Information Systems 37 (2012) 611–633

Contents lists available at SciVerse ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Continuous distance-based skyline queries in road networks

Yuan-Ko Huang [a,*], Chia-Heng Chang [b], Chiang Lee [b]

[a] Department of Information Communication, Kao-Yuan University, Kaohsiung Country, Taiwan, ROC
[b] Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, ROC

ARTICLE INFO

Article history:
Received 1 April 2011
Received in revised form
19 January 2012
Accepted 17 February 2012
Recommended by: T.G.K. Calders
Available online 3 March 2012

Keywords:
Skyline query
Continuous skyline query
Continuous d_ε-skyline query
Continuous k nearest neighbor-skyline query

ABSTRACT

In recent years, the research community has introduced various methods for processing skyline queries in road networks. A skyline query retrieves the skyline points that are not dominated by others in terms of static and dynamic attributes (i.e., the road distance). This paper addresses the issue of efficiently processing continuous skyline queries in road networks. Two novel and important distance-based skyline queries are presented, namely, the *continuous d_ε-skyline query* (Cd_ε-SQ) and the *continuous k nearest neighbor-skyline query* (*Cknn-SQ*). A grid index is first designed to effectively manage the information of data objects and then two algorithms are proposed, the *Cd_ε-SQ algorithm* and the *Cd_ε-SQ$^+$ algorithm*, which are combined with the grid index to answer the Cd_ε-SQ. Similarly, the *Cknn-SQ algorithm* and the *Cknn-SQ$^+$ algorithm* are developed to efficiently process the *Cknn-SQ*. Extensive experiments using real road network datasets demonstrate the effectiveness and the efficiency of the proposed algorithms.

1. Introduction

Given a set of multidimensional points D, a *skyline query* retrieves the data points that are not dominated by any other point. Specifically, point p *dominates* another point, p', if its value in each dimension is as good or better than that of p' and is better in at least one dimension. The points retrieved by executing the skyline query are termed the *skyline points* (*SP* for short). Fig. 1 shows an example of a skyline query, where the data points correspond to hotels with two properties: rank and price (note that a higher rank and a lower price are desired in this example). Hotels o_1, o_2, and o_3 are the *SP* among the five hotels. The others (i.e., o_4 and o_5) are dominated by o_1 and o_3 in terms of rank and price. As the skyline query is a powerful tool for preference-based data analysis, and as it has a wide range of real applications, it has attracted much attention in the database community [1–11].

In recent years, several studies [12–15] have investigated how to process skyline queries over data objects (e.g., restaurants and hotels) in road networks. Different from the traditional skyline query, processing a skyline query in road networks requires taking the location of the query object into consideration. As shown in Fig. 2, the five hotels (o_1 to o_5) and the user issuing a skyline query, q, are located in a road network. In this scenario, the road distance from each hotel to q (defined as the distance along the shortest path between them) becomes one of the dimensions considered when determining the *SP*. The road distance of each hotel needs to be computed first, and then the three properties of distance, rank, and price can be used to find the *SP* of q. As can be seen, hotel o_5, which is previously dominated by o_1 and o_3 (referring to Fig. 1) is no longer dominated, because o_5 is closer to q than o_1 and o_3 (i.e., o_5 is better in the distance dimension), and thus, o_5 is promoted to a *SP*. When the query object moves in the road network, the distance between the query object and object o_i changes as well. Hence, such a dimension is referred to as a *dynamic dimension*.

* Corresponding author. Tel.: +886 7 6077212.
E-mail addresses: huangyk@cc.kyu.edu.tw, hyk@dblab.csie.ncku.edu.tw (Y.-K. Huang), derek@imus.csie.ncku.edu.tw (C.-H. Chang), leec@mail.ncku.edu.tw (C. Lee).

0306-4379/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.is.2012.02.003

hotel	rank	price (k)
o_1	4	1.5
o_2	5	3
o_3	3	1
o_4	3	1.7
o_5	2	1.5

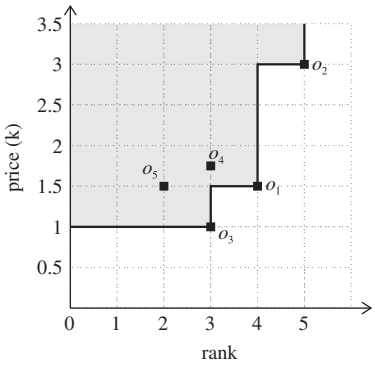
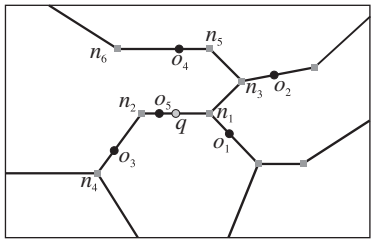


Fig. 1. Traditional skyline query.



hotel	distance	rank	price (k)
o_1	180	4	1.5
o_2	280	5	3
o_3	220	3	1
o_4	450	3	1.7
o_5	50	2	1.5

Fig. 2. Skyline query in road networks.

As opposed to the dynamic dimension, the other dimensions that remain unchanged (e.g., price and rank) are called the *static dimensions*. When a dynamic dimension is introduced into skyline query processing, the complexity of processing such a query becomes much higher, as the values in the dynamic dimension (e.g., distance) continue to change.

Previous methods for processing a skyline query in road networks have focused exclusively on determining the *SP* for a *static* query object. That is, these methods can only be used to efficiently process a *snapshot skyline query*. However, in road network applications, the query object *moves with time*. Due to the movement of the query object, the *SP* result will inevitably change. If the techniques for processing a snapshot skyline query are used, they must be repeatedly evaluated to keep producing the new *SP*. This incurs a tremendous query re-evaluation cost, especially in a highly dynamic environment where the query object frequently changes location. In this paper, a query retrieving the *SP* in a dynamic environment is referred to as a *continuous skyline query*. The goal of this paper is to study how to process such a continuous skyline query efficiently in road networks.

In some cases, there can be a large number of *SP* for a skyline query [1,8,16], with most of the *SP* are far away from the query object, and therefore, the query object is usually not interested in them. For instance, in order to find accommodation as soon as possible, a traveler will prefer a closer hotel with good quality to one that is farther away. As a result, retrieving the *SP* that are far away from the query object is simply a waste of time and incurs additional processing costs. For this reason, this paper presents two novel *distance-based* continuous skyline queries that find only the *SP* that are closer to the query object. The first is the *continuous d_ε-skyline query* (*Cd_ε-SQ* for short), and the second is the *continuous k nearest neighbor-skyline query* (*Cknn-SQ* for short). The two queries are defined as follows:

- *Cd_ε-SQ*: Given path P_q, along which query object q moves, a set of data objects S_o and distance value d_ε, *Cd_ε-SQ* retrieves a set of skyline points, SP_p, for each point p on P_q, such that the road distance from each $o \in SP_p$ to point p is less than or equal to d_ε. The *SP* satisfying *Cd_ε-SQ* (i.e., those within the distance range d_ε) are the *d_ε-SP*.
- *Cknn-SQ*: Given query path P_q, a set of data objects S_o, and a value of k, *Cknn-SQ* finds a set of skyline points, SP_p, for each point $p \subseteq P_q$, such that the number of skyline points in SP_p is equal to k and the road distance of each $o \in SP_p$ to p is less than or equal to that of each *SP* in $S_o - SP_p$ (that is, objects in SP_p are the k nearest *SP* of point p). The *SP* retrieved by *Cknn-SQ* are the *knn-SP*.

Cd_ε-SQ and *Cknn-SQ* are two useful queries that can be found in many fields and application domains. A real-world example is that of a traveler who is planning a trip from city *A* to city *B*, who may want to know which hotels are better to stay in en route. In this scenario, the traveler can issue a *Cd_ε-SQ* to find the nearest hotels (i.e., within distance range d_ε) with good quality (i.e., higher rank and lower price) during the trip. Another real-world application is collaborative promotion. Consider a set of theaters and a set of restaurants in a road network. An advertisement company can pose a *Cknn-SQ* to find the k skyline theaters that are closer to street *C*. Also, the company can find the k nearest skyline restaurants by executing a

Cknn-SQ. Then, a collaborative advertisement about the theaters and the restaurants (e.g., a restaurant offering 10% discount for customers watching movies in a theater) is broadcast to the pedestrians on street C so as to attract more customers.

Fig. 3 illustrates an example of processing a Cd_ε-*SQ* and a *Cknn-SQ*, where objects o_1 to o_4 and query object q are located in a road network, represented as a graph consisting of nodes and edges. Fig. 3(a) shows the static attributes (i.e., rank and price) of the four objects. In this example, query object q moves from node n_1 to node n_2 (the query path is $\overline{n_1n_2}$). Assume that a Cd_ε-*SQ* is issued to find the d_ε-*SP* whose road distances to q are within 300 (i.e., $d_\varepsilon = 300$). When q is located at point p_1 (as shown in Fig. 3(b)), only object o_2 is the d_ε-*SP* for the Cd_ε-*SQ*. Although o_1 is not dominated by o_2, it cannot be a d_ε-*SP* because its distance to q is greater than d_ε. When q moves to point p_2 (as shown in Fig. 3(c)), the distance of object o_4 to p_2 is equal to that of object o_2. That is, when q is located at the left side of p_2, object o_4 becomes closer to q than o_2. Thus, o_4 is better than o_2 in the distance dimension, and o_4 becomes a *SP*. As the road distances of o_2 and o_4 are both less than d_ε, they are the d_ε-*SP*. Consider another example of finding the 2*nn-SP* of q from p_1 to p_2 (i.e., processing a *C2nn-SQ*). When q is located at point p_1, objects o_2 and o_1 are the 2*nn-SP* of q. Note that although o_4 is closer to q than o_1, it still cannot be a 2*nn-SP*. This is because o_4 is dominated by o_2 in terms of distance, rank, and price. However, once q moves to the left side of p_2, o_4 is better than o_2 in distance dimension, and thus, o_4 becomes a *SP*. Since o_4 and o_2 are closer to q than o_1, the 2*nn-SP* are changed to o_4 and o_2.

To process the Cd_ε-*SQ* and the *Cknn-SQ* in road networks, the road distances between the data objects and the query object need to be computed. Although the road distance can be computed based on Dijkstra's algorithm [17] or the A* algorithm [18], which have been shown to be simple and efficient, computing the distances of all data objects will incur extremely high I/O and CPU costs, especially for a road network consisting of a large number of edges and nodes. An even greater challenge comes from the query object moving along a query path instead of being static. Hence, the distances from data objects to the query object change as the query object moves, resulting in numerous distance re-computations. It is necessary to design effective pruning strategies that can remove data objects that should not be in the query results, so as to significantly reduce the amount of distance re-computations.

Another major problem to be tackled is how to avoid repetitively processing snapshot skyline queries for any point on the query path. Consider again the example in Fig. 3. If the Cd_ε-*SQ* is processed, object o_4 is added into the query result as q moves to the left side of point p_2. That is, after point p_2, the Cd_ε-*SQ* result changes from $\{o_2\}$ to $\{o_4,o_2\}$. Similarly, when the *Cknn-SQ* is evaluated, the *Cknn-SQ* result changes after point p_2 (i.e., from $\{o_2,o_1\}$ to $\{o_4,o_2\}$). These points, after which the query result changes (e.g., point p_2), are the *result turning points*. An important characteristic of such a result turning point is that the query result between two consecutive result turning points remains the same. Based on this characteristic, the problem of repetitively processing snapshot skyline queries can be greatly reduced to finding all result turning points and their corresponding query results.

The major contributions of this paper are summarized as follows:

- Two novel distance-based continuous skyline queries are presented, namely, the Cd_ε-*SQ* and the *Cknn-SQ*, to find the skyline points in road networks. The two queries are important types of skyline queries with many real applications.
- Appropriate data structures are designed to represent a road network consisting of edges and nodes, in order to store information of the data objects in the network. Furthermore, a grid index is used to access only a small proportion of data objects while computing the Cd_ε-*SQ* and the *Cknn-SQ* results, allowing the required I/O and CPU costs to be greatly reduced.
- Two algorithms are proposed, namely, the Cd_ε-*SQ algorithm* and the Cd_ε-SQ^+ *algorithm*, which are associated with the grid index to efficiently determine the

a

hotel	rank	price(k)
o_1	2	1.5
o_2	5	2.8
o_3	1	1.5
o_4	4	3

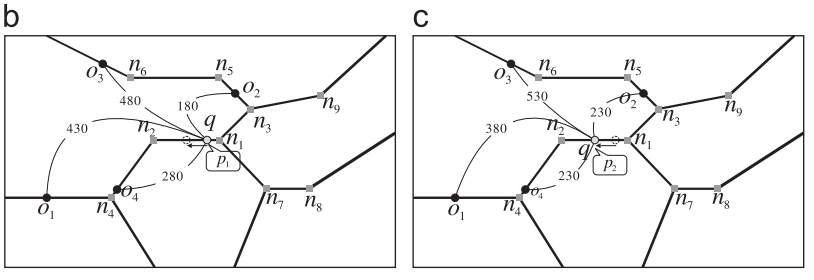


Fig. 3. Example of processing Cd_ε-*SQ* and *Cknn-SQ*. (a) Static attributes of objects. (b) Distances of objects to p_1. (c) Distances of objects to p_2.

d_ε-*SP* for any points on the query path. Similarly, two algorithms are developed, the *Cknn-SQ algorithm* and the *Cknn-SQ*$^+$ *algorithm*, to find the *knn-SP* for the query path.

- A comprehensive set of experiments is conducted. The performance results demonstrate the efficiency and the usefulness of the proposed Cd_ε-*SQ* and *Cknn-SQ* processing algorithms.

The remainder of this paper is organized as follows. In Section 2, we first review the related work for processing the skyline queries, and then discuss the previous studies on answering the continuous range and *KNN* queries in road networks. Section 3 describes the data structures used to represent a road network and the grid index used to facilitate the query processing. In Sections 4 and 5, we present how the proposed algorithms can be used to answer the Cd_ε-*SQ* and the *Cknn-SQ*, respectively. Section 6 shows extensive experiments on the performance of our methods. In Section 7, we conclude the paper with directions on future work.

2. Related work

Processing skyline queries in road networks is an emerging research topic in recent years. In this section, we first review the existing methods for processing the traditional skyline queries. Then, we discuss the related work that investigates how to answer the skyline queries in road networks. Finally, we survey some related work that focuses on answering the continuous range and *KNN* queries in road networks, which are similar to the Cd_ε-*SQ* and the *Cknn-SQ* presented in this paper.

2.1. Traditional skyline queries

The skyline query, also known as the Maximal Vector Problem [19,20], was first studied in the area of computational geometry. Several main-memory algorithms had been proposed to solve the maximal vector problem. However, those main-memory algorithms are inefficient for the skyline query, due to the large sizes of data sets. Borzsonyi et al. [1] first introduced the skyline operator into database systems and proposed two algorithms, Block-Nested-Loops (BNL) and Divided-and-Conquer (D&C), to evaluate the skyline result. BNL sequentially scans the entire dataset and compares each new data object to all skyline candidates kept in memory. Only the data objects not dominated by others are kept as skyline candidates. D&C recursively divides the dataset into partitions so as to fit in memory, and evaluates the local skyline points for each of them. Then, the global skyline points are obtained by progressively merging the local ones. However, both BNL and D&C need to process the entire dataset before reporting skyline points. To progressively report skyline points, Chomicki et al. [2] proposed the Sort-Filter-Skyline (SFS) algorithm which is an improvement of BNL. The idea of SFS is to sort the entire dataset in advance according to a monotonic preference function (e.g., sum of dimension values). Sorting the dataset guarantees that each data object cannot be dominated by ones behind it in the sorted data list. Therefore, a data object not dominated by others in front of it can be directly reported as a skyline point. A main disadvantage of SFS is that all data objects should be scanned at least once after sorting. Bartolini et al. [21] further proposed the Sort and limit skyline algorithm (SaLSa) to avoid scanning the complete set of sorted data objects. Tan et al. [10] developed two progressive methods to improve the skyline query performance. The first method encodes each data object into a bit vector, and then vectors of all objects form a bitmap. Benefiting from the bitmap transformation, a data object can be quickly determined whether it is a skyline point or not by applying a bitwise *and* operation. The second method is to transform high dimensional data objects into a single dimensional space so that a B$^+$-tree can be used to index them. Then, the local skyline points are found by traversing the B-tree and to be merged into a global one. Kossmann et al. [5] observed that the nearest object to the origin must be a skyline point. Based on this observation, they first use an R-tree to find the nearest neighbor (NN), and then based on the NN the data space is partitioned into one dominated subspace and two non-determined subspaces. The data objects in the dominated subspace can be pruned, and the next nearest neighbor is iteratively found in each non-determined subspace. The subspace is recursively partitioned until all skyline points are reported. However, a major problem of such a method is that multiple traversals of the R-tree are required for determining the skyline points. To overcome this problem, Papadias et al. [8] proposed a Branch and Bound Skyline (BBS) algorithm to retrieve the skyline points by prioritizing accesses at partially dominated nodes of the R-tree. All of these research mentioned above focused exclusively on processing snapshot skyline queries over data objects with static dimensions.

There have been some work [22–24] considering the problem of processing continuous skyline queries in streaming databases, in which the dimension value of each data object changes with time. As the changing dimension values of each object would inevitably result in different skyline points at different time instants, these works mainly investigated how to efficiently and rapidly update the skyline points. Zhang et al. [25] further addressed the issue of processing continuous skyline queries for streaming data with uncertainty. Recently, Huang et al. [26] were the first to take into consideration the distance between data objects in processing the continuous skyline queries. However, their approach aims at the Euclidean space rather than road networks as we considered in this paper. As the road distance needs to be computed based on the connectivity of the network rather than simply using the two objects' locations, efficiently processing continuous skyline queries in road networks is much more complicated.

To process the continuous skyline queries, we design a pruning strategy based on the Euclidean lower-bound property that *the road distance of an object to the query object is always greater than or equal to its Euclidean distance*. If object's road distance is close to its Euclidean distance, the pruning strategy will perform well because much expensive road distance computation can be avoided by considering their Euclidean distances. As such,

our pruning strategy utilizes the Euclidean distance to prune the non-qualifying objects so that the algorithm needs to compute the road distances of candidates only.

2.2. Skyline queries in road networks

In recent years, processing skyline queries in road networks has received considerable attention. Deng et al. [12] extended the concept of the spatial skyline [27] to road networks and presented the *multi-source skyline query* (and MSQ for short). Given a set of m query objects and a set of n data objects in a road network, each data object o is mapped to a m-dimensional point, where the value of the ith dimension refers to the road distance between o and the ith query object. Then, MSQ retrieves the skyline points that are not dominated in terms of the m dimensions. Deng et al. proposed three algorithms: Euclidean Distance Constraint (EDC), Lower Bound Constraint (LBC), and Collaborative Expansion (CE), to solve the MSQ problem. To improve the search performance, EDC and LBC utilize Euclidean distance as the lower bound of road distance to prune data objects. As for CE, the pruning strategy is to start from the m query objects to search the road network for candidate skyline objects. Once an object has been visited m times, those objects that have never been visited can be pruned. However, these algorithms may generate too many candidates and cause unnecessary road distance computation. Hence, Zou et al. [15] proposed the Shared Shortest Path (SSP) algorithm associated with the Shortest Path Tree (SP-Tree) to overcome the problems. The criteria for determining the skyline points in the above methods are based on the road distances between data objects and query objects. Other studies [28,13] considered the skyline problem in *multi-cost transportation networks* (MCN), where each edge (i.e., road segment) is associated with multiple cost values and the skyline points are determined based on these cost values. Huang et al. [29] studied another skyline problem of finding the skyline points that are not dominated in terms of only two attributes: (1) their network distance to a query location q and (2) the detour distance from q's predefined route on the road network.

Jang et al. [30] were the first to address the issue of processing continuous skyline queries in road networks. The idea is to pre-compute a range R for each data object o such that if the query object is within R, then o must be a skyline point in terms of dynamic dimension (i.e., its road distance to the query object) and static dimension. Then, the skyline points of the query object can be determined based on the pre-computed ranges of all data objects. However, two major problems have limited the applicability of this method. The first problem is that pre-computing all object ranges incurs tremendous processing cost, especially for a road network with large size. The second is that the skyline result cannot provide useful information to the user because the skyline points far away from the query object are also included in the query result. Hence, the method in [30] cannot be applied to Cd_ε-*SQ* and *Cknn-SQ* (in which the skyline result is affected by the user given d_ε and k). More appropriate and efficient methods must be designed to answer these two types of queries.

2.3. Continuous range and KNN queries

Jensen et al. [31] proposed a data model and a definition of abstract functionality required for *KNN* queries in spatial networks databases. To compute network distances, they adopt the Dijkstra's algorithm for online evaluation of the shortest path. Papadias et al. [32] described a framework that integrates network and Euclidean information, and answers static range and *KNN* queries. Kolahdouzan et al. [33] proposed a solution-based approach for static *KNN* queries in spatial networks. Their approach, called VN^3, precalculates the network Voronoi polygons (NVPs) and some network distances, and then processes the *KNN* queries based on the properties of the Network Voronoi diagrams. Kolahdouzan et al. [34] and Cho et al. [35] developed different effective techniques, UBA and UNICONS, to reduce the number of K NN evaluations by allowing the *KNN* result to be valid for a time interval. However, the techniques are adequately designed to deal with continuous *KNN* queries over static objects. Mouratidis et al. [36] first addressed the issue of continuous monitoring *KNN* on moving objects and proposed an incremental monitoring algorithm (IMA) to re-evaluate query at those time instants at which updates occur. Due to the nature of discrete location updates, IMA would return incorrect *KNN* result as long as there exist some time points within two consecutive updates at which the *KNN* set changes. To overcome this limitation, Huang et al. [37] proposed a CK NN method to determine the *KNN* at each timestamp under the situation that all objects (including the query object) move continuously in a road network.

Although the methods proposed for the above queries can be used to find the objects within a distance range or the k nearest objects, they cannot directly applied to answering the Cd_ε-*SQ* and the *Cknn-SQ* presented in this paper. For the Cd_ε-*SQ* (the *Cknn-SQ*), the objects within the distance range d_ε (the k nearest objects) may still be dominated by other objects and cannot be the d_ε-*SP* (the *knn-SP*). Thus, efficient technique for determining the dominance relationship between objects is required in processing the Cd_ε-*SQ* and the *Cknn-SQ,* especially for continuous queries with a long query path. Moreover, finding the result turning points for the Cd_ε-*SQ* and the *Cknn-SQ* is more complex than that for the continuous range and *KNN* queries. This is because both the road distance and the dominance relationship of objects are the criteria for determining the result turning points (however, only the road distance is considered in processing the continuous range and *KNN* queries).

3. Data structures and grid index

In the proposed system, all data objects are located in a road network, which is represented as an undirected weighted graph consisting of a set of nodes and a set of edges. Since databases contain large amounts of information that need to be maintained (e.g., information of data

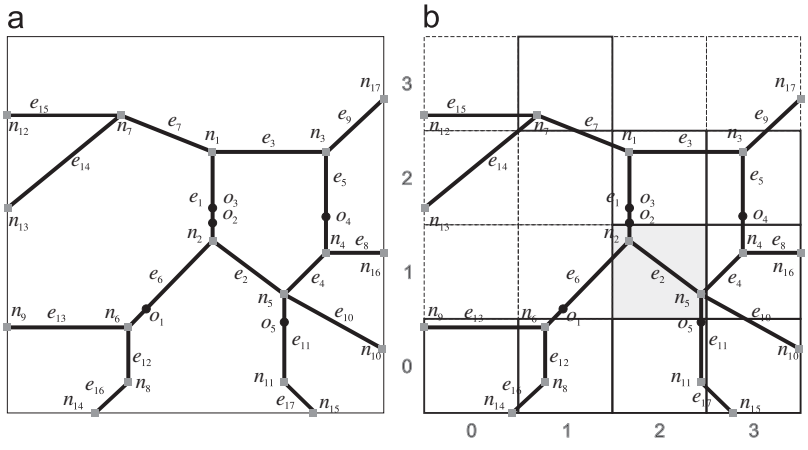
objects, nodes, and edges), a grid index is used to efficiently manage such information. However, it should be noted that the proposed algorithms (which will be discussed later) are generic and can be used together with the existing popular index structures.

The data space covering the entire road network is first divided into $N \times N$ grid cells, and then for each grid cell $g(i,j)$, information of the network and data objects is stored in the following three tables, respectively:

- The *edge table* T_{edge} stores the intersection information of each edge e with cell $g(i,j)$. The information contains: (1) the nodes n_i and n_j connecting edge e if n_i is the node within $g(i,j)$; (2) the length *len*; and (3) cell $g(m,n)$ enclosing node n_j.
- The *node table* T_{node} maintains the coordinate information (x,y) of each node that is enclosed in $g(i,j)$.
- The *object table* T_{obj} stores the information of each object o that is located on the edge intersecting cell $g(i,j)$. The information includes: (1) the coordinates of o; (2) the edge e containing o; (3) the static attributes *SA* (e.g., price and rank of a hotel); and (4) a set S_{DO} of the objects dominated by o in terms of *SA*. Note that the S_{DO} of object o is preprocessed by comparing its *SA* to those of the other objects. As the *SA* of each object is static, the preprocessed S_{DO} will remain unchanged.

To exemplify the information maintained in the three tables, a road network is shown in Fig. 4(a), which consists of 15 edges and 17 nodes. As shown in Fig. 4(b), the data space covering the road network is divided into 4×4 grid cells, with the cell numbers ranging from $g(0,0)$ to $g(3,3)$. Taking cell $g(2,1)$ (depicted as a grey rectangle) as an example, the detailed information of T_{edge}, T_{node}, and T_{obj} for cell $g(2,1)$ is shown in Fig. 4(c). As edges e_1, e_2, e_4, e_6, e_{10}, and e_{11} intersect cell $g(2,1)$, T_{edge} stores the information of the six edges. T_{node} maintains the coordinates of nodes n_2 and n_5, which are enclosed by $g(2,1)$. In T_{obj}, the information of objects o_1, o_2, o_3, and o_5 is kept because they are located on the edges intersecting cell $g(2,1)$ (i.e., edges e_6, e_1, and e_{11}). Note that some grid cells may exist that do not have any information about edges, nodes, and objects, if that information can be found from other cells. For instance, edge e_6 (and object o_1) intersects (and is enclosed in) cell $g(1,1)$. Hence, its information can be found in the tables of cell $g(1,0)$ or $g(2,1)$, and thus $g(1,1)$ need not maintain any information. With such a strategy, only the cells enclosing the nodes need to maintain T_{edge}, T_{node}, and T_{obj} (rather than all of the $N \times N$ cells), and the storage cost can therefore be reduced. In the following, we describe how to compute the road distance, denoted as $d_{o,q}$, between object o and query object q, by taking advantage of this grid index.

For ease of exposition, the road network presented in Fig. 4 is used again to illustrate the process of computing the road distance between objects. Suppose that query object q is located at node n_2 and that the road distance between objects q and o_4 needs to be computed (i.e., computing d_{n_2,o_4}). To enhance the performance of the computation, a min-heap H is used to store the nodes visited so far, in which the node with the minimum



c

T_{edge}

e	n_i	n_j	*len*	$g(m,n)$
e_1	n_2	n_1	19	$g(2,2)$
e_2	n_2	n_5	18	$g(2,1)$
e_4	n_5	n_4	13	$g(3,1)$
e_6	n_2	n_6	24	$g(1,0)$
e_{10}	n_5	n_{10}	22	$g(3,0)$
e_{11}	n_5	n_{11}	17	$g(2,0)$

T_{node}

n	(x,y)
n_2	(38,32)
n_5	(53,22)

T_{obj}

o	(x,y)	e	*SA*(*price*,*rank*)	S_{DO}
o_1	(26,19)	e_6	(120,3)	$\{\varnothing\}$
o_2	(38,35)	e_1	(80,4)	$\{o_1, o_3\}$
o_3	(38,38)	e_1	(90,3)	$\{o_1\}$
o_5	(52,23)	e_{11}	(60,2)	$\{\varnothing\}$

Fig. 4. The grid index structure. (a) A road network. (b) A grid index. (c) Information of $g(2,1)$.

distance to node n_2 is first de-heaped. Initially, the cell $g(2,1)$ enclosing node n_2 is accessed to obtain the information of edges e_1, e_2, and e_6 that connect with n_2. Having obtained the edge information, the distances from nodes n_2 to n_1, n_5, and n_6 are computed. Then, the three nodes with their distances are en-heaped into H, as shown in Fig. 5(a). As none of the edges (e_1, e_2, and e_6) contain object o_4, it can be checked to see if it is located on the edges connecting nodes n_1, n_5, and n_6. As node n_5 has the minimum distance to n_2 in H, it is first de-heaped, and then the distances from n_2 to n_4, n_{10}, and n_{11} are computed, as shown in Fig. 5(b). Again, as none of the edges contain object o_4, n_1 becomes the next node to visit, as its distance to n_2 is the minimum in H. Since node n_1 is enclosed in cell $g(2,2)$, the cell is accessed to get the information of edges e_3 and e_7 (which connect with n_1) and to compute the distances d_{n_2,n_3} and d_{n_2,n_7} (as shown in Fig. 5(c)). Subsequently, n_6 becomes the next node with the minimum distance to n_2, and it is de-heaped from H. The cell $g(1,0)$ enclosing n_6 is accessed to compute the distances of nodes n_8 and n_9 to n_2, respectively (as shown in Fig. 5(d)). Finally, when cell $g(3,1)$ is accessed, a path from n_2 to o_4 can be found, allowing the road distance d_{n_2,o_4} to be computed, using the formula $18+13+6=37$ (as shown in Fig. 5(e)). So far, the road distances to n_2 of all nodes in H are greater than or equal to 37, meaning that the distances along the paths, starting from n_2 and passing through these nodes to o_4, cannot be less than 37. Therefore, the road distance d_{n_2,o_4} is directly set to 37 and the remaining cells need not be accessed. Benefiting from the grid index, the road distance between objects can be efficiently computed, which helps to reduce the cost of query processing.

The following sections present the proposed algorithms associated with the grid index to determine the *Cd$_\varepsilon$-SQ* and *Cknn-SQ* results.

4. Processing techniques for *Cd$_\varepsilon$-SQ*

Given a query path P_q, *Cd$_\varepsilon$-SQ* retrieves the *d$_\varepsilon$-SP* for each point $p \subseteq P_q$, such that the road distance between each *d$_\varepsilon$-SP* and p is less than or equal to d_ε. In this section, the *Cd$_\varepsilon$-SQ algorithm* and the *Cd$_\varepsilon$-SQ$^+$ algorithm* are developed to efficiently determine the *d$_\varepsilon$-SP*. The main idea of the proposed algorithms is to divide the query path P_q into a set of edges, S_e, which are then considered sequentially in determining the *d$_\varepsilon$-SP*. For each edge $e \in S_e$, the procedure of the *Cd$_\varepsilon$-SQ* approaches (i.e., both the *Cd$_\varepsilon$-SQ* algorithm and the *Cd$_\varepsilon$-SQ$^+$* algorithm) includes (1) a *global SP determination phase*: obtains a set of global *SP*, denoted as *GSP*, by accessing only a few cells of the grid index, such that object $o \in GSP$ must be a *d$_\varepsilon$-SP* for point p on edge e (i.e., o is a *SP* for point p and its road distance $d_{p,o} \le d_\varepsilon$), and (2) a *result turning point determination phase*: determines the result turning points on edge e, such that the *d$_\varepsilon$-SP* between two consecutive result turning points remain the same, and finds the corresponding *d$_\varepsilon$-SP* of these points.

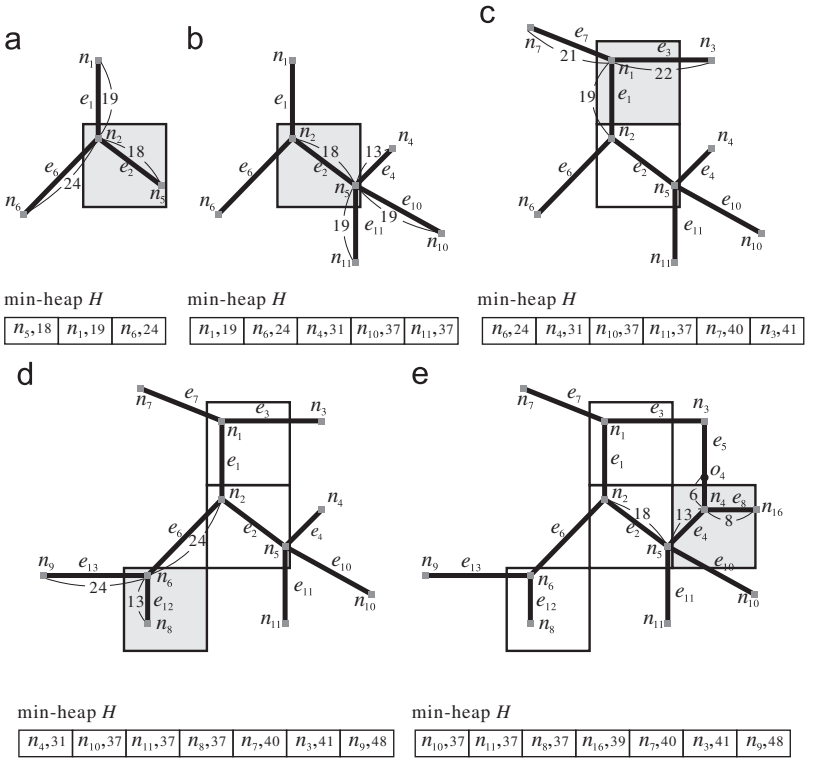


Fig. 5. Computing the road distance between n_2 and o_4. (a) Accessing $g(2,1)$. (b) Accessing $g(2,1)$. (c) Accessing $g(2,2)$. (d) Accessing $g(1,0)$. (e) Accessing $g(3,1)$.

The Cd_ε-SQ algorithm and the Cd_ε-SQ^+ algorithm are described in Sections 4.1 and 4.2, respectively. Finally, a complexity analysis of the proposed Cd_ε-SQ approaches is given in Section 4.3.

4.1. Cd_ε-SQ algorithm

In the global *SP* determination phase of the Cd_ε-SQ algorithm, the purpose of obtaining the *GSP* for edge e is to greatly reduce the number of objects that need to be considered in finding the d_ε-SP. That is, the objects not in the *GSP* can be pruned immediately. Here, the following theorem is provided to facilitate the determination of the *GSP*.

Theorem 4.1. *Given a set of d_ε-SP, denoted as SP_p^ε, for a point p on an edge e connecting two nodes n_s and n_e, we have $SP_p^\varepsilon \subseteq SP_{n_s}^\varepsilon \bigcup SP_{n_e}^\varepsilon \bigcup S_e$, where $SP_{n_s}^\varepsilon$ and $SP_{n_e}^\varepsilon$ are the d_ε-SP sets for nodes n_s and n_e, respectively, and S_e is the set of objects located on edge e.*

Proof. We prove the theorem by contradiction. Assume that an object $o \in SP_p^\varepsilon$ but $o \notin SP_{n_s}^\varepsilon \bigcup SP_{n_e}^\varepsilon \bigcup S_e$. As $o \notin SP_{n_s}^\varepsilon \bigcup SP_{n_e}^\varepsilon$, one of the following four conditions holds: (1) both the distance $d_{n_s,o}$ from n_s to o and the distance $d_{n_e,o}$ from n_e to o are greater than d_ε, (2) an object $o' \in SP_{n_s}^\varepsilon$ dominates o in terms of the static attributes and the dynamic attribute (i.e., $d_{n_s,o'} \le d_{n_s,o}$), and the road distance $d_{n_e,o}$ from n_e to o is greater than d_ε, (3) the road distance $d_{n_s,o} > d_\varepsilon$, and o is dominated by an object $o'' \in SP_{n_e}^\varepsilon$ whose distance $d_{n_e,o''} \le d_{n_e,o}$, or (4) there exists an object $o' \in SP_{n_s}^\varepsilon$ (and $o'' \in SP_{n_e}^\varepsilon$) dominating o and the distance $d_{n_s,o'} \le d_{n_s,o}$ (and $d_{n_e,o''} \le d_{n_e,o}$).

- Condition 1 holds: Due to $o \notin S_e$ (i.e., o is not on edge e), the distances along the paths passing through n_s or n_e to p must be greater than d_ε. Therefore, o cannot be a d_ε-SP for point p.
- Condition 2 holds: As o is dominated by o' in terms of static attributes, it can be the d_ε-SP for point p only if its distance $d_{p,o}$ (i.e., the length of the path passing through n_s to p) is less than $d_{p,o'}$. However, $o \notin S_e$ indicates that $d_{p,o} \ge d_{p,o'}$, and thus o is not a d_ε-SP.
- Condition 3 holds: Similar to the reason in Condition 2.
- Condition 4 holds: Because both objects o' and o'' dominate o, the necessary condition for o to be a d_ε-SP is that $d_{p,o} < d_{p,o'}$ and $d_{p,o} < d_{p,o''}$. However, $o \notin S_e$ implies that either $d_{p,o} \ge d_{p,o'}$ or $d_{p,o} \ge d_{p,o''}$, so that o is not a d_ε-SP.

No matter which condition holds, o cannot be a d_ε-SP for point p. This leads to a contradiction that $o \in SP_p^\varepsilon$. □

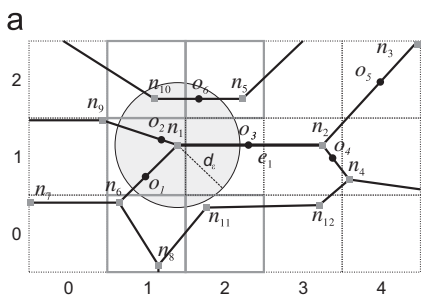
The above theorem adequately demonstrates that if the *GSP* for edge e is set to $SP_{n_s}^\varepsilon \bigcup SP_{n_e}^\varepsilon \bigcup S_e$, then the d_ε-SP for any point p on edge e will be included in the *GSP*. As such, it is only necessary to retrieve the objects on edge e and the d_ε-SP for nodes n_s and n_e in order to obtain the *GSP*. Next, we discuss how to efficiently retrieve the d_ε-SP for nodes n_s and n_e by accessing only a few cells of the grid index.

Recall that object o is the d_ε-SP for node n only if: (1) it cannot be dominated by other objects in terms of static and dynamic attributes; and (2) the road distance $d_{n,o}$ is less than or equal to d_ε. This means that the objects whose distances to n are greater than d_ε can be pruned, no matter if they are dominated by others or not. However, computing the road distances of all objects to compare with d_ε will incur significant CPU and I/O costs. In order to filter out those distant objects without computing their road distances, a pruning strategy is designed based on the property of *the road distance $d_{n,o}$ always being greater than or equal to their Euclidean distance $d_{n,o}^E$*. That is, the objects whose $d_{n,o}^E > d_\varepsilon$ are directly pruned because their road distances must be greater than d_ε. The grid cells that intersect the circle centered at node n with radius d_ε are accessed to retrieve objects. If the Euclidean distance $d_{n,o}^E$ of a retrieved object o is greater than d_ε, then its road distance $d_{n,o}$ need not be computed. Otherwise, $d_{n,o}$ is computed using the method mentioned in Section 3 and compared with d_ε. For each object o whose $d_{n,o} \le d_\varepsilon$, looking up the S_{DO} information can determine whether o is dominated by the other objects within the distance range d_ε, in terms of the static attributes. Only those objects not dominated by others in terms of either static attributes or dynamic attribute (i.e., the road distance to n) are the d_ε-SP for node n. By performing the above procedure, the d_ε-SP sets, $SP_{n_s}^\varepsilon$ and $SP_{n_e}^\varepsilon$, for nodes n_s and n_e can be determined, respectively. Algorithm 1 gives the details of how the *GSP* for edge e is obtained.

Algorithm 1. The global *SP* determination phase of the Cd_ε-SQ algorithm.

Input: A grid index, a distance d_ε, an edge e connecting two nodes n_s and n_e
Output: The *GSP* for edge e
/* determining $SP_{n_s}^\varepsilon$ for node n_s */
access the cells that intersect the circle centered at n_s with radius d_ε to obtain T_{edge}, T_{node}, and T_{obj};
foreach *object o_i stored in T_{obj}* **do**
  compute the Euclidean distance d_{n_s,o_i}^E;
  if $d_{n_s,o_i}^E \le d_\varepsilon$ **then**
    compute the road distance d_{n_s,o_i} by accessing the grid index;
foreach *object o_i whose $d_{n_s,o_i} \le d_\varepsilon$* **do**
  /* o_i is not dominated by others in terms of static attributes */
  if *$o_i \notin S_{DO}$ of the other objects within the distance range d_ε* **then** insert o_i into $SP_{n_s}^\varepsilon$;
  /* o_i is not dominated by others in terms of dynamic attribute */
  elseif *$o_i \in o'.S_{DO}$ and $d_{n_s,o_i} < d_{n_s,o'}$* **then** inserto_i into $SP_{n_s}^\varepsilon$;
determine $SP_{n_e}^\varepsilon$ for node n_e in a similar way;
retrievethe set S_e of objects on edge e;
set *GSP* to $SP_{n_s}^\varepsilon \bigcup SP_{n_e}^\varepsilon \bigcup S_e$;

The example in Fig. 6 is used to illustrate how to obtain the *GSP* for edge e_1 connecting nodes n_1 and n_2. Initially, the d_ε-SP set for node n_1 (i.e., $SP_{n_1}^\varepsilon$) is determined, which is shown in Fig. 6(a). As the circle centered at n_1 with radius d_ε intersects five cells $g(1,0)$, $g(1,1)$, $g(1,2)$, $g(2,0)$, and $g(2,2)$ (cell $g(2,1)$ does not store any information because no node is contained in it, the objects o_1, o_2, o_3, and o_6 stored in the T_{obj} tables of the five cells will be retrieved. Because the Euclidean distance d_{n_1,o_3}^E between node n_1



o_i	$d_{n_1,o}$	$d_{n_2,o}$	S_{DO}
o_1	80	340	$\{o_5\}$
o_2	30	290	$\{o_3, o_4, o_5\}$
o_3	130	130	$\{o_5\}$
o_4	290	30	$\{o_3, o_5\}$

Fig. 6. The global *SP* determination phase of the Cd_ε-*SQ* algorithm. (a) Determining the SP^ε_{n1} for node n_1. (b) Obtaining the *GSP* for edge e_1.

and object o_3 is greater than d_ε, the road distance of object o_3 need not be computed. As for object o_6, its road distance d_{n_1,o_6} to node n_1 is computed and is found to not be the d_ε-*SP*, as $d_{n_1,o_6} > d_\varepsilon$. Only the two objects o_1 and o_2 are within the distance d_ε from node n_1 and could be the d_ε-*SP*. Having looked up the S_{DO} information, it is known that $o_1 \notin o_2.S_{DO}$ and $o_2 \notin o_1.S_{DO}$ (meaning that o_1 and o_2 do not dominate each other in terms of the static attributes), and thus, both objects are the d_ε-*SP* for node n_1 ($SP^\varepsilon_{n_1} = \{o_1, o_2\}$). The $SP^\varepsilon_{n_2}$ for node n_2 can be obtained in a similar way, which is equal to $\{o_4\}$. As object o_3 is located on edge e_1 (i.e., $S_{e_1} = \{o_3\}$), the *GSP* for edge e_1 is finally set to $\{o_1, o_2, o_3, o_4\}$ as shown in Fig. 6(b). The four objects will be considered in the result turning point determination phase to find the result turning points with the corresponding d_ε-*SP*.

The main idea of the result turning point determination phase is to set up landmarks on edge e and then find the result turning points. Only the landmarks are possible result turning points. There are four types of landmarks, which are defined as follows:

- $L^{\vdash}_o$: Assume that the road distance $d_{q,o}$ between object o and query object q is greater than d_ε, and that q *moves toward* o. $L^{\vdash}_o$ is the landmark on edge e at which $d_{q,o} = d_\varepsilon$. That is, object o cannot be a d_ε-*SP* of q until q passes through $L^{\vdash}_o$. For example, in Fig. 7(a) where $d_\varepsilon = 100$, the landmark $L^{\vdash}_{o_3}$ of object o_3 refers to the point at which the road distance to n_1 is equal to 30. Before q reaches $L^{\vdash}_{o_3}$, the road distance $d_{q,o_3} > 100$, therefore, o_3 cannot be the d_ε-*SP*.
- $L^{\dashv}_o$: Assume that $d_{q,o} < d_\varepsilon$, and that q *moves away from* o. Once q passes through the landmark $L^{\dashv}_o$, the road distance $d_{q,o}$ between o and q is greater than d_ε, therefore, o is not the d_ε-*SP* of q. For instance, object o_3, shown in Fig. 7(a), cannot be the d_ε-*SP* when q moves to the right of $L^{\dashv}_{o_3}$, as the distance to n_1 is equal to 230.
- $L^{\vdash}_{o\backslash o'}$: Assume that object o dominates another object, say o', in terms of the static attributes (i.e., $o' \in o.S_{DO}$). The landmark $L^{\vdash}_{o\backslash o'}$, at which $d_{q,o'} = d_{q,o}$, means that if q is on the left of $L^{\vdash}_{o\backslash o'}$, then o cannot dominate o' in terms of the dynamic attribute (i.e., the road distance $d_{q,o'} < d_{q,o}$). On the other hand, o dominates o' when q is on the right of $L^{\vdash}_{o\backslash o'}$. Consider objects o_3 and o_4 in Fig. 7(a). As object $o_3 \in o_4.S_{DO}$ (referring to Fig. 6(b)), there exists $L^{\vdash}_{o_4\backslash o_3}$ on the point whose distance to n_1 is equal to 210. Note that if $L^{\vdash}_{o\backslash o'}$ is between $L^{\dashv}_{o'}$ and $L^{\vdash}_o$ (i.e., within the range $[L^{\dashv}_{o'}, L^{\vdash}_o]$), then $L^{\vdash}_{o\backslash o'}$ can be ignored, because both o and o' are not d_ε-*SP* when q moves within $[L^{\dashv}_{o'}, L^{\vdash}_o]$.
- $L^{\dashv}_{o\backslash o'}$: Assume that $o' \in o.S_{DO}$. Before q reaches the landmark $L^{\dashv}_{o\backslash o'}$, o can dominate o' in terms of the static and dynamic attributes. However, once q passes $L^{\dashv}_{o\backslash o'}$, o cannot dominate o' because $d_{q,o'} < d_{q,o}$. Similarly, when $L^{\dashv}_{o\backslash o'}$ is within $[L^{\dashv}_{o'}, L^{\vdash}_{o'}]$ (e.g., $L^{\dashv}_{o_2\backslash o_4}$ in Fig. 7(a)), it is ignored.

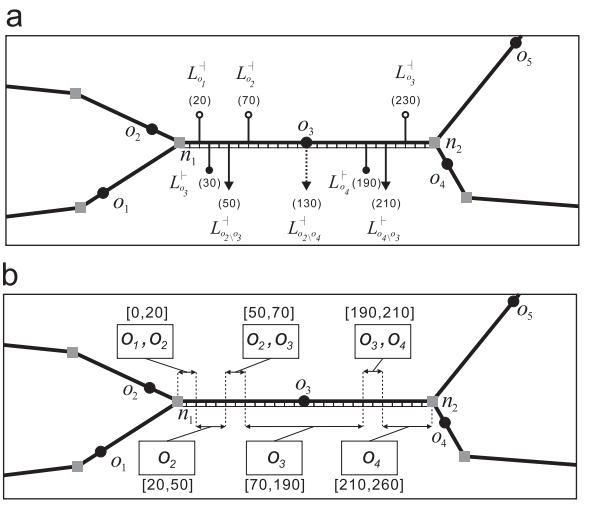


Fig. 7. The result turning point determination phase of the Cd_ε-*SQ* algorithm. (a) Landmarks of *GSP*. (b) Determining the result turning points.

Having obtained the *GSP* for edge e, the four landmarks of each object in the *GSP* are determined on-the-fly (note that only the objects in the *GSP* need to be considered in finding the landmarks). First, we describe how to obtain the landmarks $L^{\vdash}_o$ and $L^{\dashv}_o$ of each object $o \in GSP$. If object $o \in S_e$ (i.e., o is on edge e), $L^{\vdash}_o$ and $L^{\dashv}_o$ are set to the points on edge e at which the distances to n_s and n_e are equal to $d_{n_s,o} - d_\varepsilon$ and $d_{n_e,o} - d_\varepsilon$, respectively. Otherwise, it is either $o \in SP^\varepsilon_{n_s}$ for node n_s or $o \in SP^\varepsilon_{n_e}$ for node n_e. In the case where $o \in SP^\varepsilon_{n_s}$ (or $o \in SP^\varepsilon_{n_e}$), landmark $L^{\dashv}_o$ (or $L^{\vdash}_o$) on edge e is the only one whose distance to n_s (or n_e) is equal to

$d_\varepsilon - d_{n_s,o}$ (or $d_\varepsilon - d_{n_e,o}$). The number of landmarks $L_o^{\vdash}$ and $L_o^{\dashv}$ is at most $2 \cdot |S_e| + |SP^\varepsilon_{n_s}| + |SP^\varepsilon_{n_e}|$, where $|S_e|$, $|SP^\varepsilon_{n_s}|$, and $|SP^\varepsilon_{n_e}|$ are the numbers of objects in S_e, $SP^\varepsilon_{n_s}$, and $SP^\varepsilon_{n_e}$, respectively. Next, the determination of the landmark $L^{\vdash}_{o\backslash o'}$ is divided into two cases: (1) object $o' \in SP^\varepsilon_{n_s}$ but $o' \notin S_e$ and (2) object $o' \in S_e$. In the first case, if there exists the landmark $L^{\vdash}_{o\backslash o'}$ on edge e, then $L^{\vdash}_{o\backslash o'}$ is set to the point at which the distance to n_s is equal to $(d_{n_s,o} - d_{n_s,o'})/2$. In the later case, as object o' is on edge e, $L^{\vdash}_{o\backslash o'}$ is the point where the distance to n_s is equal to $(d_{n_s,o} + d_{n_s,o'})/2$. Let $|SP^\varepsilon_{n_s} - S_e|$ (and $|SP^\varepsilon_{n_e} - S_e|$)be the number of objects in $SP^\varepsilon_{n_s}$ (and $SP^\varepsilon_{n_e}$) but not in S_e. The number of landmark $L^{\vdash}_{o\backslash o'}$ is at most $|SP^\varepsilon_{n_s} - S_e| \cdot (|GSP| - |SP^\varepsilon_{n_s} - S_e|) + |S_e| \cdot |SP^\varepsilon_{n_e} - S_e|$. Finally, the landmark $L^{\dashv}_{o\backslash o'}$ is obtained according to the following cases: (1) object $o \in SP^\varepsilon_{n_s}$ but $o \notin S_e$ and (2) object $o \in S_e$. In the first (or second) case, the landmark $L^{\dashv}_{o\backslash o'}$ is set to the point at which the distance to n_s is equal to $(d_{n_s,o'} - d_{n_s,o})/2$ (or $(d_{n_s,o'} + d_{n_s,o})/2$). The maximum number of $L^{\dashv}_{o\backslash o'}$ is similar to that of $L^{\vdash}_{o\backslash o'}$. Therefore, the total number of the determined landmarks is at most $2 \cdot |S_e| + |SP^\varepsilon_{n_s}| + |SP^\varepsilon_{n_e}| + 2 \cdot (|SP^\varepsilon_{n_s} - S_e| \cdot (|GSP| - |SP^\varepsilon_{n_s} - S_e|) + |S_e| \cdot |SP^\varepsilon_{n_e} - S_e|)$.

Algorithm 2 illustrates how to determine the result turning points with the corresponding d_ε-*SP* by using the landmarks of objects in the *GSP*. Initially, queue Q is utilized to keep the information of the object landmarks. If the landmark $L_o^{\vdash}$ (or $L_o^{\dashv}$) of object $o \in GSP$ is located on edge e, it will be inserted into Q so as to know the location where o will be within (or out of) distance range d_ε. Also, object o's landmark $L^{\vdash}_{o\backslash o'}$ (or $L^{\dashv}_{o\backslash o'}$) on edge e is kept in Q so that the point through which o can (or cannot) dominate o' is determined. Then, the landmarks in Q are sorted in ascending order of their distances to node n_s, and will be considered sequentially to determine which are the result turning points:

- When the landmark $L_o^{\vdash}$ of object o is considered: if there exists a landmark $L^{\dashv}_{o'\backslash o}$ of object o' in Q, then $L_o^{\vdash}$ is not a result turning point. This is because even though the road distance of o to q is less than d_ε between $[L_o^{\vdash}, L^{\dashv}_{o'\backslash o}]$, o is still dominated by o' before q reaches $L^{\dashv}_{o'\backslash o}$. Otherwise (i.e., no $L^{\dashv}_{o'\backslash o}$ exists), o becomes a d_ε-*SP* after $L_o^{\vdash}$, and thus $L_o^{\vdash}$ is a result turning point.
- When the landmark $L_o^{\dashv}$ of object o is considered: if object o is currently in the d_ε-*SP* set, its road distance $d_{q,o}$ to q must be less than or equal to d_ε. Once q passes through $L_o^{\dashv}$, $d_{q,o}$ is greater than d_ε, and o needs to be removed from the d_ε-*SP* set. That is, $L_o^{\dashv}$ is a result turning point.
- When the landmark $L^{\vdash}_{o\backslash o'}$ of object o is considered: as query object q reaches the landmark $L^{\vdash}_{o\backslash o'}$, object o will dominate object o' in terms of the static and dynamic attributes. If both objects o and o' are currently in the d_ε-*SP* set, then after $L^{\vdash}_{o\backslash o'}$, o' is no longer the d_ε-*SP* because it is dominated by o. Therefore, $L^{\vdash}_{o\backslash o'}$ is a result turning point.
- When the landmark $L^{\dashv}_{o\backslash o'}$ of object o is considered: the landmark $L^{\dashv}_{o\backslash o'}$ indicates that object o cannot dominate o' when q reaches $L^{\dashv}_{o\backslash o'}$. If object o is currently in the d_ε-*SP* set and no landmark $L^{\dashv}_{o''\backslash o'}$ of object o'' can be found in Q, object o' is promoted to a d_ε-*SP* at $L^{\dashv}_{o\backslash o'}$ and a result turning point occurs.

Algorithm 2. The result turning point determination phase of the Cd_ε-*SQ* algorithm.

Input: A distance d_ε, a set *GSP*, an edge e connecting two nodes n_s and n_e
Output: A set of tuples in form of $\langle [n_i,n_j], SP^\varepsilon \rangle$, where SP^ε is the d_ε-*SP* set between $[n_i,n_j]$
create an empty queue Q;
/* determining the landmarks of each object in *GSP* */
foreach *object* $o \in GSP$ **do**
  determine o's landmarks $L_o^{\vdash}$ and $L_o^{\dashv}$;
  if $L_o^{\vdash}$ *is on* e **then** insert $L_o^{\vdash}$ into Q;
  if $L_o^{\dashv}$ *is on* e **then** insert $L_o^{\dashv}$ into Q;
  foreach *object* $o' \in GSP - \{o\} \cap o.S_{DO}$ **do**
    determine o's landmarks $L^{\vdash}_{o\backslash o'}$ or $L^{\dashv}_{o\backslash o'}$;
    if $L^{\vdash}_{o\backslash o'}$ *is on* e *and* $L_o^{\vdash}$ *is closer to* n_s *than* $L^{\vdash}_{o\backslash o'}$ **then** insert $L^{\vdash}_{o\backslash o'}$ into Q;
    if $L^{\dashv}_{o\backslash o'}$ *is on* e *and* $L^{\dashv}_{o\backslash o'}$ *is closer to* n_s *than* $L_o^{\dashv}$ **then** insert $L^{\dashv}_{o\backslash o'}$ into Q;
sort the landmarks in Q in ascending order of their distances to node n_s;
set n_i and SP^ε to n_s and the d_ε-*SP* for n_s, respectively;
/* determining the result turning points with corresponding d_ε-*SP* */
while Q *is not empty* **do**
  de − queue $o.L$;
  switch $o.L$ **do**
    case $L_o^{\vdash}$
      if *there is no* $L^{\dashv}_{o'\backslash o} \in Q$ **then** return $\langle [n_i,L_o^{\vdash}], SP^\varepsilon \rangle$, $n_i = L_o^{\vdash}$, and add o into SP^ε;
    case $L_o^{\dashv}$
      if $o \in SP^\varepsilon$ **then** return $\langle [n_i,L_o^{\dashv}], SP^\varepsilon \rangle$, $n_i = L_o^{\dashv}$, and remove o from SP^ε;
    case $L^{\vdash}_{o\backslash o'}$
      if $o \in SP^\varepsilon$ *and* $o' \in SP^\varepsilon$ **then** return $\langle [n_i,L^{\vdash}_{o\backslash o'}], SP^\varepsilon \rangle$, $n_i = L^{\vdash}_{o\backslash o'}$, and remove o' from SP^ε;
    otherwise
      /* $o.L$ corresponds to $L^{\dashv}_{o\backslash o'}$
      if $o \in SP^\varepsilon$ *and there is no* $L^{\dashv}_{o''\backslash o'} \in Q$ **then** return $\langle [n_i,L^{\dashv}_{o\backslash o'}], SP^\varepsilon \rangle$, $n_i = L^{\dashv}_{o\backslash o'}$, and add o' into SP^ε;
return $\langle [n_i,n_e], SP^\varepsilon \rangle$;

Consider again the example in Fig. 7, where the initial d_ε-*SP* set is $\{o_1,o_2\}$ (i.e., the $SP^\varepsilon_{n_1}$ of node n_1) and seven landmarks from $L^{\dashv}_{o_1}$ to $L^{\dashv}_{o_3}$ are possible result turning points. (1) Considering $L^{\dashv}_{o_1}$: object o_1 is removed from the d_ε-*SP* set as its road distance $d_{q,o_1} > d_\varepsilon$ after $L^{\dashv}_{o_1}$. Thus, $L^{\dashv}_{o_1}$ is a result turning point and the d_ε-*SP* set changes from $\{o_1,o_2\}$ to $\{o_2\}$. (2) Considering $L^{\vdash}_{o_3}$: even though the road distance $d_{q,o_3} < d_\varepsilon$, object o_3 cannot be the d_ε-*SP* because there exists a landmark $L^{\dashv}_{o_2\backslash o_3}$ behind $L^{\vdash}_{o_3}$ (meaning that o_3 is dominated by o_2 between $[L^{\vdash}_{o_3}, L^{\dashv}_{o_2\backslash o_3}]$). (3) Considering $L^{\dashv}_{o_2\backslash o_3}$: as no object can dominate o_3, it becomes the d_ε-*SP*, and $L^{\dashv}_{o_2\backslash o_3}$ is a result turning point where the d_ε-*SP* set is $\{o_2,o_3\}$. (4) Considering $L^{\dashv}_{o_2}$: object o_2 is removed from the d_ε-*SP* set since its distance $d_{q,o_2} > d_\varepsilon$. Therefore, $L^{\dashv}_{o_2}$ is a result turning point. (5) Considering $L^{\vdash}_{o_4}$: object o_4 is added into the d_ε-*SP* set and thus $L^{\vdash}_{o_4}$ is a result turning point. (6) Considering $L^{\vdash}_{o_4\backslash o_3}$: object o_3 will be dominated by o_4 in terms of the static and dynamic attributes. As such, o_3 needs to be removed from the d_ε-*SP* set

(changing from $\{o_3,o_4\}$ to $\{o_4\}$). (7) Considering $L^{\perp}_{o_3}$: because object o_3 is currently not in the d_ε-*SP* set, its landmark $L^{\perp}_{o_3}$ will not affect the d_ε-*SP* set (i.e., not a result turning point). The complete d_ε-*SP* results from nodes n_1 to n_2 are shown in Fig. 7(b).

4.2. Cd_ε-SQ$^+$ algorithm

The Cd_ε-SQ^+ algorithm is proposed to further improve the query performance of the Cd_ε-*SQ* algorithm by applying a different pruning strategy in the global *SP* determination phase. Recall that the Cd_ε-*SQ* algorithm utilizes the Euclidean distance between objects as a lower bound of the road distance in order to prune the non-qualifying objects. Although the Euclidean distance can be easily computed with respect to the objects' coordinates, it cannot properly approximate the road distance. This will result in: (1) more grid cells needing to be accessed (e.g., cell $g(2,0)$ in Fig. 6(a) which slightly intersects the circle with radius d_ε); and (2) more candidates whose Euclidean distances are less than or equal to d_ε but with road distances that are greater than d_ε (e.g., object o_6).

Algorithm 3. The global *SP* determination phase of the Cd_ε-SQ^+ algorithm.

Input: A grid index, a distance d_ε, an edge e connecting two nodes n_s and n_e
Output: The *GSP* for edge e
/* determining $SP^\varepsilon_{n_s}$ for node n_s */
create an empty queue Q;
insert cell $g(i,j)$ enclosing node n_s into Q;
while Q *is not empty* **do**
  de − queue $g(i,j)$ and access its T_{edge}, T_{node}, and T_{obj};
  foreach *edge* e *stored in* T_{edge} **do**
    if *node* n_i *has been visited* **then**
      foreach *object* o *on edge* e **do**
        compute the road distance $d_{n_s,o}$ of o to n_s;
        if $d_{n_s,o} \le d_\varepsilon$ **then** insert o into a candidate set $S_{candidate}$;
      compute the road distance d_{n_s,n_j} from nodes n_s to n_j;
      if $d_{n_s,n_j} < d_\varepsilon$ **then** insert cell $g(m,n)$ enclosing n_j into Q;
foreach *object* $o \in S_{candidate}$ **do**
  if $o \notin S_{DO}$ *of the other objects in* $S_{candidate}$ **then** insert o into $SP^\varepsilon_{n_s}$;
  elseif $o \in o'$. S_{DO} *and* $d_{n_s,o} < d_{n_s,o'}$ **then** insert o *into* $SP^\varepsilon_{n_s}$;
determine $SP^\varepsilon_{n_e}$ for node n_e in a similar way;
retrieve the set S_e of objects on edge e;
set *GSP* to $SP^\varepsilon_{n_s} \bigcup SP^\varepsilon_{n_e} \bigcup S_e$;

To remedy this problem, the Cd_ε-SQ^+ algorithm conducts the Breadth First Search on the road network (starting from nodes n_s and n_e), and examines objects in the order they are visited. Suppose that the d_ε-*SP* set of node n_s (i.e., $SP^\varepsilon_{n_s}$) needs to be determined. During the course of determining the $SP^\varepsilon_{n_s}$, queue Q is used to keep the accessed cells of the grid index, and initially Q only contains the cell enclosing node n_s. In each iteration, an element of Q (i.e., a cell $g(i,j)$) will be retrieved, so as to obtain its corresponding T_{edge}, T_{node}, and T_{obj} information (referring to Fig. 4). For each edge e stored in T_{edge}, once its node n_i has been visited, the road distance d_{n_s,n_j} from n_s to its node n_j will be computed. If $d_{n_s,n_j} < d_\varepsilon$, the road distances of those objects located on edge e must be less than d_ε. Thus, they are kept for dominance test and the cell $g(m,n)$ enclosing node n_j is inserted into Q. Otherwise, the cell $g(m,n)$ enclosing n_j need not be accessed (i.e., $g(m,n)$ is not inserted into Q). This is because all the road distances of the unvisited objects stored in T_{obj} of $g(m,n)$ are greater than d_ε. In this case, only the objects on edge e whose road distances to n_s are less than or equal to d_ε will be kept. The above process proceeds until Q is empty. Finally, $SP^\varepsilon_{n_s}$ is determined by performing the dominance tests among the kept objects. Also, $SP^\varepsilon_{n_e}$ of node n_e can be obtained in a similar way. The pseudo code of the detailed steps for the global *SP* determination phase of the Cd_ε-SQ^+ algorithm is given in Algorithm 3.

Consider again the example in Fig. 6. If the $SP^\varepsilon_{n_1}$ of node n_1 is obtained by accessing the cells that intersect the circle with radius d_ε centered at n_1, five cells $g(1,0)$, $g(1,1)$, $g(1,2)$, $g(2,0)$, and $g(2,2)$ need to be accessed. However, only the cell $g(1,1)$ enclosing node n_1 will be accessed, as the determination of $SP^\varepsilon_{n_1}$ is based on the Breadth First Search (the road distances from nodes n_1 to n_2, n_6, and n_9 are all greater than d_ε). By applying the Breadth First Search on the grid index, the cells that do not need to be accessed and the road distance computations of those distant objects can be effectively reduced so as to improve the performance of the global *SP* determination phase. As the result turning point determination phase of the Cd_ε-SQ^+ algorithm is similar to that of the Cd_ε-*SQ* algorithm, it is omitted from presentation.

4.3. Time complexity analysis of Cd_ε-SQ approaches

Since the Cd_ε-*SQ* approaches (i.e., the Cd_ε-*SQ* and Cd_ε-SQ^+ algorithms) include the global *SP* determination phase and the result turning point determination phase, we consider the two phases individually:

- In the global *SP* determination phase, $SP^\varepsilon_{n_s}$ and $SP^\varepsilon_{n_e}$ for nodes n_s and n_e, respectively, need to be determined so as to obtain the *GSP*. Consider first the time complexity of determining $SP^\varepsilon_{n_s}$. Let $|o_{n_s}|$ be the number of objects whose road distances to n_s have to be computed and $|o'_{n_s}|$ be the number of objects whose road distances to n_s are less than or equal to d_ε, and $|o'_{n_s}| \le |o_{n_s}|$. Note that for the Cd_ε-SQ^+ algorithm, the size of $|o_{n_s}|$ is equal to that of $|o'_{n_s}|$. Then, the time complexity of determining $SP^\varepsilon_{n_s}$ is $O(|o_{n_s}| \cdot C_{dist} + |o'_{n_s}|^2)$, where C_{dist} refers to the cost of computing road distance by accessing the grid index. Similarly, the time complexity of determining $SP^\varepsilon_{n_e}$ is $O(|o_{n_e}| \cdot C_{dist} + |o'_{n_e}|^2)$. Thus, we have the time complexity of the global *SP* determination phase which is $O((|o_{n_s}| + |o_{n_e}|) \cdot C_{dist} + |o'_{n_s}|^2 + |o'_{n_e}|^2)$.
- The result turning point determination phase consists of two steps (referring to Algorithm 2). The first step is to determine the landmarks of each object in *GSP* and maintain a queue Q to store these landmarks. The second step is to find the result turning points with corresponding d_ε-*SP*. Let $|GSP|$ and $|Q|$ be the sizes of *GSP* and queue Q, respectively. The time complexity of the first step is $O(|GSP|^2 + |Q| \cdot \log|Q|)$, where $O(|GSP|)$ refers to the cost for determining the landmarks and

$O(|Q| \cdot \log|Q|)$ represents the cost for sorting the landmarks. The complexity of second step is $O(|Q|)$. Therefore, the total time complexity of the result turning point determination phase is $O(|GSP|^2 + |Q| \cdot (\log|Q| + 1))$.

5. Processing techniques for *Cknn-SQ*

A *Cknn-SQ* can be utilized to find the k nearest *SP* (i.e., the *knn-SP*) for each point p of query path P_q. The *Cknn-SQ algorithm* and the *Cknn-SQ*$^+$ *algorithm* are proposed to efficiently process such a *Cknn-SQ*, both of which divide P_q into a set of edges and then consider these edges sequentially to find the *knn-SP*. Similar to the Cd_c-*SQ* approaches presented in Section 4, the *Cknn-SQ* and *Cknn-SQ*$^+$ *algorithms* consist of two phases: (1) the global *SP* determination phase that accesses the grid index to obtain the *GSP* for each edge e; and (2) the result turning point determination phase that determines the result turning points with their corresponding *knn-SP* by taking into consideration the objects in the *GSP* only.

5.1. Cknn-SQ algorithm

In the global *SP* determination phase of the *Cknn-SQ* algorithm, the *GSP* for edge e is determined based on the following corollary, which demonstrates that the *knn-SP* for any point on edge e belong to the union of the objects on e and the *knn-SP* for the two endpoints of edge e.

Corollary 5.1. *Given a set of knn-SP, denoted as* SP_p^k, *for a point p on an edge e connecting two nodes* n_s *and* n_e, *we have* $SP_p^k \subseteq SP_{n_s}^k \bigcup SP_{n_e}^k \bigcup S_e$, *where* $SP_{n_s}^k$ *and* $SP_{n_e}^k$ *are the knn-SP sets for nodes* n_s *and* n_e, *respectively, and* S_e *is the set of objects located on edge e.*

The main idea of finding the *knn-SP* set for node n is to determine a search region, denoted as SR_n, to ensure that at least k skyline points of node n can be found within this region. That is, an object could be the *knn-SP* for node n only if it is within SR_n. In order to determine the region SR_n, we first find the *SP* in terms of the static attributes (termed the *static SP*), and then compute the road distances of the k static *SP* whose Euclidean distances to node n are the smallest, so as to obtain the kth smallest road distance, denoted as d_{kth}. The region SR_n can be represented as a circle centered at node n with radius d_{kth}, which guarantees that k static *SP* must be inside SR_n. As such, each object outside SR_n is either farther from n than the k static *SP* or is dominated by one of them so that it cannot be the *knn-SP*. Using the above approach, the search regions SR_{n_s} and SR_{n_e} of nodes n_s and n_e can be, respectively, determined to prune the non-qualifying objects. Then, Algorithm 1 mentioned in Section 4 can be applied to find the *GSP* for edge e by changing the circle centered at n_s (and n_e) with radius d_c to the region SR_{n_s} (and SR_{n_e}).

Continue the example shown in Fig. 6, where objects o_1, o_2, and o_6 are the static *SP*. Assume that a *Cknn-SQ* is issued to find the *knn-SP* for edge e_1, and that $k=2$. During the course of finding the *knn-SP* set for node n_1 (i.e., $SP_{n_1}^k$), the road distances d_{n_1,o_1} and d_{n_1,o_2} of objects o_1 and o_2 will be computed because they have the smallest Euclidean distances to n_1. Then, the search region SR_{n_1} is represented as a circle centered at n_1 with a radius d_{n_1,o_1}. As SR_{n_1} intersects cells $g(1,1)$, $g(1,2)$, and $g(2,2)$, the three cells are accessed to retrieve the information of objects o_1, o_2, o_3, and o_6. Referring to Fig. 6(b), $SP_{n_1}^k = \{o_2, o_1\}$. Similarly, $SP_{n_2}^k$ can be obtained as $\{o_4, o_2\}$ (note that objects o_3 and o_5 are dominated by o_4 in terms of static and dynamic attributes). Finally, as $S_{e_1} = \{o_3\}$ (i.e., o_3 is on edge e_1), the *GSP* for edge e_1 is set to $\{o_2, o_1, o_4, o_3\}$.

In the result turning point determination phase, three types of landmarks are set up on edge e by using the dominance relationships between objects in the *GSP* and each is individually checked to find the result turning points. The first type of landmark is defined as $L_{o>o'}$, which refers to the point at which the road distance of object o to $L_{o>o'}$ is equal to that of object o'. Note that objects o and o' cannot dominate each other in terms of the static attributes (that is, $o \notin o'.S_{DO}$ and $o' \notin o.S_{DO}$). Once query object q passes $L_{o>o'}$, object o' is closer to q than o. For instance, in Fig. 8(a), object o_3 starts to get closer to q than o_1 after the landmark $L_{o_1>o_3}$ (i.e., the point where the road distance to n_1 is equal to 25). The determination of the landmark $L_{o>o'}$ is divided into two cases: (1) object $o \in SP_{n_s}^k$ but $o \notin S_e$ and (2) object $o \in S_e$. In the first (or second) case, the landmark $L_{o>o'}$ is set to the point at which the distance to n_s is equal to $(d_{n_s,o'} - d_{n_s,o})/2$ (or $(d_{n_s,o'} + d_{n_s,o})/2$). The number of landmark $L_{o>o'}$ on edge e is at most $|SP_{n_s}^k - S_e| \cdot (|GSP| - |SP_{n_s}^k - S_e|) + |S_e| \cdot |SP_{n_e}^k - S_e|$ (refer to Section 4.1). The other two types of landmarks are the same as $L_{o\backslash o'}^{\vdash}$ and $L_{o\backslash o'}^{\dashv}$ defined in Section 4.1, and hence the total number of landmarks is at most $3 \cdot (|SP_{n_s}^k - S_e| \cdot (|GSP| - |SP_{n_s}^k - S_e|) + |S_e| \cdot |SP_{n_e}^k - S_e|)$.

The procedure of the result turning point determination phase is detailed in Algorithm 4. During the process, the objects in the *GSP* are sorted in ascending order of their road distances to query object q, and the first k objects not dominated by others are kept in set SP^k (i.e., the *knn-SP* set). Initially, q is located on node n_s, and thus, the SP^k is set to $SP_{n_s}^k$. Then, the landmarks $L_{o>o'}$, $L_{o\backslash o'}^{\vdash}$, and $L_{o\backslash o'}^{\dashv}$ of each object o in the *GSP* are determined and inserted into queue Q, in which the landmarks closer to n_s will be considered first:

- When the landmark $L_{o>o'}$ of object o is considered: it implies that objects o and o' cannot dominate each other in terms of the static attributes, and that o' is closer to q than o after $L_{o>o'}$. That is, the order of objects o and o' in the *GSP* exchanges once q reaches $L_{o>o'}$. If $o \notin SP^k$, then SP^k remains unchanged no matter whether $o' \in SP^k$ or not (i.e., $L_{o>o'}$ is not a result turning point). In the case where $o \in SP^k$ and $o' \in SP^k$, as o and o' cannot dominate each other, at $L_{o>o'}$ the change of their order will not affect SP^k. The only case where $L_{o>o'}$ could be a result turning point is when $o \in SP^k$ but $o' \notin SP^k$. Due to $o' \notin SP^k$, object o' is either: (1) dominated by another object $o'' \in SP^k$; or (2) a skyline point but not a *knn-SP*. If the first condition holds, at $L_{o>o'}$ object o' is still dominated by o'' and it

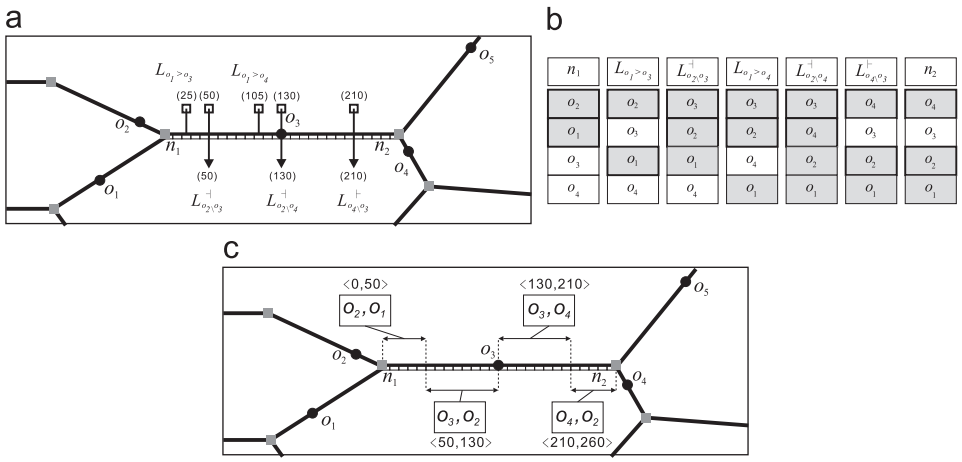


Fig. 8. The result turning point determination phase of the *Cknn-SQ* algorithm. (a) Landmarks of *GSP*. (b) *knn-SP* for each landmark. (c) Determining the result turning points.

cannot be inserted into SP^k (that is, SP^k remains the same). Otherwise, the second condition holds. Because object o' replaces o as the kth nearest SP at $L_{o>o'}$, object o (or o') needs to be removed from (or added into) SP^k. Hence, $L_{o>o'}$ is a result turning point.

- When the landmark $L^{\vdash}_{o\backslash o'}$ of object o is considered: it indicates that object o dominates o' after $L^{\vdash}_{o\backslash o'}$. Also, the order of objects o and o' in the GSP exchanges after $L^{\vdash}_{o\backslash o'}$. If object $o' \notin SP^k$, then $L^{\vdash}_{o\backslash o'}$ must not be a result turning point regardless of whether o is in SP^k. Otherwise (i.e., $o' \in SP^k$), o' will be removed from SP^k because o begins to dominate o' after $L^{\vdash}_{o\backslash o'}$, and hence $L^{\vdash}_{o\backslash o'}$ becomes a result turning point. In such a case, if $o \notin SP^k$ (the reason is that o is not one of the k nearest SP), then o replaces o' as the kth nearest SP and is inserted into SP^k. If $o \in SP^k$, the number of objects in SP^k will be less than k because o' is removed from SP^k, and thus, the kth SP in the GSP needs to be inserted into SP^k.
- When the landmark $L^{\dashv}_{o\backslash o'}$ of object o is considered: it implies that after $L^{\dashv}_{o\backslash o'}$, object o' is no longer dominated by o (i.e., $o' \notin SP^k$ before $L^{\dashv}_{o\backslash o'}$), and that o' precedes o in GSP. In the case where $o \notin SP^k$ and $o' \notin SP^k$, there still exists object $o'' \in SP^k$ dominating o after $L^{\dashv}_{o\backslash o'}$, therefore, SP^k remains unchanged. In the case where $o \in SP^k$ but $o' \notin SP^k$, o' has a chance to be a *knn-SP* (i.e., $L^{\dashv}_{o\backslash o'}$ could be a result turning point). If no object in SP^k can dominate o', then o' becomes a new *knn-SP* and the kth nearest SP in SP^k needs to be removed.

Algorithm 4. The result turning point determination phase of the *Cknn-SQ* algorithm.

Input: A number k, a set GSP, an edge e connecting two nodes n_s and n_e
Output: A set of tuples in form of $\langle [n_i,n_j], SP^k \rangle$, where SP^k is the *knn-SP* set between $[n_i,n_j]$
sort the objects in GSP in ascending order of their distances to node n_s and set SP^k to $SP^k_{n_s}$;
create an empty queue Q;
/* determining the landmarks of each object in GSP */
foreach *object* $o \in GSP$ **do**
  foreach *object* $o' \in GSP-\{o\}$ **do**
    if $o' \notin o.S_{DO}$ **then**
      determine o's landmark $L_{o>o'}$;
      if $L_{o>o'}$ *is on* e **then** insert $L_{o>o'}$ into Q;
    else
      determine o's landmarks $L^{\vdash}_{o\backslash o'}$ or $L^{\dashv}_{o\backslash o'}$;
      if $L^{\vdash}_{o\backslash o'}$ *is on* e **then** insert $L^{\vdash}_{o\backslash o'}$ into Q;
      if $L^{\dashv}_{o\backslash o'}$ *is on* e **then** insert $L^{\dashv}_{o\backslash o'}$ into Q;
sort the landmarks in Q in ascending order of their distances to node n_s;
set n_i to n_s;
/* determining the result turning points with corresponding *knn-SP* */
while Q *is not empty* **do**
  de − queue $o.L$;
  switch $o.L$ **do**
    case $L_{o>o'}$
      if $o \in SP^k$ *and* $o' \notin SP^k$ **then**
        if *there is no object* $o'' \in SP^k$ *dominating* o' **then** return $\langle [n_i,L_{o>o'}], SP^k \rangle$, $n_i = L_{o>o'}$,
        remove o from SP^k,and insert o' into SP^k;
    case $L^{\vdash}_{o\backslash o'}$
      if $o' \in SP^k$ **then**
        return $\langle [n_i,L_{o>o'}], SP^k \rangle$, $n_i = L_{o>o'}$, remove o' from SP^k;
        if $o \notin SP^k$ **then** insert o *into* SP^k;
        else insert the k-th SP in GSP into SP^k;
    otherwise
      /* $o.L$ corresponds to $L^{\dashv}_{o\backslash o'}$ */
      if $o \in SP^k$ *and* $o' \notin SP^k$ **then**
        if *there is no object* $o'' \in SP^k$ *dominating* o' **then**
          return $\langle [n_i,L^{\dashv}_{o\backslash o'}], SP^k \rangle$, $n_i = L^{\dashv}_{o\backslash o'}$,
        remove the k-th SP from SP^k, and insert o' into SP^k;
return $\langle [n_i,n_e], SP^k \rangle$;

Fig. 8 continues the previous example of *Cknn-SQ* in Fig. 6, where the objects in the GSP are sorted in ascending order of their distances to node n_1 and are represented as $\{o_2,o_1,o_3,o_4\}$. The initial result of SP^k (i.e., $SP^k_{n_1}$) is set to $\{o_2,o_1\}$. As shown in Fig. 8(a), five landmarks are considered sequentially, from landmarks $L_{o_1>o_3}$ to $L_{o_3>o_4}$, so as to find the result turning points with corresponding *knn-SP*. (1)

Considering $L_{o_1 > o_3}$: the order of objects o_1 and o_3 in the *GSP* exchanges, and $GSP = \{o_2, o_3, o_1, o_4\}$. Because object $o_3 \in o_2.S_{DO}$ (referring to Fig. 6(b)), o_3 is still dominated by o_2 at $L_{o_1 > o_3}$, and hence SP^k remains unchanged. (2) Considering $L^{\dashv}_{o_2 \backslash o_3}$: object o_2 cannot dominate o_3 and the *GSP* changes to $\{o_3, o_2, o_1, o_4\}$. As no object dominates o_3, it becomes a *knn-SP* and the second *knn-SP* o_1 in SP^k is removed. That is, $L^{\dashv}_{o_2 \backslash o_3}$ is a result turning point and SP^k is updated to $\{o_3, o_2\}$. (3) Considering $L_{o_1 > o_4}$: the *GSP* changes to $\{o_3, o_2, o_4, o_1\}$. Because both objects o_1 and o_4 are not in SP^k, the *knn-SP* remains the same. (4) Considering $L^{\dashv}_{o_2 \backslash o_4}$: the *GSP* changes to $\{o_3, o_4, o_2, o_1\}$. As object o_3 cannot dominate o_4, SP^k is updated from $\{o_3, o_2\}$ to $\{o_3, o_4\}$. (5) Considering $L^{\vdash}_{o_4 \backslash o_3}$: object o_4 begins to dominate o_3 and the *GSP* changes to $\{o_4, o_3, o_2, o_1\}$. After removing o_3 from SP^k, the second *SP* in the *GSP* (i.e., object o_2) needs to be added into SP^k, and thus SP^k is updated to $\{o_4, o_2\}$. Finally, the *knn-SP* result for each result turning point is obtained and shown in Fig. 8(c).

5.2. Cknn-SQ$^+$ algorithm

Similar to the Cd_ε-SQ$^+$ algorithm, the goal of the *Cknn-SQ*$^+$ algorithm is to further improve the performance of the global *SP* determination phase by performing the Breadth First Search on the road network. Algorithm 5 illustrates how to determine the *GSP* for edge e based on the Breadth First Search. To determine the *knn-SP* set $SP^k_{n_s}$ of node n_s, we use queue Q (and list $\mathcal{L}$) to keep the accessed cells of the grid index (and the objects that have been visited so far and could be the *SP*). Note that the objects in $\mathcal{L}$ are sorted in ascending order of their distances to node n_s, and that initially the cell enclosing node n_s is inserted into Q. In each iteration, an element of Q (i.e., a cell $g(i,j)$) will be retrieved so as to obtain its corresponding T_{edge}, T_{node}, and T_{obj} information. For each edge e stored in T_{edge}, when its node n_i has been visited, the road distances $d_{n_s,o}$ from n_s to each object o on edge e will be first computed and then o is inserted into $\mathcal{L}$. If all of the objects in front of o in $\mathcal{L}$ cannot dominate o, then o needs to be kept in $\mathcal{L}$, because it not only could be a *SP* but could also dominate the objects behind it (that is, the objects behind o will be removed from $\mathcal{L}$ if they are dominated by o). Otherwise, o is removed from $\mathcal{L}$ because it cannot be a *SP*. When the number of objects in $\mathcal{L}$ is greater than or equal to k, distance d_{kth}, which refers to the road distance of the kth object to n_s, can be used to determine whether it is necessary to examine the rest of the grid cells not yet accessed. If the road distance d_{n_s,n_j} from n_s to node n_j is less than d_{kth}, the cell $g(m,n)$ enclosing node n_j needs to be accessed (i.e., $g(m,n)$ is inserted into Q). Otherwise, the cell $g(m,n)$ is not accessed. This is because all of the road distances of the unvisited objects stored in T_{obj} of $g(m,n)$ must be greater than those of the first k objects in $\mathcal{L}$, and thus, those objects cannot be the *knn-SP*. The grid cells are repeatedly accessed until Q is empty, and finally the first k objects in $\mathcal{L}$ are the *knn-SP* of n_s. Similarly, the *knn-SP* set $SP^k_{n_e}$ of node n_e can be obtained using the method mentioned above. After retrieving the set S_e of the objects on edge e, the *GSP* is set to $SP^k_{n_s} \bigcup SP^k_{n_e} \bigcup S_e$.

5.3. Time complexity analysis of Cknn-SQ approaches

Similar to the analysis of the Cd_ε-SQ approaches, we analyze the time complexities of the global *SP* determination phase and the result turning point determination phase separately:

- In order to find the *GSP*, the *knn-SP* sets for nodes n_s and n_e (i.e., $SP^k_{n_s}$ and $SP^k_{n_e}$) should be determined. The time complexity of determining $SP^k_{n_s}$ is $O(|o_{n_s}| \cdot C_{dist} + |o'_{n_s}|^2)$. For the *Cknn-SQ* algorithm, $|o_{n_s}|$ (and $|o'_{n_s}|$) refers to the number of objects within the search region SR_{n_s} (and whose road distances to n_s are less than or equal to that of the kth nearest static *SP*), and C_{dist} is the cost of computing the object's road distance. As for the *Cknn-SQ*$^+$ algorithm, $|o_{n_s}|$ is the same as $|o'_{n_s}|$ and refers to the number of objects whose road distances are less than or equal to that of the kth nearest *knn-SP*. As the time complexity for determining $SP^k_{n_e}$ is $O(|o_{n_e}| \cdot C_{dist} + |o'_{n_e}|^2)$, the total complexity is $O((|o_{n_s}| + |o_{n_e}|) \cdot C_{dist} + |o'_{n_s}|^2 + |o'_{n_e}|^2)$.
- The result turning point determination phase consists of two steps (referring to Algorithm 4). The first step is to determine the object landmarks and maintain a queue Q to store these landmarks. Its time complexity is $O(|GSP|^2 + |Q| \cdot \log|Q|)$, where $|GSP|$ and $|Q|$ are the sizes of *GSP* and queue Q, respectively. The second is to find the result turning points and their *knn-SP*, and the complexity is $|Q|$. As such, the time complexity of the result turning point determination phase is estimated as $O(|GSP|^2 + |Q| \cdot (\log|Q| + 1))$.

Algorithm 5. The global *SP* determination phase of the *Cknn-SQ*$^+$ algorithm.

Input: A grid index, a number k, an edge e connecting two nodes n_s and n_e
Output: The *GSP* for edge e
/* determining $SP^k_{n_s}$ for node n_s */
create an empty queue Q and a sorted list $\mathcal{L}$, and set $d_{kth} = \infty$;
insert cell $g(i,j)$ enclosing node n_s into Q;
while Q *is not empty* **do**
  de-queue $g(i,j)$ and access itsT_{edge}, T_{node}, and T_{obj};
  foreach *edge e stored in* T_{edge} **do**
    if *node* n_i *has been visited* **then**
      foreach *object o on edge e* **do**
        compute the road distance $d_{n_s,o}$ of o to n_s and insert o into $\mathcal{L}$;
        if $o \notin o'.S_{DO}$ *of each object* o' *in front of it in* $\mathcal{L}$ **then**
          foreach *object* o'' *behind* o **do**
            if $o'' \in o.S_{DO}$ **then** remove o' from $\mathcal{L}$;
        else remove o from $\mathcal{L}$;
      if $|\mathcal{L}| < k$ **then** $d_{kth} = \infty$;
      else $d_{kth} = d_{n_s,o_{kth}}$ of the k-th object o_{kth} in $\mathcal{L}$;
      compute the road distance d_{n_s,n_j} from nodes n_s to n_j;
      if $d_{n_s,n_j} < d_{kth}$ **then** insert cell $g(m,n)$ enclosing n_j into Q;
insert the first k objects in $\mathcal{L}$ into $SP^k_{n_s}$;
determine $SP^k_{n_e}$ for node n_e in a similar way;
retrieve the set S_e of objects on edge e;
set *GSP* to $SP^k_{n_s} \bigcup SP^k_{n_e} \bigcup S_e$;

6. Performance evaluation

We conduct four sets of experiments for the Cd_ε-SQ and the *Cknn-SQ* approaches in this section. The first one is evaluated to study the effect of the number of grid cells on the performance of processing Cd_ε-SQ and *Cknn-SQ*. We decide an appropriate cell number from these experiments and use it to build the grid index for query processing in the rest of the experiments. The second one investigates the performance of the Cd_ε-SQ and Cd_ε-SQ$^+$ algorithms by measuring the CPU time and the number of accessed cells for processing a Cd_ε-SQ. The third set is conducted to demonstrate the efficiency of the *Cknn-SQ* and *Cknn-SQ*$^+$ algorithms in terms of CPU time and number of accessed cells. Finally, the last set of experiments studies how well our approaches work for larger road networks.

6.1. Experimental settings

All experiments are performed on a PC with AMD Athlon 64 X2 5200 CPU and 2 GB RAM. The algorithms are implemented in C++. We use three road maps in our simulation and generate objects (varying from 0.1 K to 200 K) using the generator proposed in [38]. The first road map is the Oldenburg (a city in Germany) consisting of about 6000 nodes and 7000 edges, the second is the San Joaquin County with about 18,200 nodes and 23,800 edges, and the third is the California Road Network which contains 21,048 nodes and 21,693 edges [39–43]. Each generated object has several static attributes (ranging from 2 to 6) whose values are normalized in the range $[0, 1]$. In the experimental space, we also generate 30 query paths and each of them consists of multiple edges (ranging from 1 to 16). The start edge e of each query path is randomly chosen, and once e has been considered, the next edge is randomly selected from the edges connecting e. For each query path, we perform a Cd_ε-SQ and a *Cknn-SQ* (i.e., find the d_ε-SP and the *knn-SP* for each point within the query path), where the default values of d_ε and k are set to 1% of the entire space and 10, respectively. The performance is measured by the average CPU time and the average number of accessed cells in performing workloads of the 30 queries. Table 1 summarizes the parameters under investigation, along with their default values and ranges.

We compare the proposed Cd_ε-SQ approaches with a naive approach, in which the road distances of all objects are first computed and then the BBS algorithm [8] is applied to finding the skyline points with respect to the computed distances and static attributes. Only the skyline points within the distance range d_ε can be the query result. Similarly, we compare the performance of the *Cknn-SQ* approaches against a naive approach, where the skyline points are first determined using the BBS algorithm and then sorted according to their road distances so as to retrieve the k nearest skyline points.

Table 1
System parameters.

Parameter	Default	Range
Number of objects (K)	100	0.1, 1, 10, 100, 200
Number of static attributes	4	2, 3, 4, 5, 6
Number of grid cells	128^2	32^2, 64^2, 128^2, 256^2, 512^2
Length of query path	4	1, 2, 4, 8, 16
d_ε (%)	1	0.1, 0.5, 1, 2, 3
k	10	1, 5, 10, 15, 20

6.2. Effect of number of grid cells

Fig. 9 studies the effect of the number of grid cells on the performance of processing Cd_ε-SQ and *Cknn-SQ*. In the experiments, we vary grid cells from 32×32 to 512×512 and measures the CPU cost and the number of accessed cells for the proposed algorithms. As shown in Fig. 9(a) and (b), the CPU costs for the Cd_ε-SQ and the *Cknn-SQ* approaches decrease with the increasing number of grid cells. This is mainly because each cell would store fewer object as the number of grid cells increases (i.e., the cell size is smaller), so that the time required for finding the *GSP* in the global *SP* determination phase can be reduced. On the other hand, a larger number of grid cells would result in the fact that more cells are accessed to find the *GSP*, which is illustrated in Fig. 9(c) and (d). Because the number of cells decreases the CPU time but increases the number of accessed cells, we need to decide an appropriate cell number used to build the grid index for query processing. As we can see from Fig. 9, when the cell number is greater than 128×128, the improvement of CPU cost for all algorithms becomes insignificant, while the number of cells need to be accessed increases sharply. The experimental result reveals that 128×128 is a better choice than the others. Hence, we will use it as the default cell number in all the rest experiments.

6.3. Performance of Cd_ε-SQ and Cd_ε-SQ$^+$ algorithms

In this subsection, we compare the Cd_ε-SQ and Cd_ε-SQ$^+$ algorithms with a naive approach in terms of the CPU time and the number of accessed cells. Four experiments are conducted to investigate the effects of five important factors on the performance of processing Cd_ε-SQ. These important factors are *the number of objects*, *the number of static attributes*, *the length of query path*, *the size of query range* (i.e., the value of d_ε), and *the distribution of objects inside each grid cell*.

Fig. 10 illustrates the performance of the Cd_ε-SQ approaches as a function of the number of objects (ranging from 0.1 K to 200 K). Note that hereafter all figures use a logarithmic scale for the y-axis. As shown in Fig. 10(a), the Cd_ε-SQ$^+$ algorithm significantly outperforms the naive approach in the CPU time, even for a smaller number of objects (e.g., 0.1 K) which corresponds to the number of gas stations or hotels. This is mainly because for the naive approach the road distances of all objects have to be computed which incurs high computation cost as the object number increases. Similarly, the performance gap between the Cd_ε-SQ$^+$ algorithm and Cd_ε-SQ algorithm increases with the increasing number of objects. The reason is that in the Cd_ε-SQ algorithm some objects whose road distances are greater than d_ε have to be considered because their Euclidean distances are less than or equal to d_ε, whereas in the Cd_ε-SQ$^+$ algorithm the

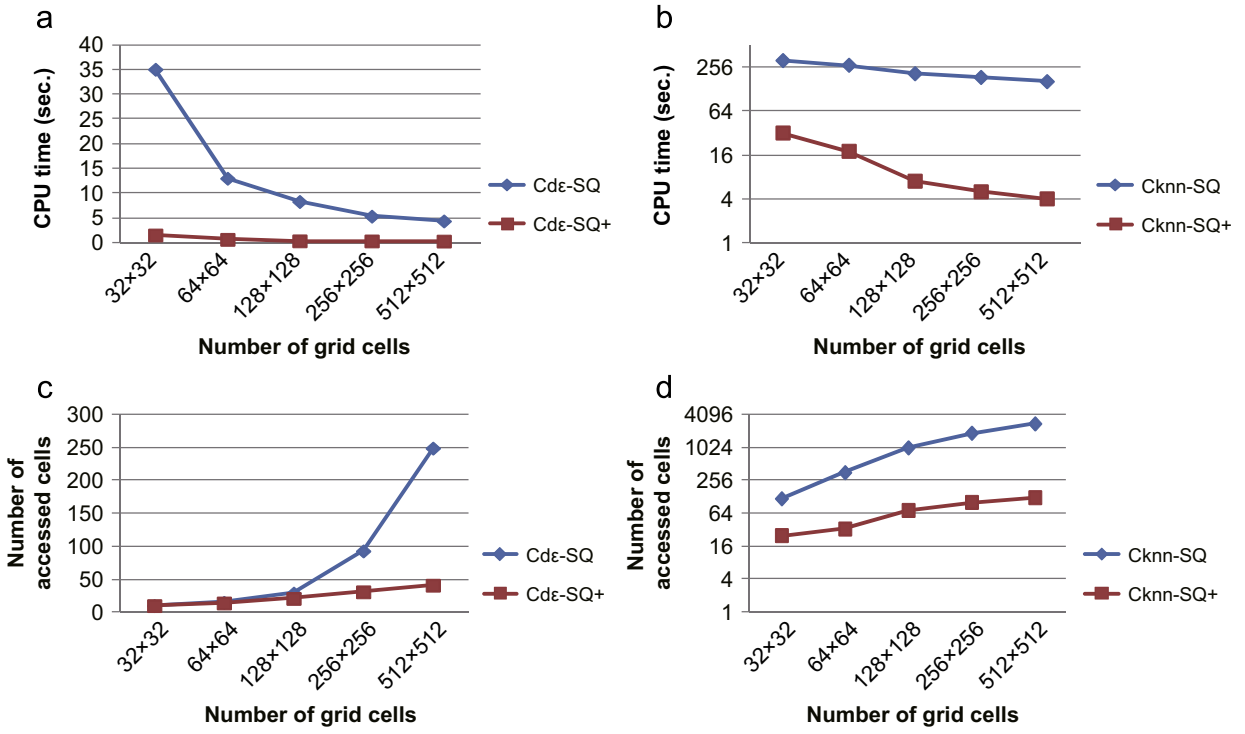


Fig. 9. Effect of the number of grid cells. (a) CPU cost for the Cd_ε-SQ. (b) CPU cost for the *Cknn-SQ*. (c) Number of accessed cells for the Cd_ε-SQ. (d) Number of accessed cells for the *Cknn-SQ*.

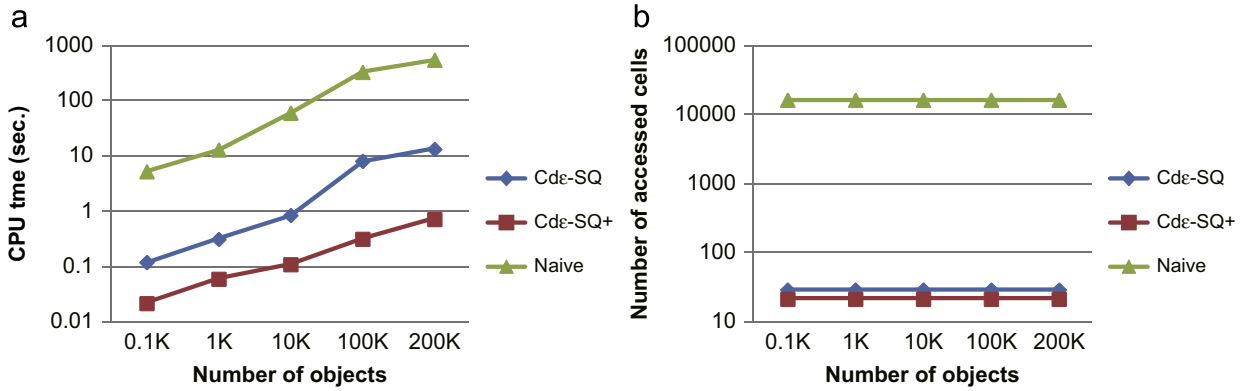


Fig. 10. Cd_ε-SQ performance vs. number of objects. (a) CPU cost for the Cd_ε-SQ. (b) Number of accessed cells for the Cd_ε-SQ.

unnecessary distance computation of these objects can be avoided by applying the Breadth First Search on the road network. The experimental result shown in Fig. 10(b) illustrates that the number of accessed cells for the Cd_ε-SQ algorithm, the Cd_ε-SQ^+ algorithm, and the naive approach remains constant regardless of the number of objects. This is because no matter what the object quantity is, only the cells storing objects within distance range d_ε would be accessed in the Cd_ε-SQ and Cd_ε-SQ^+ algorithms (for the naive approach all cells need to be accessed). Hence, the number of accessed cells in the proposed algorithms is dominated by d_ε and insensitive to the amount of objects.

Fig. 11 studies the impact of the number of static attributes on the performance of processing Cd_ε-SQ. Fig. 11(a) measures the CPU time for the Cd_ε-SQ algorithm, the Cd_ε-SQ^+ algorithm, and the naive approach by varying the number of static attributes from 2 to 6. When this number increases, the CPU overhead for all algorithms increases slightly because more attributes would result in more skyline points so that more dominance tests are performed. As for the number of accessed cells (shown in Fig. 11(b)), for the Cd_ε-SQ and Cd_ε-SQ^+ algorithms it is constant and under 30 in spite of the number of attributes (for the naive approach the number of accessed cells is equal to 128×128). The reason is the

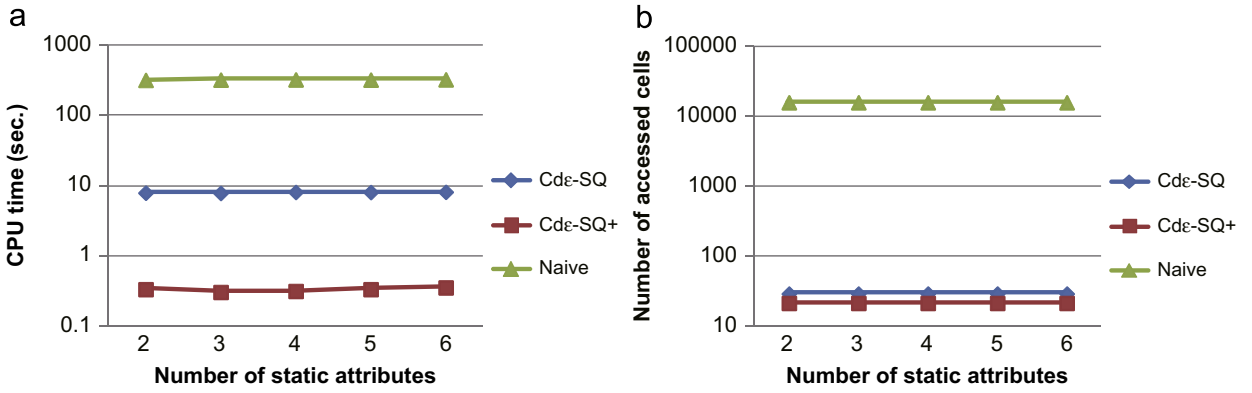


Fig. 11. Cd_ε-SQ performance vs. number of static attributes. (a) CPU cost for the Cd_ε-SQ. (b) Number of accessed cells for the Cd_ε-SQ.

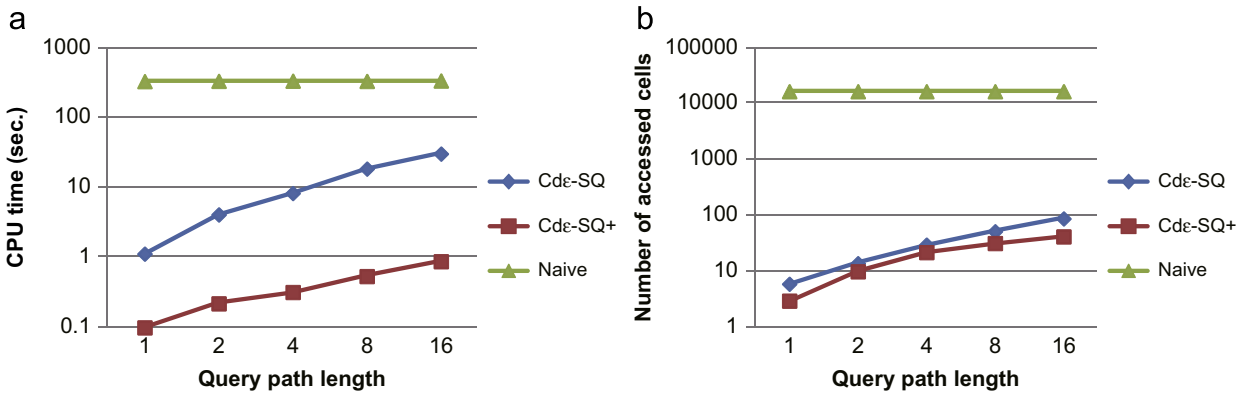


Fig. 12. Cd_ε-SQ performance vs. query path length. (a) CPU cost for the Cd_ε-SQ. (b) Number of accessed cells for the Cd_ε-SQ.

same as that for Fig. 10(b). The experimental results show that the Cd_ε-SQ^+ algorithm outperforms its competitors significantly in all cases, which confirms again that applying the Breadth First Search to prune the non-qualifying objects can greatly improve the performance of processing Cd_ε-SQ.

Fig. 12 evaluates the CPU cost and the number of accessed cells for the Cd_ε-SQ algorithm, the Cd_ε-SQ^+ algorithm, and the naive approach under various query path lengths (varying from 1 to 16). As we can see, the naive approach gives the worst performance in all cases because no pruning strategy is adopted to reduce the distance computations of objects. As for the Cd_ε-SQ and Cd_ε-SQ^+ algorithms, when the query path length is equal to 1 (that is, only the d_ε-SP for one edge need to be determined), the performance of the Cd_ε-SQ^+ algorithm is as good as that of the Cd_ε-SQ algorithm. However, when the query length is greater than 1, the Cd_ε-SQ^+ algorithm outperforms the Cd_ε-SQ algorithm by a factor of 20 to 40 in terms of the CPU cost. In addition, the Cd_ε-SQ^+ algorithm reduces the number of accessed cells by almost a factor of 2 compared to the Cd_ε-SQ algorithm. The improvement is due to the fact that the Cd_ε-SQ^+ algorithm retrieves only the objects whose road distances to one of the nodes within the query path are less than or equal to d_ε. However, in the Cd_ε-SQ algorithm the objects would be retrieved once their Euclidean distances are less than or equal to d_ε.

In Fig. 13, we measure the CPU time and the number of accessed cells under different values of d_ε. For the Cd_ε-SQ^+ algorithm, as the non-qualifying objects (i.e., the objects outside the distance range d_ε) can be precisely pruned by applying the Breadth First Search, both the required CPU time and the number of accessed cells increase slightly with the increasing d_ε. For the Cd_ε-SQ algorithm, a smaller d_ε is favorable because less computation of road distances of the qualifying objects is performed. On the other hand, the performance (i.e., both the CPU time and the accessed cell number) of the Cd_ε-SQ algorithm significantly degrades as the value of d_ε increases. For the naive approach, the effect of increasing d_ε is not significant because computing the road distances of all objects dominates the overall performance.

Finally in this subsection, we study how the distribution of objects inside a grid cell affects the performance of the Cd_ε-SQ algorithm, the Cd_ε-SQ^+ algorithm, and the naive approach, by considering three types of object distributions. These object distributions are the *uniform distribution*, the *Gaussian distribution* (with variance 0.05 and a mean being the coordinate of the center of grid cell), and the *Zipf distribution* (with skew coefficient 0.8). As shown in Fig. 14(a), the CPU time of all approaches

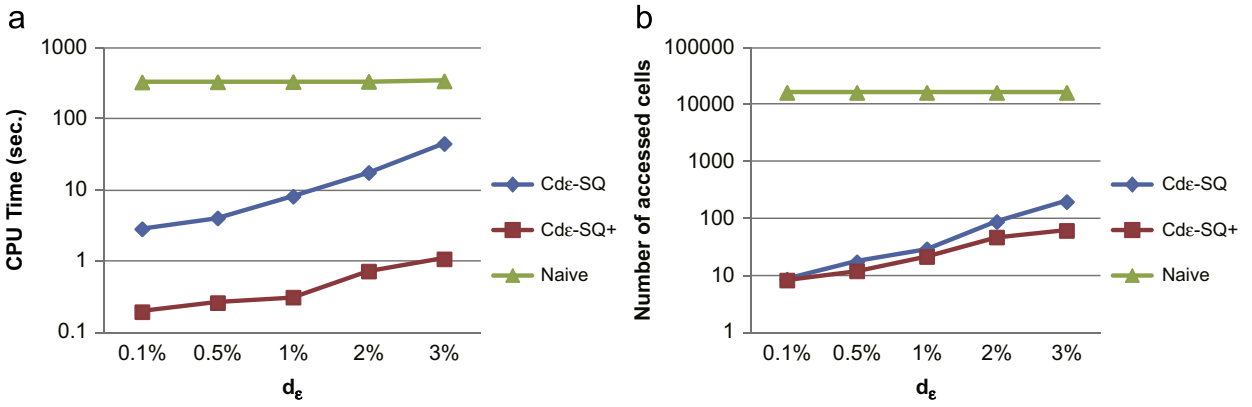


Fig. 13. Cd_ε-*SQ* performance vs. d_ε. (a) CPU cost for the Cd_ε-*SQ*. (b) Number of accessed cells for the Cd_ε-*SQ*.

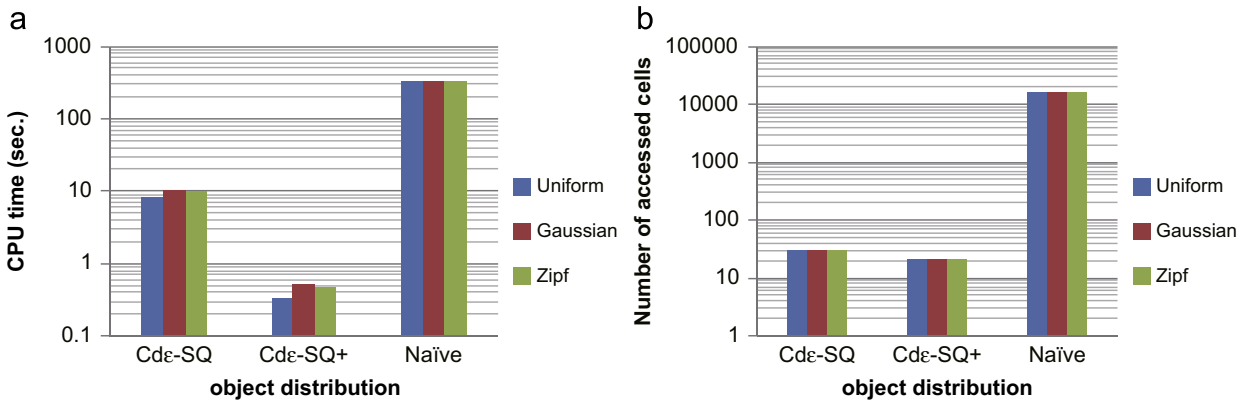


Fig. 14. Cd_ε-*SQ* performance vs. object distribution. (a) CPU cost for the Cd_ε-*SQ*. (b) Number of accessed cells for the Cd_ε-*SQ*.

for the uniform distribution is slightly lower than that for the Gaussian and Zipf distributions. This is because for the Gaussian and Zipf distributions, the objects inside a grid cell are closer to each other (comparing to the uniform distribution). Therefore, more landmarks need to be examined in the result turning point determination phase (no matter which approach is adopted), so that it takes more CPU time to find the result turning points. From the experimental result, we also find that the CPU performance is insensitive to the object distribution. As for the number of accessed cells, Fig. 14(b) indicates that all approaches have the constant number of accessed cells regardless of object distribution, because for our Cd_ε-*SQ* and Cd_ε-SQ^+ algorithms the number of accessed cells is dominated by d_ε, and for the naive approach all grid cells have to be accessed to compute the road distances of objects.

6.4. Performance of Cknn-SQ and Cknn-SQ^+ algorithms

To exploit the efficiency of the proposed algorithms in processing a *Cknn-SQ*, we compare them in terms of the CPU cost and the number of accessed cells. We study the impacts of different factors on the performance of processing the *Cknn-SQ*. These factors are *the number of objects*, *the number of static attributes*, *the query path length*, *the number of nearest neighbors* (i.e., *k*), and *the object distribution within a grid cell*.

Fig. 15 demonstrates the effect of various numbers of objects on the performance of the *Cknn-SQ* algorithm, the *Cknn-*SQ^+ algorithm, and the naive approach. Fig. 15(a) and (b) investigates the CPU cost and the number of accessed cells, respectively, by varying object number from 0.1 K to 200 K. As shown in Fig. 15(a), the CPU overhead for all algorithms increases with the increase of number of objects because more distance computations of objects are required for finding the *GSP*. Nevertheless, the *Cknn-*SQ^+ algorithm significantly outperforms the *Cknn-SQ* algorithm and the naive approach in all cases. An interesting observation from Fig. 15(b) is that a larger number of objects results in a less number of cells that need to be accessed in processing the *Cknn-SQ*. This is mainly because when the road network is sparse (e.g., object number is 0.1 K or 1 K), more cells have to be accessed so as to find the *k*th nearest *SP*, whereas a dense network (e.g., number of objects is 200 K) may lead to early termination of finding the *k*th nearest *SP* so that less cells are accessed.

In Fig. 16, we study how the number of static attributes affects the CPU time and the number of accessed

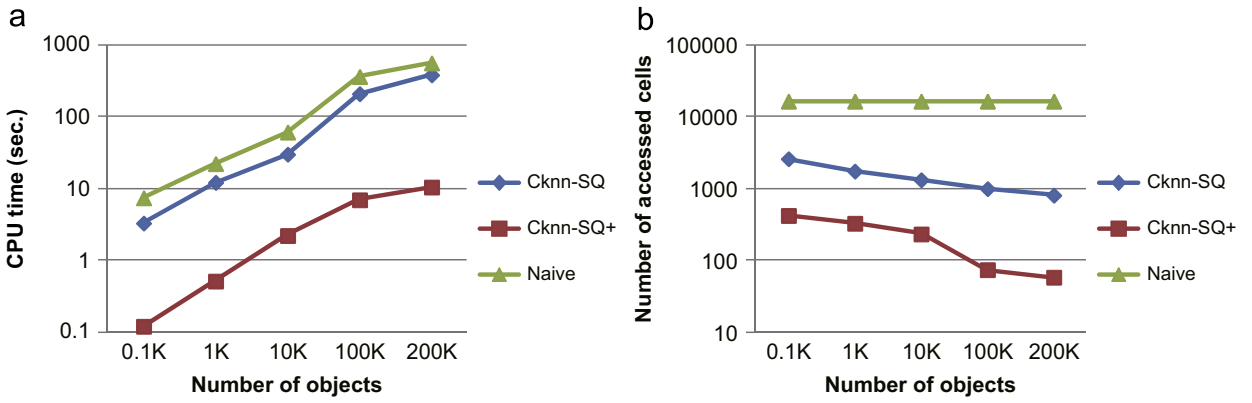


Fig. 15. *Cknn-SQ* performance vs. number of objects. (a) CPU cost for the *Cknn-SQ*. (b) Number of accessed cells for the *Cknn-SQ*.

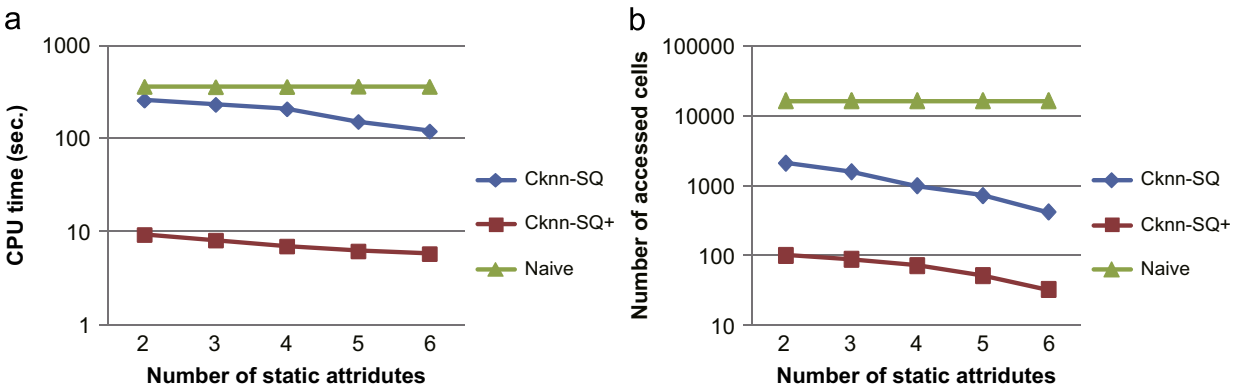


Fig. 16. *Cknn-SQ* performance vs. number of static attributes. (a) CPU cost for the *Cknn-SQ*. (b) Number of accessed cells for the *Cknn-SQ*.

cells for the *Cknn-SQ* algorithm, the *Cknn-SQ*$^+$ algorithm, and the naive approach. Different from the previous experimental results, the curves for both the *Cknn-SQ* algorithm and the *Cknn-SQ*$^+$ algorithm shown in Fig. 16(a) and (b) have the decreasing trends. That is, the CPU cost and the accessed cells can be reduced as the number of static attributes increases. The reason for this improvement is that for a small number of static attributes, most of the objects are dominated by fewer objects and thus more cells are accessed to retrieve objects and more distance computations of these objects are required in finding the kth nearest *SP*. Conversely, for a large number of attributes, the kth nearest *SP* could be determined early by only accessing the cells closer to the query path because they may already contain the k *SP*. As for the naive approach, it is insensitive to the number of static attributes so that the CPU time and the number of accessed cells remain about constant under various static attribute numbers.

Fig. 17 shows the CPU cost and the number of accessed cells for the proposed algorithms and the naive approach under various lengths of query path. Similar to the experiments in Fig. 12, we vary the query length from 1 to 16 to investigate its effect on the performance of processing the *Cknn-SQ*. As we can see, the *Cknn-SQ*$^+$ algorithm has a significantly better performance on both the CPU cost and the number of accessed cells compared to its competitors. In terms of the CPU overhead, the *Cknn-SQ*$^+$ algorithm outperforms the *Cknn-SQ* algorithm (and the naive approach) by a factor of up 25 (and 35) in all cases. In addition, the *Cknn-SQ*$^+$ algorithm reduces the number of accessed cells by almost two (and three) orders of magnitude compared to the *Cknn-SQ* algorithm (and the naive approach).

This experiment evaluates the effect of the number of nearest neighbors (i.e., k) on the CPU time and the accessed cells of the *Cknn-SQ* algorithm, the *Cknn-SQ*$^+$ algorithm, and the naive approach. As shown in Fig. 18(a), when k increases, the CPU overhead for the *Cknn-SQ* and *Cknn-SQ*$^+$ algorithms grows. The reason is that as k becomes greater, the number of qualifying objects inevitably increases so that more distance computations of these objects and more dominance tests between them are required. As for the effect of k on the number of accessed cells shown in Fig. 18(b), the curve for both algorithms has the increasing trend. This is because as k increases, the kth nearest *SP* becomes farther from the query path, and hence more cells would be accessed. Nevertheless, benefiting from the Breadth First Search, the *Cknn-SQ*$^+$ algorithm is more insensitive to k compared to the *Cknn-SQ* algorithm. The experimental result shows that the naive approach yields the worst performance no matter in the CPU cost or the number of accessed cells.

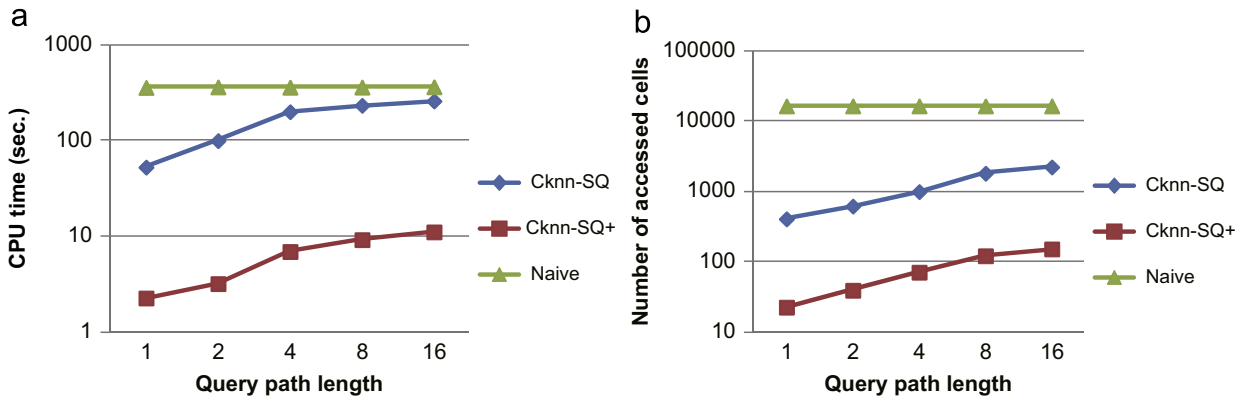


Fig. 17. *Cknn-SQ* performance vs. query path length. (a) CPU cost for the *Cknn-SQ*. (b) Number of accessed cells for the *Cknn-SQ*.

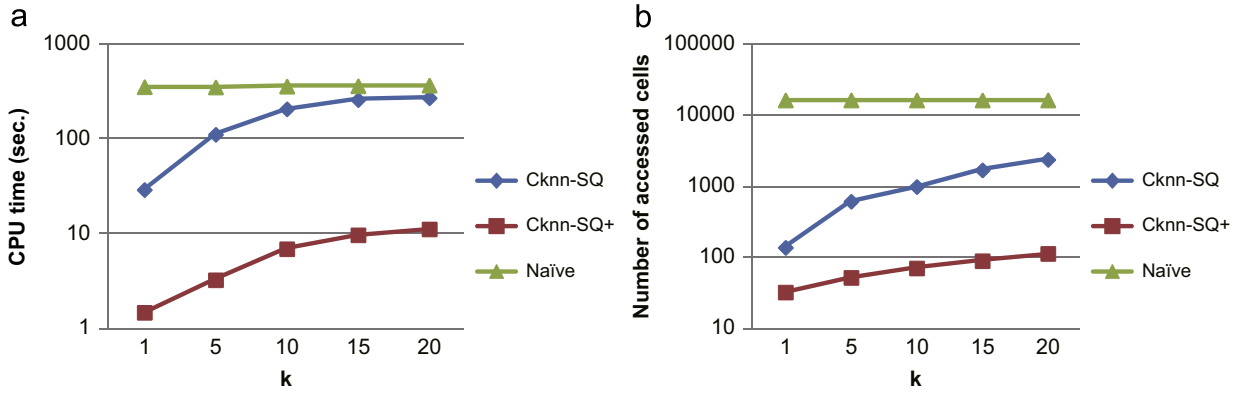


Fig. 18. *Cknn-SQ* performance vs. *k*. (a) CPU cost for the *Cknn-SQ*. (b) Number of accessed cells for the *Cknn-SQ*.

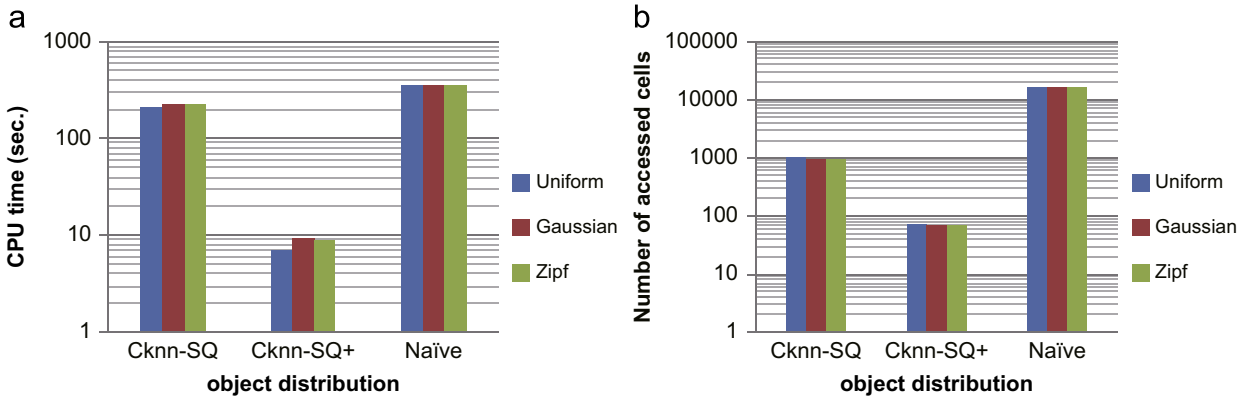


Fig. 19. *Cknn-SQ* performance vs. object distribution. (a) CPU cost for the *Cknn-SQ*. (b) Number of accessed cells for the *Cknn-SQ*.

The last experiment measures the CPU overhead and the number of accessed cells for the *Cknn-SQ* algorithm, the *Cknn-SQ*$^+$ algorithm, and the naive approach, using different object distributions (including the uniform distribution, the Gaussian distribution, and the Zipf distribution). As shown in Fig. 19(a), each *Cknn-SQ* approach achieves a slightly better CPU performance for the uniform distribution than for the Gaussian and Zipf distributions. The reason is that for the uniform distribution fewer object landmarks are considered in the result turning point determination phase. However, Fig. 19(b) shows that when the objects inside each grid cell follow the uniform distribution, the *Cknn-SQ* algorithm and the *Cknn-SQ*$^+$ algorithm require a little more accessed cells (comparing to the Gaussian and Zipf distributions). This is because the Gaussian and Zipf distributions could result in a higher chance of determining the *k*th nearest *SP* early in the global *SP* determination phase.

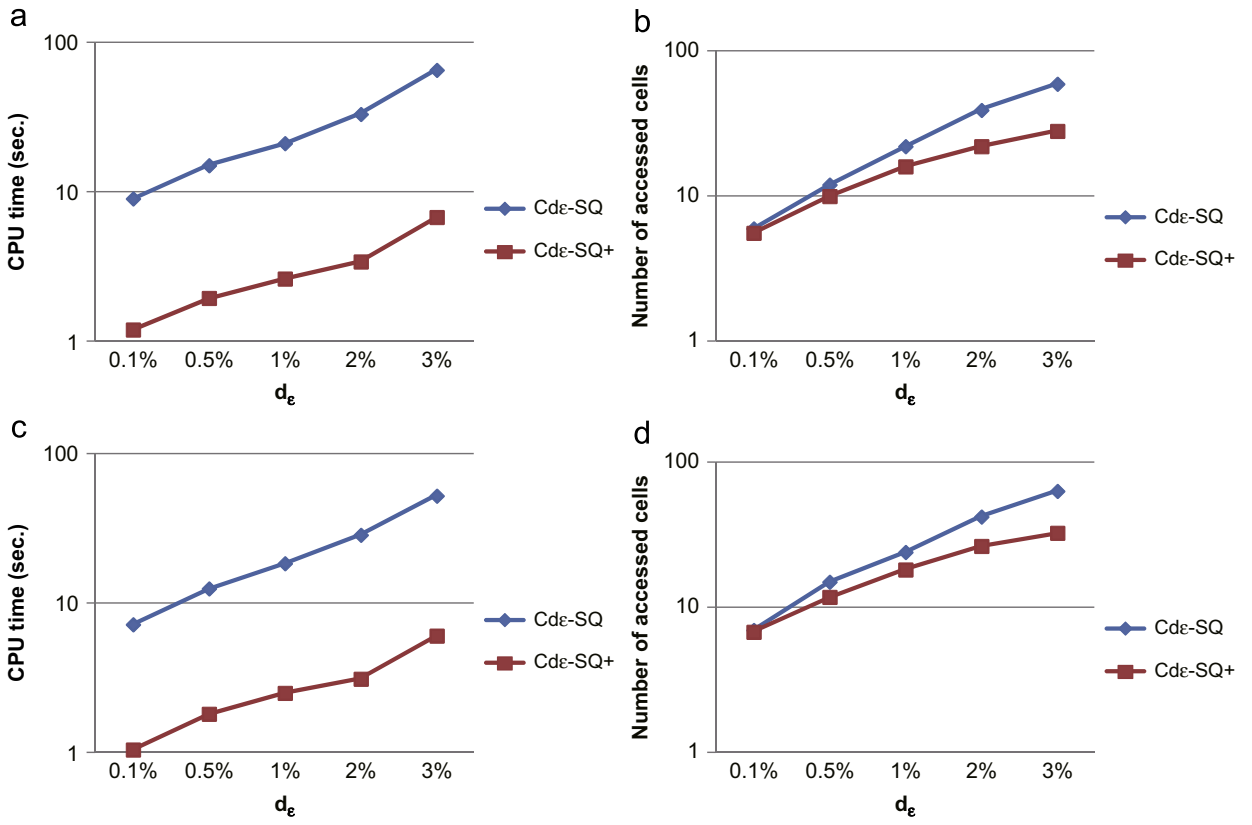


Fig. 20. Cd_ε-SQ performance for the larger road networks. (a) CPU cost for San Joaquin County. (b) Number of accessed cells for San Joaquin County. (c) CPU cost for California Road Network. (d) Number of accessed cells for California Road Network.

6.5. Performance for larger road networks

In this subsection, we conduct two sets of experiments to study how well the proposed Cd_ε-SQ and $Cknn$-SQ approaches work for two larger road networks, the San Joaquin County with about 18,200 nodes and 23,800 edges and the California Road Network consisting of 21,048 nodes and 21,693 edges. Fig. 20(a) and (b) measures the CPU time and the number of accessed cells for our Cd_ε-SQ approaches using the San Joaquin County, respectively, as a function of d_ε (varying from 0.1% to 3%). Fig. 20(c) and (d) studies the performance of our Cd_ε-SQ approaches for the California Road Network under various values of d_ε. Compared to the experimental result in Fig. 13 (which shows the performance for a small road network, Oldenburg), the CPU overhead for both the Cd_ε-SQ algorithm and the Cd_ε-SQ^+ algorithm increases, while the number of accessed cells decreases. The reason for the higher CPU time is that the road connectivity of the larger road networks is more complicated so that more time is required for computing the road distances of objects within the distance range d_ε. Nevertheless, the CPU overhead still increases within a slow rate with a fairly acceptable range. On the other hand, the decrease of accessed cells for the larger road networks is mainly because each grid cell stores more information of objects and hence the Cd_ε-SQ can be terminated by accessing fewer cells closer to the query path.

The last set of experiments investigates the performance of the proposed $Cknn$-SQ approaches for the San Joaquin County and the California Road Network by measuring the CPU time and the number of accessed cells under different values of k. As shown in Fig. 21(a) and (c), the CPU time of the $Cknn$-SQ and $Cknn$-SQ^+ algorithms for the San Joaquin County and the California Road Network is higher than that for the Oldenburg (shown in Fig. 18(a)). Conversely, as we can see from Fig. 21(b) and (d), the number of accessed cells for the San Joaquin County and the California Road Network is much less than that in the Oldenburg case (shown in Fig. 18(b)). Similar to the reasons mentioned in the above set of experiments (i.e., Fig. 20), the higher CPU time (the less number of accessed cells) is owing to the complicated road connectivity (more object information storing in each grid cell) of the larger road networks. From the experimental results, we know that the proposed Cd_ε-SQ and $Cknn$-SQ approaches are also suitable for a road network with large number of nodes and edges.

7. Conclusions and future work

This paper focused on processing two novel and important distance-based skyline queries in road networks. The first was the continuous d_ε-skyline query (Cd_ε-SQ) and the second was the continuous k nearest neighbor-skyline query ($Cknn$-SQ). A grid index was designed to manage the information of data objects

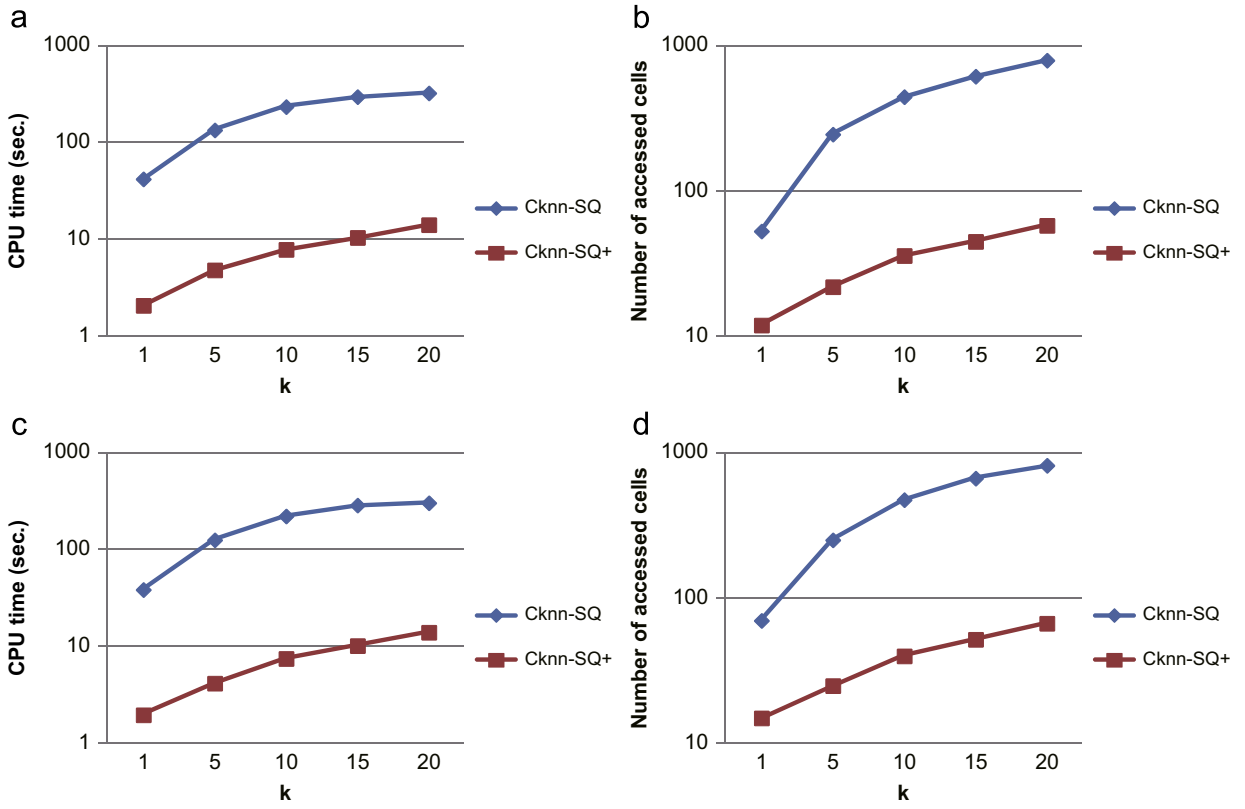


Fig. 21. *Cknn-SQ* performance for the larger road networks. (a) CPU cost for San Joaquin County. (b) Number of accessed cells for San Joaquin County. (c) CPU cost for California Road Network. (d) Number of accessed cells for California Road Network.

located in a road network consisting of edges and nodes, and made use of the grid index to access only a small proportion of data objects while computing the Cd_ε-*SQ* and *Cknn-SQ* results. The Cd_ε-*SQ* algorithm and the Cd_ε-SQ^+ algorithm were developed, both of which consist of a global *SP* determination phase and a result turning point determination phase, to efficiently process the Cd_ε-*SQ*. Also, the *Cknn-SQ* algorithm and the *Cknn-*SQ^+ algorithm were proposed to find the *Cknn-SQ* result. Comprehensive experiments demonstrated the effectiveness and the efficiency of the proposed approaches.

There are several interesting avenues for the future extensions of this work. One important avenue is to extend the Cd_ε-*SQ* and *Cknn-SQ* approaches to be suitable for dynamic environment where the objects move with time. Another extension is to address the issue of efficiently handling the updates of objects (such as updates of static and dynamic attributes). A further extension is how to make proper use of data structures so as to enhance the query performance.

Acknowledgment

This work was supported by National Science Council of Taiwan (ROC) under Grant nos. NSC100-2221-E-244-018 and NSC100-2221-E-006-249-MY3.

References

[1] S. Borzsonyi, D. Kossmann, K. Stocker, The skyline operator, in: 17th International Conference on Data Engineering, 2001, pp. 421–430.
[2] J. Chomicki, P. Godfrey, J. Gryz, D. Liang, Skyline with presorting, in: Nineteenth International Conference on Data Engineering, 2003, pp. 717–719.
[3] B. Jiang, J. Pei, Online interval skyline queries on time series, in: International Conference on Data Engineering, March 2009, pp. 1036–1047.
[4] W. Jin, A.K.H. Tung, M. Ester, J. Han, On efficient processing of subspace skyline queries on high dimensional data, in: International Conference on Scientific and Statistical Database Management, 2007.
[5] D. Kossmann, F. Ramsak, S. Rost, Shooting stars in the sky: an online algorithm for skyline queries, in: International Conference on VLDB, 2002.
[6] X. Lin, Y. Yuan, Q. Zhang, Y. Zhang, Selecting stars: the k most representative skyline operator, in: International Conference on Data Engineering, 2007, pp. 86–95.
[7] J. Lee, G.W. You, S.W. Hwang, Personalized top-k skyline queries in high-dimensional space, Information System 34 (1) (2009).
[8] D. Papadias, Y. Tao, G. Fu, B. Seeger, An optimal and progressive algorithm for skyline queries, in: ACM SIGMOD, 2003, pp. 467–478.
[9] N. Sarkas, G. Das, N. Koudas, A.K.H. Tung, Categorical skylines for streaming data, in: ACM SIGMOD, 2008, pp. 239–250.
[10] K.-L. Tan, P.-K. Eng, B.C. Ooi, Efficient progressive skyline computation, in: International Conference on Very Large Data Bases, 2001, pp. 301–310.
[11] Y. Tao, X. Xiao, J. Pei, Subsky: efficient computation of skylines in subspaces, in: International Conference on Data Engineering, 2006.
[12] K. Deng, X. Zhou, H.T. Shen, Multi-source skyline query processing in road networks, in: International Conference on Data Engineering, 2007, pp. 796–805.

[13] K. Mouratidis, Y. Lin, M.L. Yiu, Preference queries in large multi-cost transportation networks, in: International Conference on Data Engineering, 2010.
[14] M. Schubert, M. Renz, H. Kriegel, Route skyline queries: a multi-preference path planning approach, in: International Conference on Data Engineering, 2010.
[15] L. Zou, L. Chen, M.T. Ozsu, D. Zhao, Dynamic skyline queries in large graphs, in: International Conference on Database Systems for Advanced Applications, 2010.
[16] S. Zhang, N. Mamoulis, D.W. Cheung, Scalable skyline computation using object-based space partitioning, in: International Conference on ACM SIGMOD, 2009.
[17] E.W. Dijkstra, A note on two problems in connection with graphs, Numeriche Mathematik 1 (1959) 269–271.
[18] R. Kung, E. Hanson, Y. Ioannidis, T. Sellis, L. Shapiro, M. Stonebraker, Heuristic search in data base system, in: International Workshop on Expert Database Systems, 1986, pp. 537–548.
[19] H.T. Kung, F. Luccio, F.P. Preparata, On finding the maxima of a set of vectors, Journal of the ACM 22 (4) (1975) 469–476.
[20] J.L. Bentley, H.T. Kung, M. Schkolnick, C.D. Thompson, On the average number of maxima in a set of vectors and applications, Journal of the ACM 25 (4) (1978) 536–543.
[21] I. Bartolini, P. Ciaccia, M. Patella, Efficient sort-based skyline evaluation, ACM TODS 33 (4) (2008) 1–45.
[22] X. Lin, Y. Yuan, W. Wangnicta, H. Lu, Stabbing the sky: efficient skyline computation over sliding windows, in: International Conference on Data Engineering, 2005.
[23] M. Morse, J. Patel, W. Grosky, Efficient continuous skyline computation, in: Information System, 2007, pp. 3411–3437.
[24] Y. Tao, D. Papadias, Maintaining sliding window skylines on data streams, IEEE Transactions on Knowledge and Data Engineering 18 (2) (2006) 377–391.
[25] W. Zhang, X. Lin, Y. Zhang, W. Wang, J.X. Yu, Probabilistic skyline operator over sliding windows, in: IEEE International Conference on Data Engineering, Washington, DC, USA, 2009, pp. 1060–1071.
[26] Z. Huang, B.O.H. Lu, A. Tung, Continuous skyline queries for moving objects, IEEE Transactions on Knowledge and Data Engineering 18 (December) (2006) 1645–1658.
[27] M. Sharifzadeh, C. Shahabi, The spatial skyline queries, in: International Conference on Very Large Data Bases, 2006, pp. 751–762.
[28] H.-P. Kriegel, M. Renz, M. Schubert, Route skyline queries: a multi-preference path planning approach, in: International Conference on Data Engineering, 2010.
[29] X. Huang, C.S. Jensen, In-route skyline querying for location-based services, in: International Workshop on Web and Wireless Geographical Information Systems, 2004, pp. 120–135.
[30] S.M. Jang, J.S. Yoo, Processing continuous skyline queries in road networks, in: International Symposium on Computer Science and its Applications, 2008.
[31] C.S. Jensen, J. Kolářvr, T.B. Pedersen, I. Timko, Nearest neighbor queries in road networks, in: International Symposium on Advances in Geographic Information Systems, New York, NY, USA, 2003, pp. 1–8.
[32] D. Papadias, J. Zhang, N. Mamoulis, Y. Tao, Query processing in spatial network databases, in: Proceedings of the 29th International Conference on Very Large Data Bases, Berlin, Germany, 9–12 September 2003, pp. 802–813.
[33] M. Kolahdouzan, C. Shahabi, Voronoi-based k nearest neighbor search for spatial network databases, in: Proceedings of the Thirtieth International Conference on Very Large Data Bases, 2004, pp. 840–851.
[34] M.R. Kolahdouzan, C. Shahabi, Continuous k-nearest neighbor queries in spatial network databases, in: Proceedings of the Second International Workshop on Spatio-Temporal Database Management, 2004, pp. 33–40.
[35] H.-J. Cho, C.-W. Chung, An efficient and scalable approach to cnn queries in a road network, in: International Conference on Very Large Data Bases, 2005, pp. 865–876.
[36] K. Mouratidis, M.L. Yiu, D. Papadias, N. Mamoulis, Continuous nearest neighbor monitoring in road networks, in: Proceedings of the 32nd International Conference on Very Large Data Bases, 2006, pp. 43–54.
[37] Y.-K. Huang, Z.-W. Chen, C. Lee, Continuous k-nearest neighbor query over moving objects in road networks, in: International Conference on APWeb-WAIM, 2–4 April 2009.
[38] T. Brinkhoff, A framework for generating network-based moving objects, GeoInformatica 6 (2) (2002) 153–180.
[39] Z. Chen, H.T. Shen, X. Zhou, J.X. Yu, Monitoring path nearest neighbor in road networks, in: ACM SIGMOD, 2009, pp. 591–602.
[40] H. Gonzalez, J. Han, X. Li, M. Myslinska, J.P. Sondag, Adaptive fastest path computation on a road network: a traffic mining approach, in: International Conference on Very Large Data Bases, 2007.
[41] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, S. Teng, On trip planning queries in spatial databases, in: SSTD, 2005.
[42] M.L. Yiu, N. Mamoulis, D. Papadias, Aggregate nearest neighbor queries in road networks, IEEE Transactions on Knowledge and Data Engineering 17 (6) (2005).
[43] H. Zhu, J. Su, O.H. Ibarra, Trajectory queries and octagons in moving object databases, in: International Conference on CIKM, McLean, VA, USA, 2002.