL and virtual optimal methods. In Figure 8(a), the GNL algorithm evaluates three kNN queries at v_l, v_m, and r_i. However, the kNN query at r_i is redundant because the query results at v_l and v_m can be reused to find the k inner objects closest to r_i. This shows that the GNL algorithm does not always use the minimum number of kNN queries to evaluate the kNN join, as previously mentioned. However, for Figures 8(b) to 8(d) and 9(a) to 9(d), the GNL method evaluates the same number of kNN queries as the virtual optimal method.

5.2. Evaluation of an Example kNN Join Using the GNL Method. Returning to the example kNN join in Figure 3, we reevaluate the kNN join to show that the GNL method evaluates a smaller number of kNN queries than the basic GNL method. We recall that the basic GNL method evaluates five kNN queries at r_1, r_2, r_3, r_4, and r_6 to evaluate the kNN join, as described in Section 4.4. However, the GNL method evaluates three kNN queries at intersection vertices v_1 and v_3, and an outer object r_6 because $\Omega(v_1) = \Omega(v_3) = 3$ and $|\overline{r_6}| = 1$ are given. Specifically, the kNN query at v_1 replaces the two kNN queries that the basic GNL method issues at r_3 and r_4. Similarly, the kNN query at v_3 replaces the two kNN queries that the basic GNL method issues at r_1 and r_2.

We process $\overline{r_1 r_5 r_4}$, $\overline{r_2 r_3}$, and r_6 in this order. We retrieve kNNs of each outer object in the segments $\overline{r_1 r_5 r_4}$, $\overline{r_2 r_3}$, and r_6 from a set of candidate inner objects. First, we retrieve two NNs of an outer object r_1 in $\overline{r_1 r_5 r_4}$. For this, we should

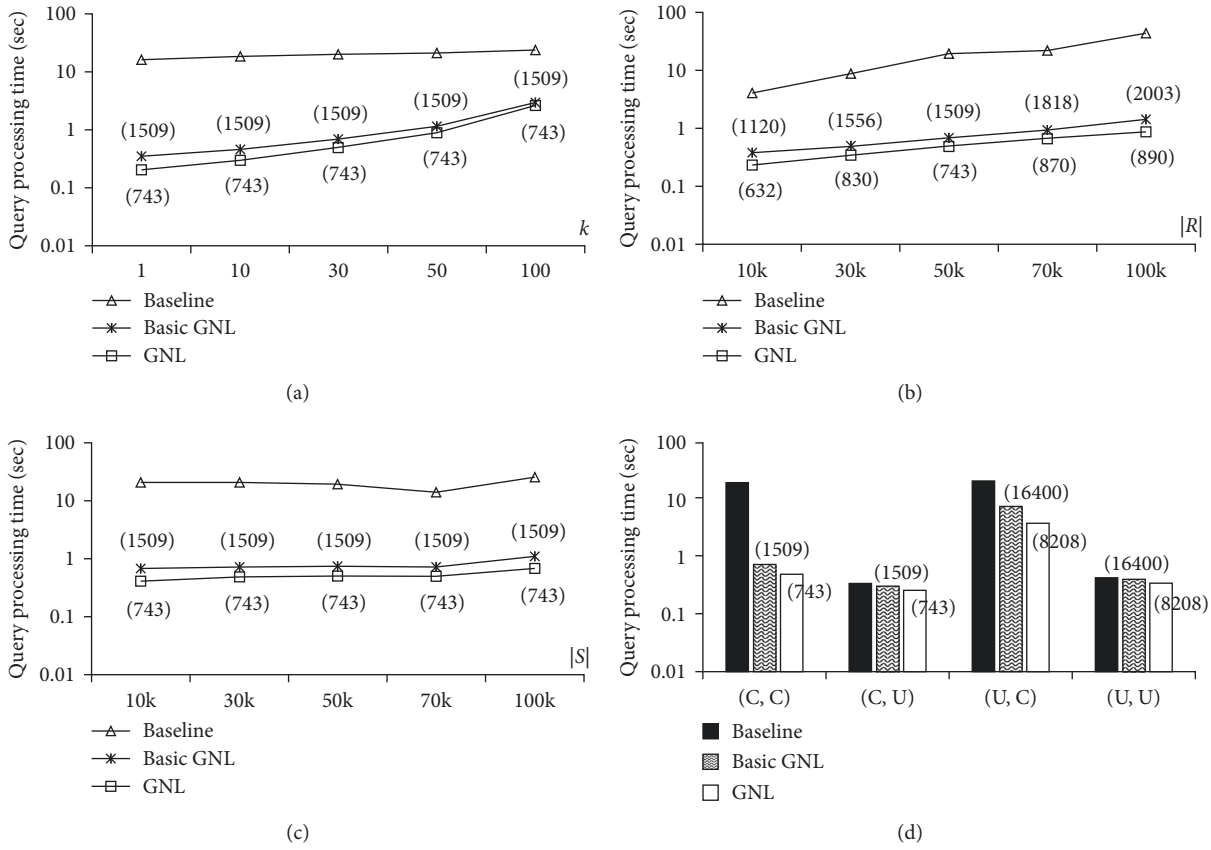


Figure 11: Comparison of the baseline, basic GNL, and GNL methods for NA: (a) varying k, (b) varying $|R|$, (c) varying $|S|$, and (d) varying the distribution of objects.

determine $\text{dist}(r_1, s_1)$, $\text{dist}(r_1, s_2)$, and $\text{dist}(r_1, s_3)$ to retrieve two inner objects closest to r_1. According to Table 5, $s_1 \in S(\overline{v_1 v_2 v_3}) - (S_k^{v_1} \cup S_k^{v_3})$, and the distance from r_1 to s_1 is $\text{dist}(r_1, s_1) = \text{len}(r_1, s_1) = 2$. Similarly, according to Table 5, $s_2 \in S_k^{v_1} \cap S_k^{v_3} \cap S(\overline{v_1 v_2 v_3})$, and the distance from r_1 to s_2 is $\text{dist}(r_1, s_2) = \min\{\text{len}(r_1, v_1) + \text{dist}(v_1, s_2), \text{len}(r_1, v_3) + \text{dist}(v_3, s_2), \text{len}(r_1, s_2)\} = \{17, 9, 5\} = 5$. Finally, according to Table 5, $s_3 \in S_k^{v_1} \cap S_k^{v_3} - S(\overline{v_1 v_2 v_3})$, and the distance from r_1 to s_3 is $\text{dist}(r_1, s_3) = \min\{\text{len}(r_1, v_1) + \text{dist}(v_1, s_3), \text{len}(r_1, v_3) + \text{dist}(v_3, s_3)\} = \{16, 8\} = 8$. Consequently, we have $S_k^{r_1} = \{s_1, s_2\}$ from $\text{dist}(r_1, s_1) = 2$, $\text{dist}(r_1, s_2) = 5$, and $\text{dist}(r_1, s_3) = 8$, which generates the partial join result $\Psi(r_1) = \{\langle r_1, s_1\rangle, \langle r_1, s_2\rangle\}$. In the same manner, we can compute the distance from an outer object $r \in \{r_2, r_3, r_4, r_5, r_6\}$ to each of its candidate objects and determine the two inner objects closest to r. Table 6 presents a summary of computing the distances from an outer object $r \in \{r_1, r_2, r_3, r_4, r_5, r_6\}$ to its candidate objects and determining the two inner objects closest to r. Note that the two inner objects s_2 and s_3 closest to r_6 are retrieved directly from the kNN query at r_6. Finally, using the GNL method, we can obtain the kNN join result $\Psi = \{\langle r_1, s_1\rangle, \langle r_1, s_2\rangle, \langle r_2, s_3\rangle, \langle r_2, s_4\rangle, \langle r_3, s_3\rangle, \langle r_3, s_4\rangle, \langle r_4, s_2\rangle, \langle r_4, s_3\rangle, \langle r_5, s_1\rangle, \langle r_5, s_2\rangle, \langle r_6, s_2\rangle, \langle r_6, s_3\rangle\}$.

6. Performance Evaluation

In this section, we present our empirical analysis of the GNL method, where the experimental settings are given in Section 6.1 and the experimental results in Section 6.2.

6.1. Experimental Settings. In the experiments, we use three real-life roadmaps freely available in [27]. The first road map comprising 175,813 vertices and 179,179 edges includes major roads (e.g., highways) in North America (NA). The second road map comprising 174,956 vertices and 223,001 edges includes major roads (e.g., city streets) in San Francisco (SF), California. The third road map comprising 18,263 vertices and 23,874 edges includes major roads (e.g., city streets) in San Joaquin County (SJ), California. The numbers of vertex sequences in NA, SF, and SJ are 12,416, 192,276, and 20,040, respectively. The settings of the experimental parameters are given in Table 7.

The positions of both the outer objects and inner objects follow either a centroid or uniform distribution. The centroid dataset is generated so that it resembles real-world data. First, 10 centroids are selected randomly, and the objects around each centroid follow the Gaussian distribution, where the mean is set to the centroid and the

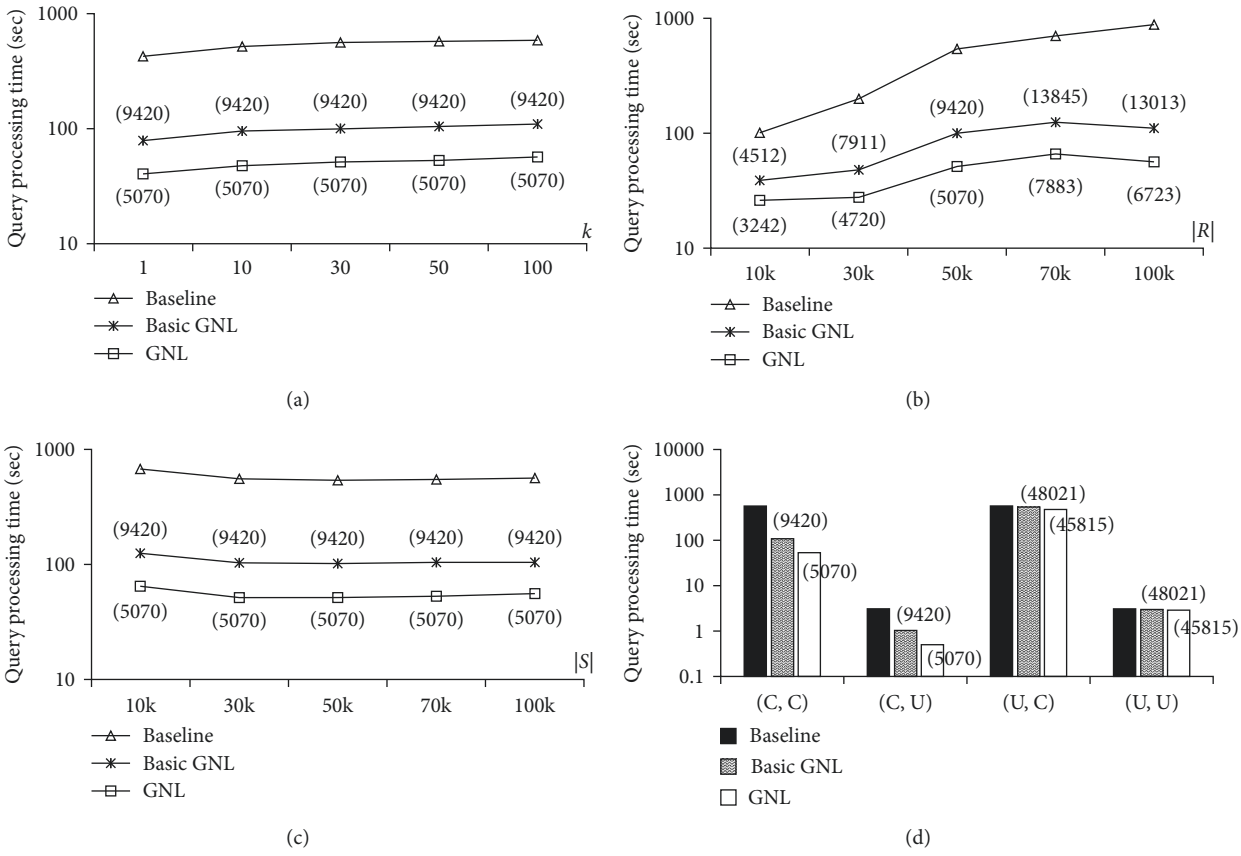


Figure 12: Comparison of the baseline, basic GNL, and GNL methods for SF: (a) varying k, (b) varying $|R|$, (c) varying $|S|$, and (d) varying the distribution of objects.

standard deviation is set to 1% of the side length of the data universe. In each experiment, we vary a single parameter within the range shown in Table 7, while keeping the other parameters at the default values shown in bold. The outer objects and inner objects follow a centroid distribution unless stated otherwise.

We implement and evaluate two versions of the GNL method, that is, the basic GNL and GNL methods. As a benchmark for our proposed method, we use a baseline method that computes the kNNs of every outer object using INE [16]. The three methods are implemented with the maximize speed option using C++ in Visual Studio 2015 and run on a desktop PC running Windows 10 operating system with 32 GB RAM and a quad-core processor (i7-6700 K) at 4 GHz. We consider that the indexing structures of all techniques should be memory resident to ensure responsive query processing, which is assumed recently in many studies [1, 4] and is crucial to online map services and commercial navigation systems. We determine the average values based on 10 experiments.

6.2. Experimental Results. Figure 11 compares the query processing times using the baseline, basic GNL, and GNL methods when evaluating kNN joins in the NA road map, where each chart illustrates the effect of changing one of the parameters in Table 7. The values in parentheses indicate the numbers of kNN queries that are evaluated by the basic GNL and GNL methods to compute the kNN joins. The numbers of kNN queries evaluated by the baseline method to compute the kNN joins are omitted because the numbers of the kNN queries using this method equal the numbers of the outer objects. Figure 11(a) shows the query processing time as a function of the number of requested NNs, that is, k. The processing times using all the methods increase slightly with k. However, the GNL method shows the best performance, and the processing times using the GNL method are up to 80 times shorter than those using the baseline method in all cases. During the shared execution, the basic GNL method only evaluates two kNN queries to retrieve the kNNs of all the outer objects in a segment, which decreases the processing time significantly. The baseline, basic GNL, and GNL methods evaluate a total of 50,000 kNN queries, 1,509 kNN queries, and 743 kNN queries, respectively. This means that the numbers of kNN queries evaluated by the basic GNL and GNL methods represent 3% and 1.5% of the kNN queries evaluated by the baseline method, respectively. This indicates a strong relationship between the query processing times and the numbers of the kNN queries evaluated to compute the kNN joins. Figure 11(b) shows the

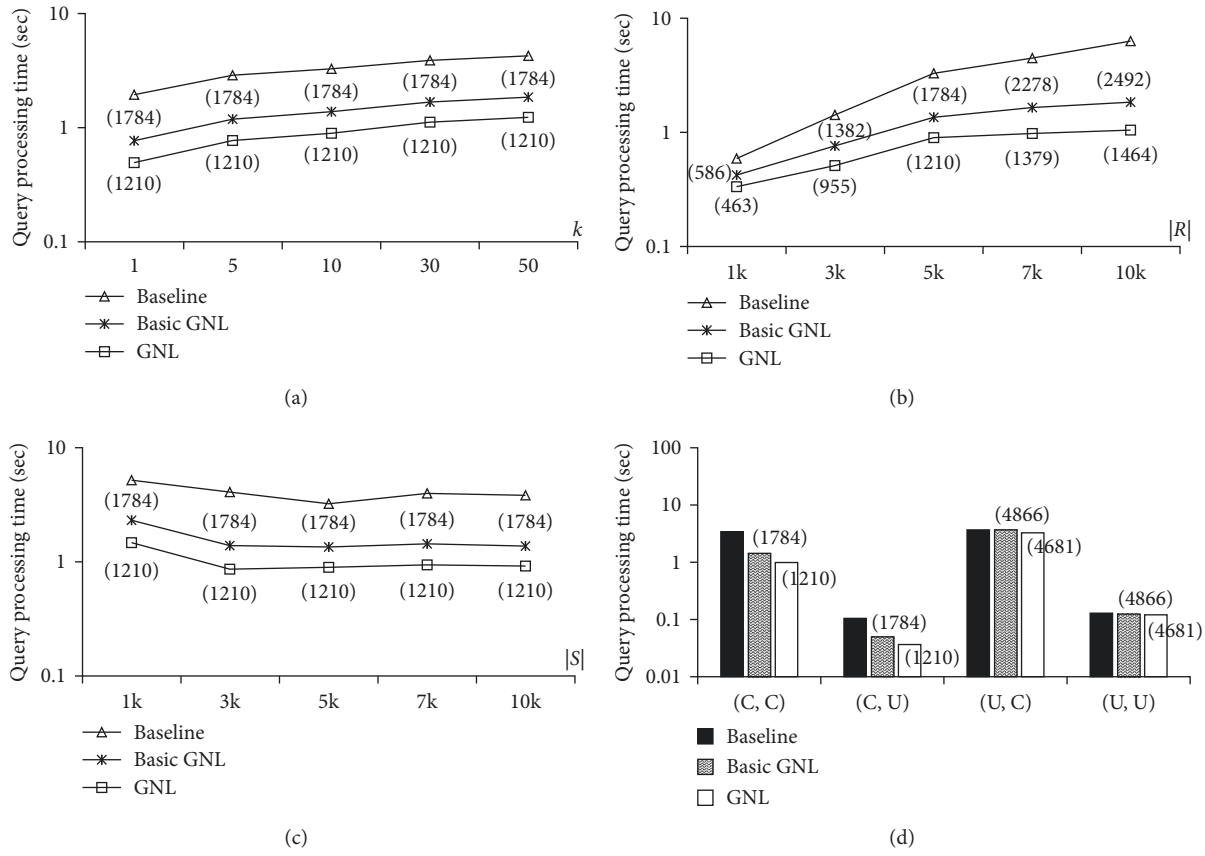


Figure 13: Comparison of the baseline, basic GNL, and GNL methods for SJ: (a) varying k, (b) varying $|R|$, (c) varying $|S|$, and (d) varying the distribution of objects.

query processing time as a function of the number of outer objects, that is, $|R|$. The GNL method performs better than the other methods in all cases. The number of kNN queries in the baseline method increases linearly with $|R|$. However, because of the benefits of shared execution processing, the numbers of kNN queries evaluated by the basic GNL and GNL methods increase slightly with $|R|$. Figure 11(c) shows the query processing time as a function of the number of inner objects, that is, $|S|$. The GNL method performs better than the other methods in all cases. The numbers of kNN queries evaluated by the basic GNL and GNL methods are independent of $|S|$. Figure 11(d) shows the query processing time for various distributions of outer objects and inner objects, where each ordered pair (i.e., $\langle C, C\rangle$, $\langle C, U\rangle$, $\langle U, C\rangle$, and $\langle U, U\rangle$) denotes a combination of the distributions of outer objects and inner objects. The processing time is very long for the baseline method, particularly when the inner objects follow a centroid distribution (i.e., $\langle C, C\rangle$ and $\langle U, C\rangle$). The processing time is also long with the basic GNL and GNL methods for $\langle U, C\rangle$ because the outer objects are widely scattered and most of the outer segments are generated with a few outer objects, which hinders the shared execution processing.

Figure 12 compares the query processing times of the three methods when evaluating kNN joins in the SF road map. Figure 12(a) shows the query processing time as a function of k between 1 and 100. The GNL method shows the best performance in all cases because the GNL method evaluates the smallest number of kNN queries among the three methods. The baseline, basic GNL, and GNL methods evaluate 50,000 kNN queries, 9,420 kNN queries, and 5,070 kNN queries, respectively. Figure 12(b) shows the query processing time as a function of $|R|$ between 10^4 and 10^5. The processing time with the baseline method increases in a linear manner with the value of $|R|$. However, the basic GNL and GNL methods are not sensitive to increases in the value of $|R|$ due to the shared execution processing. The GNL evaluates the smallest number of kNN queries in all cases. Figure 12(c) shows the query processing time as a function of $|S|$ between 10^4 and 10^5. The GNL method outperforms the basic GNL method in all cases because the GNL method evaluates 5,070 kNN queries compared to 9,420 kNN queries for the basic GNL method. Figure 12(d) shows the query processing time for various distributions of outer objects and inner objects. The basic GNL and GNL methods outperform the baseline method for $\langle C, C\rangle$ and $\langle C, U\rangle$. However, the basic GNL and GNL methods show similar performance to the baseline method for $\langle U, C\rangle$ and $\langle U, U\rangle$. This is because the uniform distribution of outer objects obstructs the shared execution processing of the basic GNL and GNL methods.

Figure 13 compares the query processing times of the three methods when evaluating kNN joins in the SJ road map. Figure 13(a) shows the query processing time as a function of k between 1 and 50. The GNL method outperforms the other methods in all cases because it evaluates 1,210 kNN queries, which are fewer than the other two methods. Figure 13(b) shows the query processing time as a function of $|R|$ between 10^3 and 10^4. The processing time using the basic GNL and GNL methods increases less with increasing values of $|R|$ than the baseline method. Figure 13(c) shows the query processing time as a function of $|S|$ between 10^3 and 10^4. The GNL method outperforms the other methods in all cases. Figure 13(d) shows the query processing time for various distributions of outer objects and inner objects. The GNL method outperforms the other methods when the outer objects follow a centroid distribution, that is, $\langle C, C\rangle$ and $\langle C, U\rangle$. However, all methods show a similar performance when the outer objects follow a uniform distribution, that is, $\langle U, C\rangle$ and $\langle U, U\rangle$.

7. Conclusions

In this study, we investigated the kNN join problem in road networks. The kNN join is an operation that combines each object of a dataset with its kNNs in another dataset and is used to facilitate data mining tasks such as clustering, classification, and outlier detection. The kNN join can also provide more meaningful query results than the range join. We proposed the GNL method as an efficient kNN join algorithm, which employs grouping of adjacent outer objects and distance computations based on shared execution to avoid evaluating redundant kNN queries. We evaluated the performance of the GNL method based on several real-life roadmaps in a wide range of problem settings. The empirical results confirmed that the GNL method is efficient and scalable with the number of outer objects and is significantly superior to the baseline method and that the GNL method outperforms the basic GNL method, particularly when the outer objects follow a nonuniform distribution.

Conflicts of Interest

The author declares that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This research was supported by Kyungpook National University Bokhyeon Research Fund, 2016.

References

[1] T. Abeywickrama, M. A. Cheema, and D. Taniar, "k-nearest neighbors on road networks: a journey in experimentation and in-memory implementation," *Proceedings of the VLDB Endowment*, vol. 9, no. 6, pp. 492–503, 2016.

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

[3] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh, "A road network embedding technique for k-nearest neighbor search in moving object databases," *GeoInformatica*, vol. 7, no. 3, pp. 255–273, 2003.

[4] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou, "Shortest path and distance queries on road networks: an experimental evaluation," *Proceedings of the VLDB Endowment*, vol. 5, no. 5, pp. 406–417, 2012.

[5] R. Zhong, G. Li, K.-L. Tan, L. Zhou, and Z. Gong, "G-tree: an efficient and scalable index for spatial search on road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2175–2189, 2015.

[6] C. Böhm and F. Krebs, "The k-nearest neighbour join: turbo charging the KDD Process," *Knowledge and Information Systems*, vol. 6, no. 6, pp. 728–749, 2004.

[7] C. Xia, H. Lu, B. C. Ooi, and J. Hu, "Gorder: an efficient method for kNN join processing," in *Proceedings Very Large Data Bases (VLDB)*, pp. 756–767, Toronto, ON, Canada, 2004.

[8] C. Yu, B. Cui, S. Wang, and J. Su, "Efficient index-based kNN join processing for high-dimensional data," *Information and Software Technology*, vol. 49, no. 4, pp. 332–344, 2007.

[9] W. Lu, Y. Shen, S. Chen, and B. C. Ooi, "Efficient processing of k nearest neighbor joins using MapReduce," *Proceedings of the VLDB Endowment*, vol. 5, no. 10, pp. 1016–1027, 2012.

[10] B. Yao, F. Li, and P. Kumar, "K nearest neighbor queries and kNN-joins in large relational databases (almost) for free," in *Proceedings of the International Conference on Data Engineering (ICDE)*, pp. 4–15, Long Beach, CA, USA, March 2010.

[11] C. Yu, R. Zhang, Y. Huang, and H. Xiong, "High-dimensional kNN joins with incremental updates," *GeoInformatica*, vol. 14, no. 1, pp. 55–82, 2010.

[12] C. Zhang, F. Li, and J. Jestes, "Efficient parallel kNN joins for large data in MapReduce," in *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pp. 38–49, Berlin, Germany, March 2012.

[13] M. R. Kolahdouzan and C. Shahabi, "Voronoi-based k nearest neighbor search for spatial network databases," in *Proceedings of the Very Large Data Bases (VLDB)*, pp. 840–851, Toronto, ON, USA, August-September 2004.

[14] M. R. Kolahdouzan and C. Shahabi, "Continuous k-nearest neighbor queries in spatial network databases," in *Proceedings of the Scientific and Statistical Database Management (SSDBM)*, pp. 33–40, Aalborg, Denmark, June-July 2004.

[15] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Distance join queries on spatial networks," in *Proceedings of the ACM International Symposium on Geographic Information Systems (GIS)*, pp. 211–218, Arlington, VA, USA, Nobember 2006.

[16] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *Proceedings of the Very Large Data Bases (VLDB)*, pp. 802–813, Berlin, Germany, September 2003.

[17] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *Proceedings of the ACM SIGMOD Conference*, pp. 43–54, Vancouver, BC, Canada, June 2008.

[18] X. Huang, C. S. Jensen, and S. Saltenis, "The islands approach to nearest neighbor querying in spatial networks," in *Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD)*, pp. 73–90, Hong Kong, China, August 2005.

[19] S. Nutanong and H. Samet, "Memory-efficient algorithms for spatial network queries," in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pp. 649–660, Brisbane, Australia, April 2013.

[20] X. Huang, C. S. Jensen, H. Lu, and S. Saltenis, "S-GRID: a versatile approach to efficient query processing in spatial networks," in *Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD)*, pp. 93–111, Boston, MA, USA, July 2007.

[21] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of the ACM SIGMOD Conference*, pp. 47–57, Boston, MA, USA, 1984.

[22] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "iDistance: an adaptive B^+-tree based indexing method for nearest neighbor search," *ACM Transactions on Database Systems*, vol. 30, no. 2, pp. 364–397, 2005.

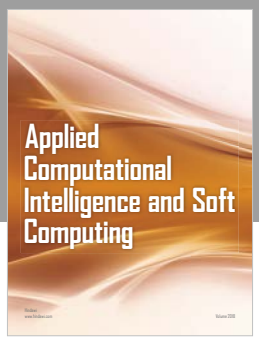
[23] L. Li and D. Taniar, "A taxonomy for distance-based spatial join queries," *International Journal of Data Warehousing and Mining*, vol. 13, no. 3, pp. 1–24, 2017.

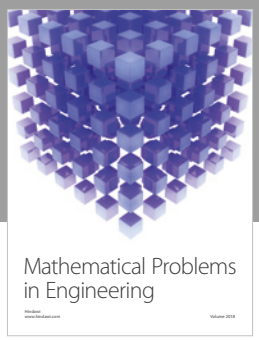
[24] L. Li, D. Taniar, M. Indrawan-Santiago, and Z. Shao, "Surrounding join query processing in spatial databases," in *Proceedings of the Australasian Database Conference (ADC)*, pp. 17–28, Brisbane, Australia, 2017.

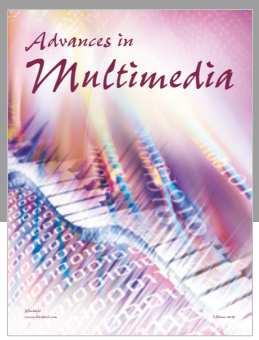
[25] C. Xiao, W. Wang, and X. Lin, "Ed-join: an efficient algorithm for similarity joins with edit distance constraints," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 933–944, 2008.

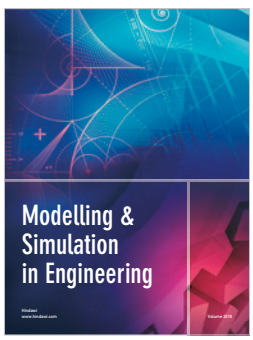
[26] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest pair queries in spatial databases," in *Proceedings of the ACM SIGMOD Conference*, pp. 189–200, Indianapolis, IN, USA, June 2000.

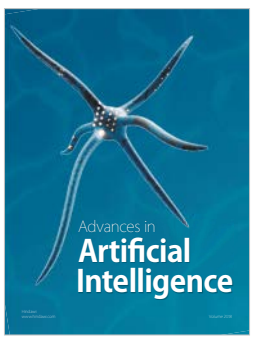
[27] Real datasets for spatial databases, 2005, https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm.

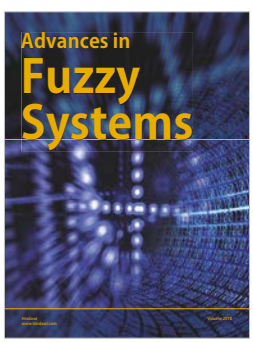


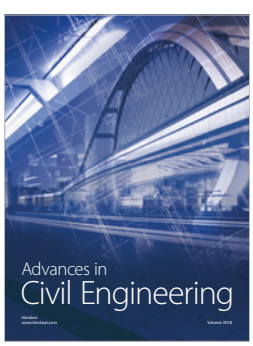


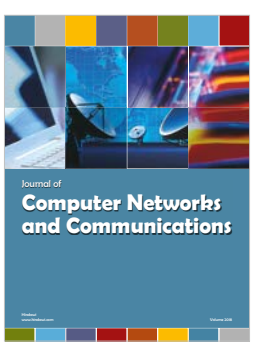


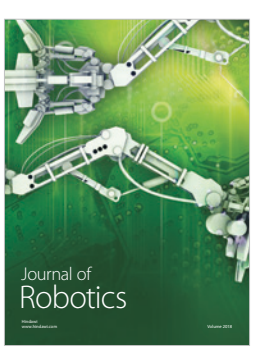


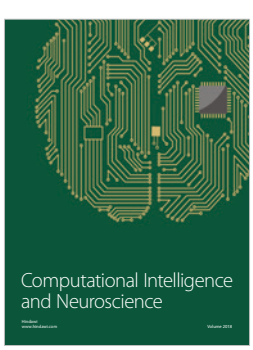












Hindawi

Submit your manuscripts at
www.hindawi.com