International Journal of Geo-Information

MDPI

Article

Within Skyline Query Processing in Dynamic Road Networks

Yuan-Ko Huang

Department of Maritime Information and Technology, National Kaohsiung Marine University, Kaohsiung City 80543, Taiwan; huangyk@webmail.nkmu.edu.tw

Academic Editor: Wolfgang Kainz
Received: 22 January 2017; Accepted: 27 April 2017; Published: 29 April 2017

Abstract: The continuous within skyline query is an important type of location-based query, which can provide useful skyline object information for the user. Previous studies on processing the continuous within skyline query focus exclusively on a static road network, where the object attributes and the conditions of roads remain unchanged. However, in real-world applications, object attributes and road conditions inevitably vary with time, which severely limits the applicability of previous studies in practice. Therefore, in this paper, we address the issue of efficiently processing the continuous within skyline query in dynamic road networks with time-varying information. We design three elaborate data structures, the object attribute dominating matrix (OADM), the road distance sorted list (RDSL) and the skyline object expansion tree (SOET), to maintain the information of objects and the road network. Combined with OADM, RDSL and SOET, we develop an efficient algorithm, namely the within skyline object updating algorithm, to provide real-time processing of the time-varying information. Finally, a thorough experimental evaluation is conducted to show the merits of the proposed approaches.

Keywords: continuous within skyline query; skyline objects; dynamic road networks; time-varying information; within skyline object updating algorithm

1. Introduction

With the fast advance of positioning techniques in mobile systems and the popularization of portable computers (e.g., laptops, 3G mobile phones and tablet PCs), spatio-temporal databases that aim at efficiently managing a large number of moving objects so as to support various types of location-based queries have attracted much attention in the database community [1–4]. Many applications, such as geographical information systems, traffic control systems and location-aware advertisements, can benefit from efficient processing of the location-based queries. The distance-based skyline query is an important type of location-based query that can provide useful information for preference-based data analysis and has a wide range of real applications [5–7]. Given a set of data objects S_o with m dimensional attributes and a query object q in a road network, the distance-based skyline query finds the objects in S_o that are not dominated by any other object, in terms of the m attributes and the road distance to q. More specifically, object p dominates another object p' if (1) its value in each attribute is as good or better than that of p' and is better in at least one attribute and (2) it is closer to q than p'. The objects not dominated by others are termed the skyline objects.

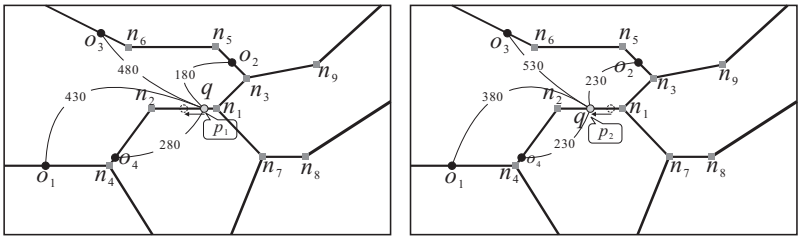
A novel distance-based skyline query, called the within skyline query, can be used to find the skyline objects within a given distance range, where the objects retrieved are termed the within skyline objects (WSOs). In the previous work [8], the continuous within skyline query is presented for continuous monitoring of WSOs in road networks. Given a path P_q, along which the query object q moves, a set of data objects S_o and a distance d, the continuous within skyline query finds a set of WSOs for each point p on P_q, such that the road distance from each WSO to point p is less than or

equal to d. The continuous within skyline query can be found in many fields and application domains. A real-world example is that of a traveler who is planning a trip, who may want to know which hotels are better to stay at en route. In this scenario, the traveler can issue the continuous within skyline query to find the hotels within the distance range d and with good quality (e.g., higher rank and lower price) during the trip.

Figure 1 illustrates an example of processing the continuous within skyline query, where objects o_1 to o_4 and the query object q are located in a road network, represented as a graph consisting of nodes and edges. Figure 1a shows the static attributes (i.e., "rank" and "price") of the four objects. In this example, the query object q moves from node n_1 to node n_2 (that is, the query path is $\overline{n_1 n_2}$). Assume that the continuous within skyline query is issued to find the WSOs whose road distances to q are within 300 (i.e., $d = 300$). When q is located at point p_1 (as shown in Figure 1b), only object o_2 is the WSO. Note that although o_1 is not dominated by o_2, it cannot be the WSO because its road distance to q is greater than d. When q moves to point p_2 (as shown in Figure 1c), the road 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, so that o_4 becomes a skyline object. As the road distances of o_2 and o_4 are both less than d, they are the WSOs.

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

(a) static attributes of objects



(b) distances of objects to p_1 (c) distances of objects to p_2

Figure 1. Example of the continuous within skyline query.

The processing techniques for the continuous within skyline query developed in [8] focus exclusively on a static road network, where the information of objects (i.e., their object attributes) and the conditions of roads remain unchanged. However, in real-world applications, object information and road conditions inevitably vary with time. For example, a hotel offers a 20% discount so as to attract more customers (that is, varying the "price" attribute), and a crash obstructs the road for several hours (in this case, the length of the road is changed to ∞). Such varying information about objects and roads may outdate the previous query result, so that the continuous within skyline query needs to be evaluated again. This incurs the problem of high re-evaluation cost, which severely limits the applicability of the approaches in [8] in practice.

In this paper, we address the issue of efficiently processing the continuous within skyline query in dynamic road networks with time-varying information, where three types of time-varying information are taken into account. The first type of time-varying information is the time-varying object attribute, in which an object o changes its attribute from $o.a$ to $o.a'$. The second type is the time-varying edge length, where the length of an edge e changes from $e.len$ to ∞ (meaning that the road is temporarily closed) or from ∞ to $e.len$ (i.e., the road now is passable). The last one is the time-varying query path,

where the query path P_q has been changed to P'_q because there are road congestions or accidents on the roads in P_q.

To provide real-time processing of the above time-varying information, we need to quickly examine whether the WSOs of the continuous within skyline query are affected by the time-varying information and then evaluate the new WSOs if necessary. Therefore, we design three elaborate data structures, the object attribute dominating matrix (OADM), the road distance sorted list (RDSL) and the skyline object expansion tree (SOET), to adequately maintain the information of objects and the road network, which can be used to facilitate the task of quickly determining which time-varying information influences the query result. Moreover, we develop an efficient algorithm, namely the within skyline object updating algorithm, combined with the three data structures to rapidly evaluate the new result of the continuous within skyline query. For better readability, Table 1 summarizes the notations used.

Table 1. The notations.

Notation	Description
S_o	a set of data objects
P_q	a path along which the query object moves
d	a user-defined distance
WSOs	the within skyline objects
Time-varying object attribute	o changes its attribute from $o.a$ to $o.a'$
Time-varying edge length	e's length is changed from $e.len$ to ∞ or from ∞ to $e.len$
Time-varying query path	P_q is changed to P'_q

The main contributions of this paper are summarized as follows.

- We address the issue of efficiently processing the continuous within skyline query in dynamic road networks, where three types of time-varying information, the time-varying object attribute, the time-varying edge length and the time-varying query path, are taken into account in query processing.
- Three data structures, OADM, RDSL and SOET, are designed to adequately maintain the information of objects and the road network, in order to efficiently handle the time-varying information.
- We propose the within skyline object updating algorithm, combined with the three data structures, to rapidly evaluate the new query result affected by the time-varying information.
- A comprehensive set of experiments is conducted to demonstrate the merits of the proposed approaches.

The remainder of this paper is organized as follows. Section 2 reviews some related works. In Section 3, we present the three data structures, SADM, RDSL and SOET. Section 4 illustrates how the within skyline object updating algorithm works. Section 5 shows extensive experiments on the performance of the proposed methods. Finally, Section 6 concludes the paper with directions on future work.

2. Related Works

The skyline query is first studied in the area of computational geometry [9–13]. Several processing methods have been proposed to solve the continuous skyline query in Euclidean spaces. Cheema et al. [14] study the problem of continuously monitoring a moving skyline query by using a safe zone-based approach. Zheng et al. [15] propose continuous skyline computation over an incremental motion model, where the query point moves incrementally in discrete time steps with no restrictions and predictability. Vu et al. [16] further address the issue of processing the skyline query for object data with uncertainty.

In recent years, processing skyline queries in road networks has received considerable attention. Deng et al. [17] extend the concept of the spatial skyline [7] to road networks and present the

multi-source skyline query (MSQ). 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 an m-dimensional point, where the value of the i-th dimension refers to the road distance between o and the i-th query object. Then, MSQ retrieves the skyline points that are not dominated in terms of the m dimensions. Deng et al. propose three algorithms, the Euclidean distance constraint (EDC), the 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. [18] propose 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 [19,20] consider 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. [21] study 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. [22] 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 its road distance to the query object and object attribute. 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 limit 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 a 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. Recently, Huang et al. [8] propose several approaches to process the continuous within skyline query in a static road network. For each edge $e \in P_q$, the procedure includes: (1) obtaining a set of global within skyline objects (GWSO) such that object $o \in GWSO$ must be a WSO for point p on edge e; and (2) determining some points on edge e such that the WSOs between two consecutive points remain the same and finding the corresponding WSOs of these points. However, as discussed in the Introduction, the time-varying information limits the applicability of their approach in practice.

The related works mentioned above focus exclusively on: (1) processing the skyline queries in Euclidean spaces (e.g., [14,15]), where the distance between objects is computed by simply using the objects' locations rather than based on the connectivity of the road network; (2) answering the traditional skyline queries and their variants in the road networks (e.g., [7,17,18]); or (3) considering the continuous skyline query processing in a static road network (e.g., [8,22]), in which the information of objects and the conditions of the roads remain unchanged. In this paper, our efforts are devoted to overcoming the limitations of the previous works. That is, we investigate the continuous skyline problem in dynamic road networks with time-varying information.

3. Data Structures

The approach in [8] first determines a set GWSO for each edge $e \in P_q$ and then finds the WSOs for point p on edge e by taking into consideration the objects in GWSO only. The set GWSO is represented as a union of the WSOs of the two nodes connected by the edge e and the objects on e, where determining the WSOs of the nodes dominates the overall performance of processing the continuous within skyline query since it involves a large number of road distance computations. To efficiently process the continuous within skyline query in dynamic road networks with time-varying information, we design three data structures, the object attribute dominating matrix (OADM), the road distance sorted list (RDSL) and the skyline object expansion tree (SOET), to maintain the information of objects

and the road network. Benefiting from the data structures, we can quickly update the set GWSO for each edge $e \in P_q$ affected by the time-varying information. In the following, we describe separately the three data structures in detail and discuss how to update them for the time-varying information (i.e., the time-varying object attribute, the time-varying edge length and the time-varying query path).

3.1. Object Attribute Dominating Matrix

For each node n belonging to the query path P_q, the object attribute dominating matrix (OADM), represented as a 2D matrix, is designed to maintain the dominance relationships between objects, in terms of their object attributes. When the continuous within skyline query is processed, the rows and the columns of the OADM correspond to the objects whose road distances to n are less than or equal to the distance d. The reason why only such objects are kept in the OADM is that if their dominance relationships have been changed by the time-varying object attributes, the WSOs of node n could be affected. In other words, the time-varying object attributes of the objects not in the OADM cannot affect n's WSOs.

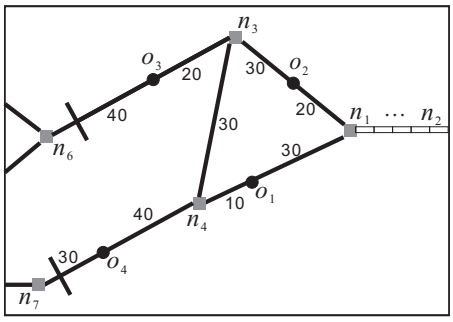
In the OADM, the value of each entry (o_i, o_j) is represented as follows:

$$(o_i, o_j) = \begin{cases} \times & \text{if } i = j, \\ 1 & \text{if } o_i \text{ dominates } o_j, \\ -1 & \text{if } o_j \text{ dominates } o_i, \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where "1" and "-1" mean that there is a dominance relationship between objects o_i and o_j, and "0" shows that o_i and o_j cannot dominate each other, in terms of their object attributes. Consider the example in Figure 2a, where four objects o_1 to o_4 are located on a road network, and have "price" and "rank" attributes. Assume that a continuous within skyline query is processed to find the skyline objects within the distance range $d = 100$. As the road distances between objects o_1 to o_4 and the node n_1 are less than or equal to 100, they all are kept in the OADM of n_1. In terms of the "price" and "rank" attributes of objects, the value of each entry in the OADM can be determined by using Equation (1), as shown in Figure 2b. In this figure, only objects o_1, o_2 and o_4 are the WSOs of node n_1 (because o_3 is dominated by o_2).

When the time-varying information (i.e., the time-varying object attribute, the time-varying edge length or the time-varying query path) has been changed, the OADM of node n needs to be updated accordingly, as discussed in the following.

- For the time-varying object attribute of an object o where o's attribute is changed from $o.a$ to $o.a'$: if object o appears in the OADM, then the dominance relationships between o and the other objects in the OADM are re-checked based on $o.a'$, so as to update the values of entries in the row and the column containing o. Otherwise, the OADM need not be updated as o does not appear in the OADM.
- For the time-varying edge length of an edge e where the length of e is changed from $e.len$ to ∞ (or ∞ to $e.len$): in the case that e's length is changed from $e.len$ to ∞, the road distances of some objects to n would increase. Conversely, changing e's length from ∞ to $e.len$ would decrease some objects' road distances. As a result, the objects whose road distances updated by the time-varying edge length are less (greater) than or equal to d need to be added into (removed from) the OADM. Then, the values of new entries in the OADM are determined using Equation (1).
- For the time-varying query path where the query path P_q is changed to P'_q: if the edge e connecting node n belongs to $P_q \bigcap P'_q$, then the OADM of n is still usable because e is not affected by the time-varying query path. Otherwise, e belongs to $P_q - P'_q$, and thus, the OADM of n is removed as n is not on the query path.

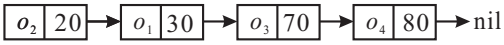


o_i	price	rank
o_1	120	5
o_2	30	4
o_3	35	3
o_4	20	2

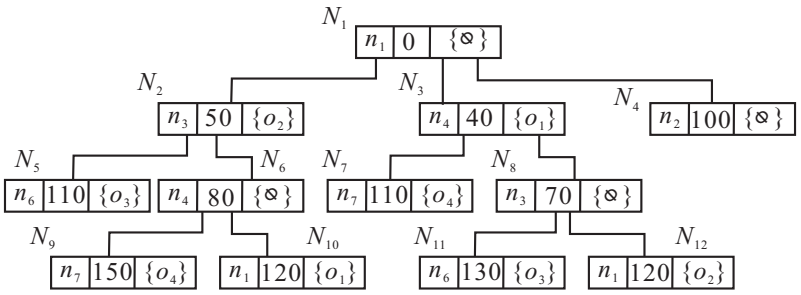
(a) Four objects with attributes

	o_1	o_2	o_3	o_4
o_1	X	0	0	0
o_2	0	X	1	0
o_3	0	-1	X	0
o_4	0	0	0	X

(b) *OADM*



(c) *RDSL*



(d) *SOET*

Figure 2. Three data structures: object attribute dominating matrix (OADM), the road distance sorted list (RDSL) and the skyline object expansion tree (SOET).

3.2. Road Distance Sorted List

As the road distance between each object and the query object plays an important role in determining the WSOs result, we design the road distance sorted list (RDSL) to store, for each node n belonging to the query path P_q, the objects in ascending order of their road distances to n. Similar to the OADM, only the objects whose road distances to n do not exceed the distance d are stored in the RDSL. If the position of an object o in the RDSL is in front of that of another object o', then o has a chance to dominate o' because of its smaller distance to n. Consider again Figure 2a, where objects o_1 to o_4 are within the distance range $d = 100$. According to their road distances to node n_1, the RDSL of n_1 is shown in Figure 2c. In this figure, the entry (o_2, o_3) in the OADM is equal to one (refer to Figure 2b), and o_2 is in front of o_3 in the RDSL. Therefore, o_3 is dominated by o_2 in terms of "price", "rank" and "distance" attributes. Note that although o_3 is not a WSO, it is still kept in the OADM and the RDSL because its distance to n_1 does not exceed d.

Due to the time-varying information, the road distances of objects in the RDSL of node n could be changed, as well as the order of objects. Once the RDSL is affected by the time-varying object attribute, the time-varying edge length or the time-varying query path, it needs to be updated as follows.

- For the time-varying object attribute of an object o: changing o's attribute from $o.a$ to $o.a'$ affects only the dominance relationship between o and the other objects, in terms of object attributes. The road distances of objects to node n remain unchanged so that the order of objects in the RDSL is not affected by the time-varying object attribute.
- For the time-varying edge length of an edge e: if the length of e is changed from $e.len$ to ∞ (i.e., e is temporarily closed), then the distances of some objects to n increase. For each object in the RDSL of n, once its road distance affected by e is greater than d, it needs to be removed. In the case that e's length is changed from ∞ to $e.len$, some objects would have decreasing distances to n because e is now passable. Therefore, such objects can be added into the RDSL if their decreasing distances do not exceed d. In addition to increasing or decreasing the road distance of the object, the time-varying edge length of e may also result in an adjustment in the order of objects in the RDSL.
- For the time-varying query path where the query path P_q is changed to P'_q: similar to the process of updating the OADM of node n mentioned in Section 3.1, the RDSL of n is removed only if n is not on the new query path P'_q.

3.3. Skyline Object Expansion Tree

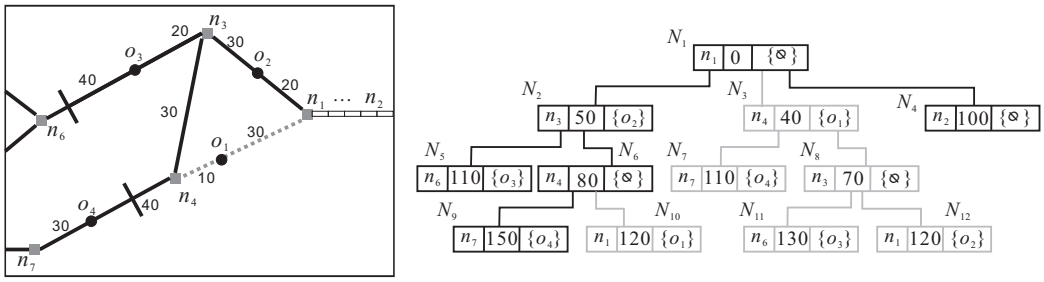
As discussed in the previous subsections, the time-varying edge length of an edge e results in the re-computations of the road distances of objects. However, computing the object distance by re-executing Dijkstra's algorithm or the A* algorithm, whenever the time-varying edge length is changed, would incur high cost in processing the continuous within skyline query. In order to greatly reduce the computation cost, the skyline object expansion tree (SOET) is designed to quickly compute the road distances of objects affected by the time-varying edge length, without the need to execute the specific algorithms.

For each node n belonging to the query path P_q, the SOET is built to keep information of the objects and the network nodes whose road distances to n are less than or equal to d. Then, by traversing the SOET, we can determine whether the time-varying edge length affects the objects within the distance range d and compute their new road distances to n if necessary. In the SOET, each tree node N has the structure $(n_i, dist_i, O_i, ptrs_N)$, where n_i refers to the network node id, $dist_i$ is the road distance of n_i to n, O_i is the set of objects on the edge connecting n_i and n_j (where n_j belongs to N's parent node) and $ptrs_N$ are the pointers to N's child nodes. For ease of exposition, the road network presented in Figure 2a is used again to illustrate the SOET structure, where the distance region starts at the node n_1 and ends at the vertical marks with distance $d = 100$. Initially, the root N_1 of the SOET stores the start node n_1's information, including the network node id n_1, the distance to n_1 (i.e., 0) and $O_1 = \{\emptyset\}$. As node n_1's adjacent nodes n_3, n_4 and n_2 in the road network have the distances less than or equal to 100, the tree nodes N_2, N_3 and N_4 in the forms of $(n_3, 50, \{o_2\})$, $(n_4, 40, \{o_1\})$ and $(n_2, 100, \{\emptyset\})$, respectively, are stored in the SOET and pointed by $ptrs_{N_1}$ (i.e., they are the child nodes of N_1). Then, consider the adjacent nodes n_6 and n_4 of n_3 in the road network. Because the distances of object o_3 and node n_4 to n_1 do not exceed 100, the tree nodes N_5 and N_6 of the SOET are represented as $(n_6, 110, \{o_3\})$ and $(n_4, 80, \{\emptyset\})$, respectively, and pointed by $ptrs_{N_2}$. Similarly, the tree nodes N_7 and N_8 in the forms of $(n_7, 110, \{o_4\})$ and $(n_3, 70, \{\emptyset\})$, respectively, are the child nodes of N_3. The corresponding SOET of node n_1, consisting of 12 tree nodes N_1 to N_{12}, is shown in Figure 2d. In the following, we discuss how to update the SOET of each node n when the time-varying information is changed.

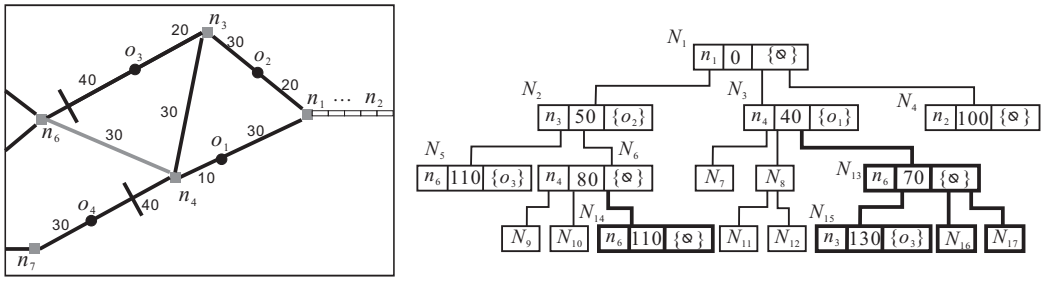
- For the time-varying object attribute of an object o: for the continuous within skyline query, the SOET of n remains valid regardless of the time-varying object attribute, because of the unchanged distance d.
- For the time-varying edge length of an edge e: the process of updating the SOET of node n needs to first determine which objects are affected by e and then re-compute their road distances to n. For the case that the length of edge e connecting nodes n_s and n_e is changed from $e.len$ to ∞ (i.e., e is temporarily closed), the SOET of node n is traversed from its root down to leaf level so as

to check whether there are parent-child relationships between the tree nodes containing n_s and n_e. If no such parent-child relationship exists, then the SOET of n remains valid because edge e connecting n_s and n_e falls out of the distance range d. Otherwise, the subtrees rooted at the tree nodes containing n_s and n_e are affected by e and need to be removed from the SOET. For the case that the length of edge e connecting nodes n_s and n_e is changed from ∞ to $e.len$, some objects' road distances can further decrease because e now is passable. As such, we perform a grown expansion starting from n_s and n_e to include the objects whose decreasing distances are less than or equal to d. Then, the subtree rooted at the tree node containing n_s (or n_e) is updated accordingly to contain information of such objects. Here, we explain the two cases using a concrete example, continuing the previous example in Figure 2. Assume that the edge connecting nodes n_1 and n_4 is temporarily closed (corresponding to the first case), as shown in Figure 3a. Having traversed the SOET of node n_1, we know that both the tree nodes N_1 and N_{10} (N_3 and N_6) contain node n_1 (n_4). As there is a parent-child relationship between N_1 and N_3 (N_6 and N_{10}), the subtree rooted at N_3 (N_{10}) is removed from the SOET (the shaded part in Figure 3a), so that the road distances of objects o_1 and o_4 are re-computed as ∞ (because o_1 is on e) and 120, respectively. As a result, objects o_1 and o_4 are removed from the OADM and the RDSL of n_1. Going back to the example in Figure 2, assume that the edge connecting nodes n_4 and n_6 is passable, and its length is equal to 30 (corresponding to the second case), as shown in Figure 3b. Starting from the tree nodes N_3 and N_6 containing node n_4, two new subtrees have been included into the SOET (the bold part in Figure 3b) because their updated distances do not exceed the distance 100.

- For the time-varying query path where the query path P_q is changed to P'_q: if the node n is not on the new query path P'_q, then the SOET of n is removed.



(a) Edge connecting n_1 and n_4 is closed



(b) Edge connecting n_4 and n_6 is passable

Figure 3. Update of the SOET.

4. Within Skyline Object Updating Algorithm

Recall that the WSOs result for each edge $e \in P_q$ is obtained from the set $GWSO$, which is a union of the WSOs of the two nodes connected by e and the objects on e. Here, we denote the WSOs set of node n as WSO_n. Determining the WSO_n dominates the overall performance of processing the

continuous within skyline query because it involves a large number of road distance computations. As the time-varying information (including the time-varying object attribute, the time-varying edge length and the time-varying query path) can potentially outdate the previous WSO_n, it needs to be updated if necessary. An intuitive method to update the WSO_n of node n is to re-execute the approach in [8] whenever the time-varying information is changed, which, however, would suffer from a long re-computation time for a large amount of time-varying information changed. Therefore, we develop the within skyline object updating algorithm to provide real-time processing of the time-varying object attribute, the time-varying edge length and the time-varying query path, so as to quickly update the WSO_n of each node n on P_q without the need to execute the approach in [8].

For the WSO_n set of each node n on P_q, the procedure of the within skyline object updating algorithm can be divided into three cases according to the type of time-varying information. The first case is that the time-varying information is changed by an object o (which may or may not be a WSO of n); the second one is that the time-varying information is changed by an edge e (that is, e is temporarily closed or now passable); and the last case is that the time-varying information corresponds to the time-varying query path (due to the road congestions or accidents on some roads belonging to P_q). In the following, we discuss the three cases separately.

4.1. Processing of Time-Varying Object Attribute

The main idea of processing the time-varying object attribute is to (1) determine whether the time-varying object attribute of object o affects the WSO_n by examining whether o appears in the OADM and the RDSL of n and (2) update the WSO_n by checking the value of entry in the OADM and the order of objects in the RDSL with respect to o. Suppose that object o changes its attribute from $o.a$ to $o.a'$. If o does not appear in the OADM and the RDSL of n, then the WSO_n of node n need not be updated regardless of object o's new attribute $o.a'$. This is because o falls outside the distance range d, and thus, the time-varying object attribute of o can be ignored. Otherwise (i.e., object o appears in the OADM and the RDSL), the dominance relationship between o and another object, say o', may be changed by $o.a'$ (that is, the values of entries (o, o') and (o', o) in the OADM have been modified), so that the previous WSO_n needs to be further verified. According to whether o and o' are contained in the WSO_n (i.e., they are the WSOs of n), the process of updating the WSO_n can be divided into four cases: (1) $o \in WSO_n$ and $o' \in WSO_n$, (2) $o \in WSO_n$ but $o' \notin WSO_n$, (3) $o \notin WSO_n$, but $o' \in WSO_n$, and (4) $o \notin WSO_n$ and $o' \notin WSO_n$, which are separately discussed in the following.

The first case where $o \in WSO_n$ and $o' \in WSO_n$ implies that no object can dominate o and o' in terms of the object attributes and the "distance" attribute (closer to n than o and o'). Let us observe the value of entry (o, o') in the OADM affected by the time-varying object attribute, consisting of the following six conditions, to determine whether the WSO_n needs to be updated.

- (o, o') is changed from -1 to zero: this means that there is no longer a dominance relationship between o and o' (i.e., o and o' cannot dominate each other). Because o and o' are still contained in the WSO_n, changing (o, o') from -1–0 cannot affect the WSO_n and can be ignored.
- (o, o') is changed from one to zero: same as the first condition, the WSO_n is not affected even though (o, o') is changed to zero.
- (o, o') is changed from -1 to one: o now can dominate o' in terms of the object attributes. Note that since o is previously dominated by o' in terms of the object attributes ($(o, o') = -1$), but still can be a WSO, o must be closer to n than o' (i.e., o is in front of o' in the RDSL). Therefore, when $(o, o') = 1$, o' needs to be removed from the WSO_n.
- (o, o') is changed from one to -1: similar to the third condition, o has to be removed from the WSO_n because it is dominated by o' in terms of the object attributes and the "distance" attribute.
- (o, o') is changed from zero to -1: as $(o, o') = -1$, o now is dominated by o' in terms of the object attributes. By checking the order of o and o' in the RDSL, o should be removed from the WSO_n if it is behind o' (otherwise, o is still kept in the WSO_n because of its better "distance" attribute).

- (o, o') is changed from zero to one: same as the fifth condition, o' is removed from the WSO_n as long as o has a better order than o' in the RDSL.

For the second case where $o \in WSO_n$, but $o' \notin WSO_n$, there exists an object, say o'', that can dominate o' in terms of the object attributes (i.e., $(o', o'') = -1$ in the OADM) and the "distance" attribute (i.e., o' is behind o'' in the RDSL). Note that o may be the object dominating o' (i.e., $o'' = o$). The six conditions regarding the value of entry (o, o') are described as follows.

- (o, o') is changed from -1 to one: $(o, o') = 0$ means that o is no longer dominated by o', which will not affect the objects in the WSO_n. Thus, the change of (o, o') can be ignored.
- (o, o') is changed from one to zero: as the dominance relationship between o and o' does not exist, o' can be promoted to a WSO (i.e., o' is added to the WSO_n) if (1) $(o', o'') = -1$ does not appear in the OADM or (2) $(o', o'') = -1$ appears, but o' is in front of o'' in the RDSL. Otherwise, o'' can still dominate o', so that the WSO_n remains unchanged.
- (o, o') is changed from -1 to one: similar to the first condition, the WSO_n is not affected by changing (o, o') to one.
- (o, o') is changed from one to -1: it implies that (1) o' can dominate o if it has a better "distance" attribute, and (2) o' can be a WSO if no object dominates it. For (1), we only need to check the positions of o and o' in the RDSL. If o is behind o', then o is removed from the WSO_n. Otherwise, o is still kept in the WSO_n. For (2), the process for the second condition can be applied to determine whether o' becomes a WSO.
- (o, o') is changed from zero to -1: because o now is dominated by o' in terms of the object attributes, o should be removed from the WSO_n once o' is better than o in the "distance" attribute. By checking the order of o and o' in the RDSL, o is removed from (kept in) the WSO_n if it is behind (in front of) o'.
- (o, o') is changed from zero to one: same as the third condition, the WSO_n is not affected by the change of (o, o').

The third case is that $o \notin WSO_n$, but $o' \in WSO_n$. In this case, there is an object o'' dominating o because $(o, o'') = -1$ in the OADM and o'' is in front of o in the RDSL (where o'' may be o'). The following six conditions describe how the WSO_n is updated according to different values of (o, o').

- (o, o') is changed from -1 to zero: $(o, o') = 0$ means that o is no longer dominated by o' in terms of the object attributes, and thus, o can be a WSO if o' is the only object previously dominating o. In other words, if there still exists an object o'' that leads to $(o, o'') = -1$ and is in front of o in the RDSL, then o cannot be added to the WSO_n. Otherwise, o becomes a new WSO and is added to the WSO_n.
- (o, o') is changed from one to zero: even though the dominance relationship between o and o' is affected by changing (o, o') from one to zero, the WSO_n remains unchanged (that is, $o \notin SO_n$ and $o' \in WSO_n$).
- (o, o') is changed from -1 to one: because o now can dominate o' in terms of the object attributes, there is a chance that o (o') is added to (removed from) the WSO_n. Here, the process for the first condition can be applied to determine whether o is added to the WSO_n or not. On the other hand, whether o' is removed from the WSO_n can be determined by checking the order of o and o' in the RDSL.
- (o, o') is changed from one to -1: similar to the second condition, the WSO_n would not be affected regardless of the change of (o, o').
- (o, o') is changed from zero to -1: $(o, o') = -1$ means that o now is dominated by o' in terms of the object attribute. As $o \notin SO_n$ (that is, an object can dominate o), the changed dominance relationship between o and o' cannot affect the WSO_n.
- (o, o') is changed from zero to one: due to $(o, o') = 1$, o can be used to dominate o' by checking whether it is in front of o' in the RDSL. If so, o' is removed from the WSO_n. Otherwise, the WSO_n need not be updated.

Considering the last case that $o \notin WSO_n$ and $o' \notin WSO_n$, there exists an object o'' (or o''') dominating o (or o') in terms of the object attributes and the "distance" attribute. In the following, we discuss how the WSO_n is affected under various values of (o, o').

- (o, o') is changed from -1 to zero: if o' is the object previously dominating o (i.e., $o' = o''$) and there is no other object whose $(o, o'') = -1$ in the OADM and in front of o in the RDSL, then o can be added to the WSO_n. Otherwise, the WSO_n is not affected by changing (o, o') from -1 to zero.
- (o, o') is changed from one to zero: although o' is no longer dominated by o, it cannot be promoted to a WSO based on the following reasons: (1) if $o = o'''$, then the object o'' previously dominating o is still better than o' in the object attributes and the "distance" attribute, even though the dominance relationship between o and o' has been changed, or (2) if $o \neq o'''$, then there is an object o''' dominating o'.
- (o, o') is changed from -1 to one: similar to the first condition, o can be a new WSO and added to the WSO_n when o' is the only object previously dominating o in terms of the object attributes and the "distance" attribute.
- (o, o') is changed from one to -1: same as the second condition, there must be an object dominating o' in terms of the object attributes and the "distance" attribute, regardless of the change of (o, o').
- (o, o') is changed from zero to -1: even if the dominance relationship between o and o' is changed, an object o'' (or o''') can still dominate o (or o') so that $o \notin WSO_n$ (or $o' \notin WSO_n$).
- (o, o') is changed from zero to one: $(o, o') = 1$ means that o now can dominate o', which, however, cannot affect the WSO_n because both o and o' are not contained in the WSO_n.

The WSO_n of node n can be updated by considering that each entry in the row containing the object o changing its attribute from $o.a$ to $o.a'$ corresponds to which case mentioned above. The example in Figure 4 is used to illustrate how to update the WSO_n of node n when the time-varying object attribute of object o is changed. As shown in Figure 4a, there are four objects o_1 to o_4 with two attributes: "price" and "rank" and the "distance" attribute (where the "distance" attribute refers to the road distance of object n). Based on the object attributes and the "distance" attribute, o_3 and o_4 are dominated by o_2 and o_1, respectively, and thus, o_1 and o_2 are the WSOs of n (i.e., $WSO_n = \{o_1, o_2\}$). The OADM of n is also shown in Figure 4a. Suppose that object o_2's "price" and "rank" attributes are changed from 30 to 40 and from four to three, respectively. Due to the change of o_2's attributes, the value of entry in the row and the column containing o_2 in the OADM has been updated (refer to Figure 4b). As objects $o_2 \in WSO_n$ and $o_3 \notin WSO_n$ and (o_2, o_3) in the OADM are changed from one to -1, meaning that o_2 now is dominated by o_3 in terms of "price" and "rank" attributes, the WSO_n is updated according to the fourth condition of the second case. Although, $(o_2, o_3) = -1$, o_2 is still kept in the WSO_n because of its better "distance" attribute than o_3. As for o_3, it can be added to the WSO_n as $(o_3, o_1) \neq -1$ and $(o_3, o_4) \neq -1$. That is, $WSO_n = \{o_1, o_2, o_3\}$. Let us consider another scenario where object o_4 changes its "price" ("rank") attribute from 130 (5) to 20 (3). The time-varying object attribute of o_4 affects the dominance relationship between o_4 and o_1, as well as o_4 and o_3. As shown in Figure 4c, the values of entries (o_4, o_1) and (o_4, o_3) are changed to zero and one, respectively. For the case that $o_4 \notin WSO_n$, $o_1 \in WSO_n$ and changing (o_4, o_1) from -1 to zero, which corresponds to the first condition of the third case, o_4 is added to the WSO_n. For the case that $o_4 \notin WSO_n$, $o_3 \notin WSO_n$ and updating (o_4, o_3) from zero to one, corresponding to the last condition of the fourth case, the WSO_n is not affected by $(o_4, o_3) = 1$. Finally, $WSO_n = \{o_1, o_2, o_4\}$.

o_i	price	rank	distance
o_1	120	5	20
o_2	30	4	30
o_3	35	3	70
o_4	130	5	80

	o_1	o_2	o_3	o_4
o_1	X	0	0	1
o_2	0	X	1	0
o_3	0	-1	X	0
o_4	-1	0	0	X

(a) Four objects with attributes

o_i	price	rank	distance
o_1	120	5	20
o_2	40	3	30
o_3	35	3	70
o_4	130	5	80

	o_1	o_2	o_3	o_4
o_1	X	0	0	1
o_2	0	X	-1	0
o_3	0	1	X	0
o_4	-1	0	0	X

(b) Time-varying object attribute of o_2

o_i	price	rank	distance
o_1	120	5	20
o_2	30	4	30
o_3	35	3	70
o_4	20	3	80

	o_1	o_2	o_3	o_4
o_1	X	0	0	0
o_2	0	X	1	0
o_3	0	-1	X	-1
o_4	0	0	1	X

(c) Time-varying object attribute of o_4

Figure 4. Example of processing the time-varying object attribute.

4.2. Processing of Time-Varying Edge Length

Different from the time-varying object attribute, which changes only the value of the entry in the OADM, the time-varying edge length may affect the number of objects in the OADM and the RDSL. Specifically, an object could be added to (or removed from) the OADM and the RDSL because of its decreasing (or increasing) distance affected by the time-varying edge length. Moreover, the time-varying edge length may also result in an adjustment in the order of objects in the RDSL. As such, when the time-varying edge length of an edge e is changed, the OADM, the RDSL and the SOET of node n are updated accordingly, which are then used to determine whether the WSO_n of node n is affected by edge e's time-varying edge length.

Consider the case that the length of edge e connecting nodes n_s and n_e is changed from $e.len$ to ∞ (i.e., e is temporarily closed). In the SOET of node n, if there is no parent-child relationship between the tree nodes containing n_s and n_e, then changing e's length from $e.len$ to ∞ cannot affect the OADM, the RDSL and the SOET of node n, as well as its WSO_n (i.e., the WSOs result remains valid). Otherwise, the subtrees rooted at the tree nodes containing n_s and n_e are affected by e and removed from the SOET. As a result, the road distances between the node n and the objects contained in the subtrees removed need to be updated by traversing the remaining part of the SOET. Suppose that the updated road distance of object o to node n increases from $o.dist$ to $o.dist'$. Then, the process of updating the WSO_n is divided into the following three cases: (1) object o does not appear in the RDSL; (2) object o appears in the RDSL, but $o \notin WSO_n$; and (3) object o appears in the RDSL and $o \in WSO_n$.

- Object o does not appear in the RDSL: this means that the road distance $o.dist$ of object o to node n is greater than the distance d. As $o.dist' > o.dist$, object o cannot appear in the RDSL. Therefore, the WSO_n need not be updated.
- Object o appears in the RDSL, but $o \notin WSO_n$: if the updated distance $o.dist'$ of object o exceeds d, then o is directly removed from the OADM and the RDSL of node n (while the WSO_n remains unchanged). Otherwise (i.e., $o.dist' \leq d$), the order of the RDSL has to be adjusted according to $o.dist'$. As for the OADM and the WSO_n obtained by previous, they are still valid because o's dominance relationship is not affected and $o \notin WSO_n$.
- Object o appears in the RDSL and $o \in WSO_n$: in the case where the updated distance $o.dist'$ is greater than d, object o needs to be removed from the OADM, the RDSL and the WSO_n of node n (meaning that the WSOs result has been changed). In addition, some object, say o', that appears in the RDSL, but not in the WSO_n, can be promoted to the skyline object if it is dominated only by object o (i.e., $(o, o') = 1$ in the OADM and o is in front of o' in the RDSL). By looking up the row containing o in the previous OADM, we know the objects with $(o, o') = 1$. For each object o', if there is no object o'' such that $(o'', o') = 1$ in the OADM and o'' is in front of o' in the RDSL, then o' can be added to WSO_n as it is now a skyline object, and its distance to n is within the distance range d. In the case where the updated distance $o.dist' \leq d$, object o is still kept in the OADM and the RDSL, but it has a higher position in the RDSL than before. In this case, object o could be removed from the WSO_n because of its increasing $o.dist'$. Conversely, the objects previously dominated by o have a chance to be added to the WSO_n if they are no longer dominated. To determine whether object o is removed from the WSO_n or not, the row containing o in the OADM is first checked to find each object o' with $(o, o') = -1$, and then, the updated position of o in the RDSL is compared to that of o'. Object o is removed from the WSO_n only when o' is in front of o. On the other hand, to determine whether object previously dominated by object o can be added to the WSO_n, each object o' that has $(o, o') = 1$ in the OADM and is behind o in the RDSL is considered. Once o' is now in front of o (that is, o' is no longer dominated by o) and no object o'' in front of o' and $(o'', o') = 1$ can be found, o' is added to the WSO_n. Otherwise, o' is still dominated by o'', so that $o' \notin WSO_n$.

Consider the case that the length of edge e connecting nodes n_s and n_e is changed from ∞ to $e.len$ (that is, e is now passable). If nodes n_s and n_e are not contained in the tree nodes of the SOET of node n, which implies that edge e falls outside the distance range d, then the OADM, the RDSL and the WSO_n obtained previously remain valid regardless of changing e's length from ∞ to $e.len$. In other words, the OADM, the RDSLand the WSO_n may need to be updated if any tree node of the SOET contains n_s or n_e. This is because the road distances between node n and the objects affected by edge e may further decrease, leading to more or less objects in the OADM, the RDSL and the WSO_n. Therefore, a grown network expansion starting from n_s and n_e is performed to include the objects whose decreasing distances (attributed to e) are less than or equal to d. Furthermore, the subtree rooted at the tree node containing n_s (or n_e) is updated accordingly to contain information of such objects. Suppose that the road distance of each affected object o to node n decreases from $o.dist$ to $o.dist'$ (note that $o.dist' \leq d$). Similar to the case mentioned above, whether the WSO_n needs to be updated can be determined based on: (1) object o does not appear in the RDSL; (2) object o appears in the RDSL, but $o \notin WSO_n$; and (3) object o appears in the RDSL and $o \in WSO_n$.

- Object o does not appear in the RDSL: as the updated road distance $o.dist'$ of object o to node n is less than or equal to d, object o has to be added into the OADM (where the values of entries in the row and the column containing o are computed using Equation (1)) and the RDSL (in which the order of o is determined according to its $o.dist'$), resulting in that (a) object o could be added into the WSO_n and (b) some object $o' \in WSO_n$ could be removed from the WSO_n. For (a), if there is an object o' with $(o, o') = -1$ in the OADM and in front of o in the RDSL, then o is still dominated by o' and, thus, cannot be added to the WSO_n. Otherwise, $o \in WSO_n$. For (b), object $o' \in WSO_n$ is removed from the WSO_n only when o is better than o' in terms of the object attributes and the

"distance" attribute. Therefore, if the entry $(o, o') = 1$ exists in the OADM and o is in front of o' in the RDSL, then o' is removed from WSO_n.

- Object o appears in the RDSL, but $o \notin WSO_n$: as $o.dist' < o.dist$, object o has a better order in the RDSL than before. As a result, there is a chance that object $o' \in WSO_n$ behind o's current position in the RDSL (meaning that o is better than o' in the "distance" attribute) is now dominated by o. Having checked the row containing o in the OADM, such an object o' can be removed from the WSO_n if $(o, o') = 1$ exists in the OADM (that is, o is also better than o' in the object attributes). On the other hand, due to the decreasing distance $o.dist'$, all of the objects in front of o may no longer dominate o so that o can be added to the WSO_n. Consider again the row containing o in the OADM. If no entry $(o, o') = -1$, where o' is in front of o in the RDSL, can be found, then o is promoted to a skyline object (i.e., $o \in WSO_n$).
- Object o appears in the RDSL and $o \in WSO_n$: in this case, object o would still be kept in the WSO_n because its road distance decreases to $o.dist'$. Furthermore, o may dominate the other objects in the WSO_n, as it now has a better "distance" attribute. For each object $o' \in WSO_n$, once the two conditions that $(o, o') = 1$ in the OADM and o is in front of o' in the RDSL hold, o' is removed from the WSO_n.

4.3. Processing of the Time-Varying Query Path

Given the query path P_q, the continuous within skyline query finds the WSOs for each point on P_q. Moreover, the OADM, the RDSL, the SOET and the WSO_n of each node n on P_q are maintained for efficient processing of the time-varying information. Nevertheless, the query path P_q may be changed to P'_q due to the road congestions or accidents on some roads (i.e., edges) belonging to P_q, making the OADM, the RDSL, the SOET and the WSO_n of the two nodes connecting such edges outdated.

To process the time-varying query path, each edge e belonging to $P_q \bigcap P'_q$ is first determined because the OADM, the RDSL and the SOET of node n connecting e remain valid, as well as its WSO_n. Then, it is only necessary to execute the continuous within skyline query for the remaining part of the updated query path (that is, $P'_q - P_q$), so as to obtain the WSOs of the nodes on $P'_q - P_q$ and also their OADM, RDSL and SOET. The procedure of the within skyline object updating algorithm is detailed in Algorithm 1.

Algorithm 1: The within skyline object updating algorithm.

Input : The time-varying information changed by an object o, an edge e, or the query path P_q, and the original WSO_n set of each node n on P_q
Output: The updated WSO_n set of node n

```
/* corresponding to the time-varying object attribute                    */
if (object o changes its attribute from o.a to o.a') then
    if (o does not appear in OADM and RDSL) then
        return the original WSO_n;
    else
        update the WSO_n according to the four cases described in Section 4.1;
/* corresponding to the time-varying edge length                         */
if (the length of edge e is changed from e.len to ∞) then
    update the WSO_n based on the first case mentioned in Section 4.2;
/* changing e's length from ∞ to e.len                                   */
else
    update the WSO_n based on the second case mentioned in Section 4.2;
/* corresponding to the time-varying query path                          */
if (the query path P_q is changed to P'_q) then
    update the WSO_n using the process discussed in Section 4.3;
```

5. Performance Evaluation

In this section, we first conduct two sets of experiments to investigate the efficiency of the proposed within skyline object updating algorithm, compared to the Cd-SQ algorithm in [8] which operates without the support of the OADM, the RDSL and the SOET presented in this paper. The first set of experiments studies the effects of four important factors on the performance of processing the continuous within skyline query. The second one investigates how well the within skyline object updating algorithm and its competitor work for dynamic road networks. Then, we discuss the space requirements of the within skyline object updating algorithm and the Cd-SQ algorithm, respectively.

5.1. Experimental Settings

All experiments are performed on a PC with AMD Athlon X2 5200 CPU and 2 GB RAM. The algorithms are implemented in C++. As shown in Figure 5, the road map, Oldenburg (a city in Germany) [23], consisting of about 6000 nodes and 7000 edges, is used in our simulation. The set of data objects (varying from 10 K to 200 K) is generated using the generator proposed in [24], which is the most popular framework used in road networks [25–27]. Each data object has several attributes (ranging from two to six) whose values are normalized in the range $[0, 1]$. In the experimental space, we also generate 30 query paths, each of which consists of multiple edges (ranging from one to 16). For each query path P_q, we perform a continuous within skyline query to find the WSOs for each point on P_q, in which the distance d varies from 0.1% to 3% of the entire space. Based on the WSOs result obtained, the OADM, the RDSL and the SOET of each node belonging to P_q are maintained for the within skyline object updating algorithm. To investigate the effect of the time-varying information on the performance of the proposed approaches, for each query path P_q, we set the default number of updates to 20, and each update process involves x% of objects changing their attributes (where x varies from zero to 20), y% of edges updating their lengths (in which y varies from zero to 16), and z% of edges on P_q affected by the time-varying query path (where z varies from zero to 25). The performance is measured by the average CPU time in performing workloads of 30 continuous within skyline queries with 20 updates. Table 2 summarizes the parameters under investigation, along with their default values and ranges.

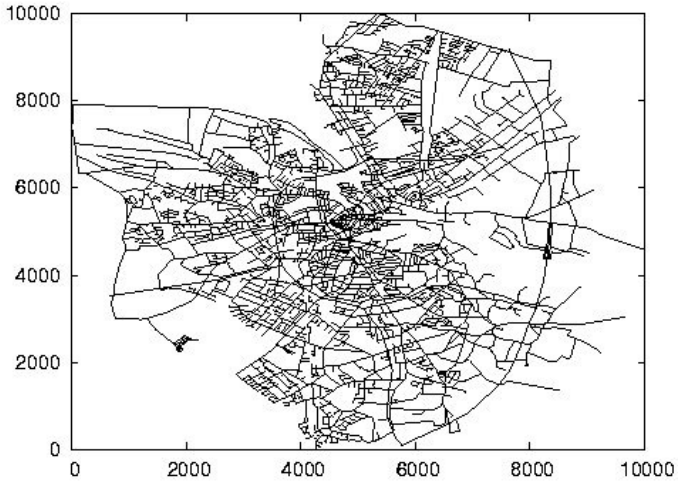


Figure 5. Oldenburg road network.

Table 2. System parameters.

Parameter	Default	Range
Number of objects	100 (K)	10, 50, 100, 150, 200 (K)
Number of attributes	4	2, 3, 4, 5, 6
Length of path length	4	1, 2, 4, 8, 16
Distance d	1%	0.1, 0.5, 1, 2, 3 (%)
Time-varying object attribute x	10%	0, 5, 10, 15, 20 (%)
Time-varying edge length y	4%	0, 2, 4, 8, 16 (%)
Time-varying query path z	10%	0, 5, 10, 20, 25 (%)

5.2. Effect of Four Important Factors

The first set of experiments demonstrates the efficiency of the within skyline object updating algorithm (WSOU) by comparing it with the Cd-SQ algorithm, in terms of the CPU time. Four experiments are implemented to investigate the effects of four important factors on the performance of processing the continuous within skyline query with time-varying information. These important factors are the number of objects, the number of attributes, the length of query path and the value of d.

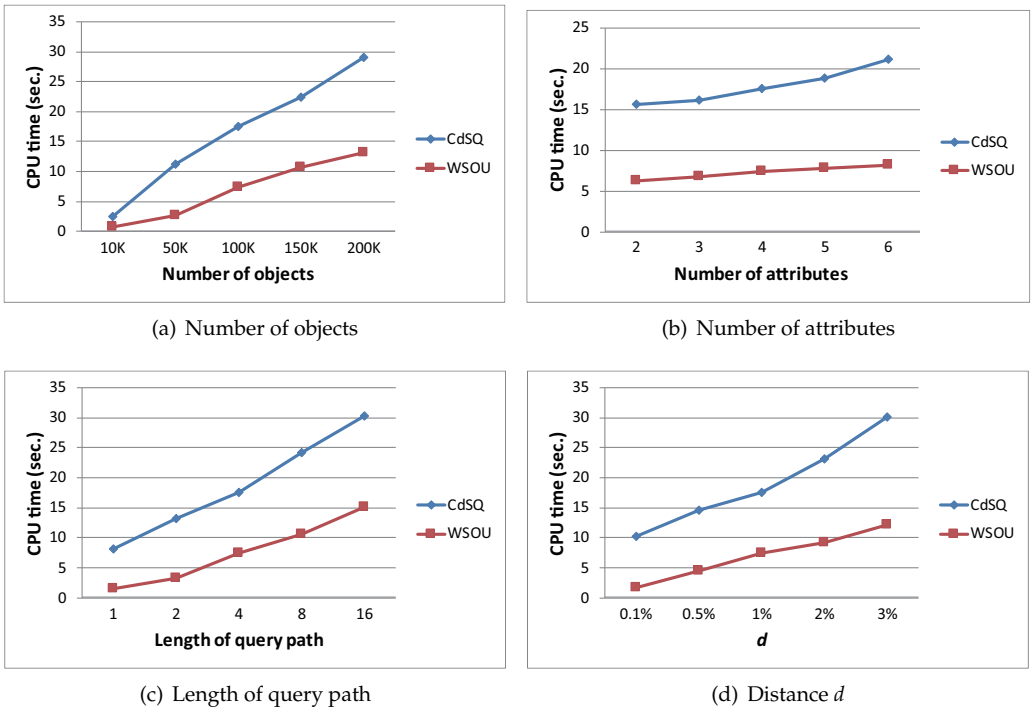
Figure 6a studies the effect of the number of objects on the performance of the WSOU algorithm and the CdSQ algorithm. In this experiment, we vary the number of objects from 10 K to 200 K and measure the CPU time for the proposed algorithms. As we can see from the simulation, the performance gap between the WSOU algorithm and the CdSQ algorithm increases with the increasing number of objects. The reason is that for the CdSQ algorithm, the query needs to be re-executed whenever an update occurs, so that when the number of objects increases, more processing time is spent on the repetitive query execution. However, for the WSOU algorithm, the query re-execution can be effectively reduced by taking advantage of the three data structures, the OADM, the RDSL and the SOET, because most of the time-varying information actually does not affect the query result and, thus, can be directly ignored.

Figure 6b illustrates the performance of the WSOU algorithm and the CdSQ algorithm as a function of the number of attributes (ranging from two to six). The simulation shows that the performance of the CdSQ algorithm degrades linearly as the number of objects increases, while the WSOU algorithm is not sensitive to the number of attributes. This is because more object attributes would result in more skyline objects (in terms of the object attributes), and thus, more dominance tests are performed for the CdSQ algorithm. The simulation confirms again that applying the OADM, the RDSL and the SOET in the WSOU algorithm can efficiently improve the performance of processing the continuous within skyline query with time-varying information.

In Figure 6c, we investigate the efficiency of the WSOU algorithm and the CdSQ algorithm by measuring the CPU cost under different lengths of query path (varying from one to 16 edges). When the length of query path increases, the CPU overhead for both algorithms increases because for a longer query path, (1) the CdSQ algorithm takes more CPU time to determine the new query result affected by each update and (2) the WSOU algorithm requires building more OADM, RDSL and SOET for the nodes belonging to the query path. Nevertheless, the WSOU algorithm outperforms its competitor in all cases. The improvement is due to the fact that the WSOU algorithm considers only the time-varying information affecting the query result (however, the CdSQ algorithm needs to re-execute the query no matter whether the result is affected or not).

Finally, in Figure 6d, we compare the performance of the WSOU algorithm and the CdSQ algorithm with respect to different values of d (ranging from 0.1% to 3%). For the WSOU algorithm, a greater d leads to more objects kept in the OADM, the RDSL and the SOET (i.e., more objects with the distance range d), and thus, the required CPU time increases slightly with the increasing d (but basically, the CPU time is still under 10 s). For the CdSQ algorithm, a smaller d is favorable because less computation of road distances of the objects within the distance range d has to be performed. However, when d increases to a larger value (e.g., 3%), meaning that more qualifying

objects are considered, the CPU cost of the C*d*SQ algorithm becomes much higher than that of the WSOU algorithm (because it has no chance of avoiding the query re-execution).



(a) Number of objects (b) Number of attributes

(c) Length of query path (d) Distance *d*

Figure 6. Effect of four important factors. WSOU, within skyline object updating.

5.3. Effect of Time-Varying Information

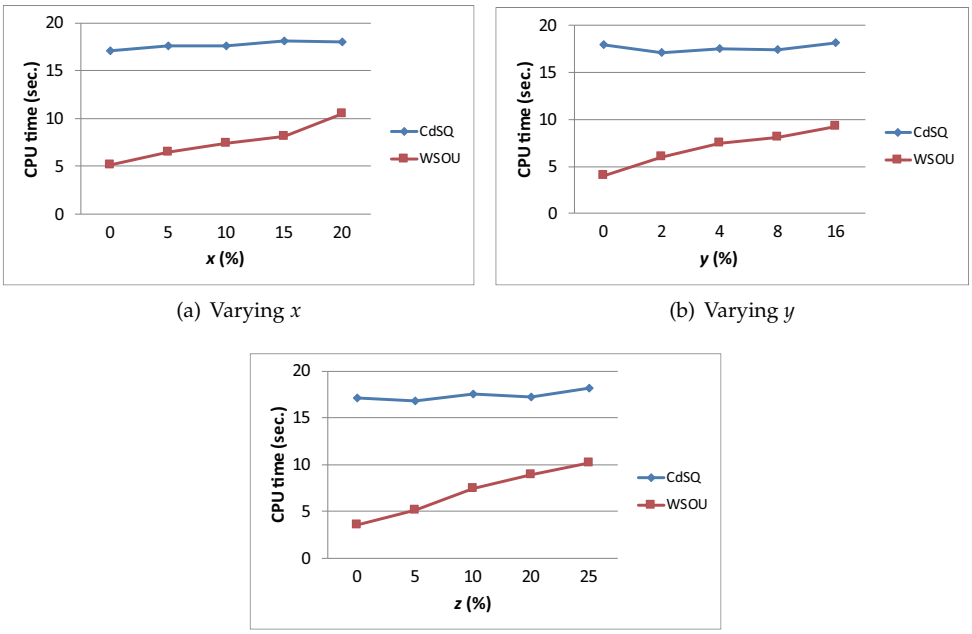
The second set of experiments studies how well the WSOU algorithm and the C*d*SQ algorithm work for dynamic road networks by varying the percentages of: (1) objects changing attributes; (2) edges updating lengths; and (3) affected edges on the query path. Hereafter, for the x-axis in the figures, the three percentages refer to x%, y% and z%, respectively.

Figure 7a investigates the impact of various numbers of objects changing their attributes (i.e., varying x%) on the performance of the WSOU algorithm and the C*d*SQ algorithm. In the experiment, we vary x% from 0% to 20% and measure the CPU cost for the two algorithms. The simulation shows that the WSOU algorithm outperforms the C*d*SQ algorithm by a factor of 1.5 to three in terms of the CPU time. The large difference in CPU time between the two algorithms comes from: (1) for the C*d*SQ algorithm, the query re-execution at each update is inevitable, no matter what the number of objects changing attributes is (that is why its cost keeps almost constant); while (2) for the WSOU algorithm, the query re-execution can be avoided by determining which of the objects changing attributes fall outside the distance range d and by only updating the affected query result if necessary, utilizing the OADM, the RDSL and the SOET.

In Figure 7b, we measure the CPU overhead of the WSOU algorithm and the C*d*SQ algorithm under different numbers of edges changing lengths (ranging from 0% to 16%). Similar to the reason explained in the previous experiment, the CPU time for the C*d*SQ algorithm is nearly constant as the number of edges changing lengths increases (i.e., increasing y%). As for the WSOU algorithm, its CPU time shows an increasing trend because as y becomes greater, the chance of the edge falling within the distance range d increases so that more distance updates of objects are required. Nevertheless, the WSOU algorithm can still achieve better performance than the C*d*SQ algorithm in all cases (by a factor of up to two), as the affected object distances can be updated by traversing the SOET (but for

the C*d*SQ algorithm, Dijkstra's algorithm or the A* algorithm must be executed to compute the affected distances).

The last experiment shown in Figure 7c evaluates the effect of the time-varying query path on the CPU time of the WSOU algorithm and the C*d*SQ algorithm, in which z% ranges from 0% to 25%. A large value of z leads to more number of edges belonging to the query path P_q and affected by the time-varying query path (that is, more edges belonging to $P'_q - P_q$, where P'_q is the updated query path). For the WSOU algorithm, the CPU cost increases with z, because more processing time is required for determining the WSOs of edges belonging to $P'_q - P_q$ using the continuous within skyline query. Note that the WSOs of edges belonging to $P_q \bigcap P'_q$ are still valid, for which the continuous within skyline query need not be executed. For the C*d*SQ algorithm, the CPU time is almost not affected by the value of z. The reason is that the continuous within skyline query has to be re-executed, regardless of whether the query path is changed or not. Again, the simulation shows that the WSOU algorithm performs better than the C*d*SQ algorithm in all cases.



(a) Varying x

(b) Varying y

(c) Varying z

Figure 7. Effect of time-varying information.

5.4. Discussion of the Space Requirement

In this subsection, we discuss the space complexity of the WSOU algorithm and the C*d*SQ algorithm, in terms of the data structures used in query processing. For the C*d*SQ algorithm (refer to [8]), three tables, T_{edge}, T_{node} and T_{obj}, are used to represent the road network and maintain information of data objects. During the course of query processing, the query path P_q is divided into a set of edges, each of which requires a $GWSO$ set to keep the within skyline objects on it. Assume that the query path P_q consists of n edges, and the average size of $GWSO$ set for each edge is $|GWSO|$. The space complexity of the C*d*SQ algorithm is estimated as $O(|T_{edge}| + |T_{node}| + |T_{obj}| + n \times |GWSO|)$, where $|T_{edge}|$, $|T_{node}|$ and $|T_{obj}|$ refer the sizes of the tables T_{edge}, T_{node} and T_{obj}, respectively. For the WSOU algorithm, in addition to the required space of the C*d*SQ algorithm, the data structures OADM, RDSL and SOET are needed to maintain the information of the dominance relationships and the road distances of objects. For the query path P_q consisting of n edges, there are $(n+1)$ nodes connecting the edges, and in each node, the three data structures are used to facilitate handling the time-varying information. Let $|OADM|$, $|RDSL|$ and $|SOET|$ be the average sizes of the OADM,

the RDSL and the SOET, respectively. Then, the space requirement for the WSOU algorithm is $O(|T_{edge}| + |T_{node}| + |T_{obj}| + n \times |GWSO| + (n+1) \times (|OADM| + |RDSL| + |SOET|))$. Although the WSOU algorithm slightly sacrifices the space cost compared to the C*d*SQ algorithm, it can greatly improve the performance of processing the continuous within skyline query in dynamic road networks, which has been demonstrated in the above simulations.

6. Conclusions

This paper focuses on efficiently processing the continuous within skyline queries in dynamic road networks, where the attributes of objects, the lengths of edges and the query paths may vary with time (called the time-varying information). To provide real-time processing of the continuous within skyline queries with time-varying information, we design three data structures, the OADM, the RDSL and the SOET, to adequately maintain information of objects and road network. Based on the three data structure, we further develop the within skyline object updating algorithm to quickly determine whether the query result is affected by the time-varying information and rapidly update the new result if necessary. Extensive experiments have been conducted to demonstrate the efficiency of the proposed approaches.

There are several interesting avenues for the future extensions of this work. One important avenue is to cope with other variations of the skyline queries, such as the continuous k nearest neighbor queries [8], in dynamic road networks with time-varying information. Another extension is to extend the within skyline object updating algorithm to be suitable for the highly dynamic environment, in which all objects move as time passes (implying that the object locations change with time). An important extension is to study the possibility of applying the within skyline object updating algorithm to distributed environments, such as the sensor networks [28–30].

Acknowledgments: This work was supported by the Ministry of Science and Technology of Taiwan under Grant MOST 105-2119-M-022-002 and Grant MOST 104-2119-M-022-001.

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

References

1. Benetis, R.; Jensen, C.S.; Karciauskas, G.; Saltenis, S. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. *VLDB J.* **2006**, *15*, 229–249.
2. Huang, Y.K.; Kuo, W.H.; Lee, C.; Wang, T.H. Shortest Average-Distance Query on Heterogeneous Neighboring Objects. In Proceedings of the International Conference on IDEAS, Yokohoma, Japan, 13–15 July 2015; pp. 116–125.
3. Mokbel, M.F.; Xiong, X.; Aref, W.G. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In Proceedings of the ACM SIGMOD, Paris, France, 13–18 June 2004; pp. 623–634.
4. Tao, Y.; Papadias, D. Time-parameterized queries in spatio-temporal databases. In Proceedings of the ACM SIGMOD, Madison, WI, USA, 2–6 June 2002; pp. 334–345.
5. Borzsonyi, S.; Kossmann, D.; Stocker, K. The skyline operator. In Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, 2–6 April 2001; pp. 421–430.
6. Huang, Z.; Lu, H.; Ooi, B.C.; Tung, A. Continuous Skyline Queries for Moving Objects. *IEEE Trans. Knowl. Data Eng.* **2006**, *18*, 1645–1658.
7. Sharifzadeh, M.; Shahabi, C. The Spatial Skyline Queries. In Proceedings of the International Conference on Very Large Data Bases, Seoul, Korea, 12–15 September 2006; pp. 751–762.
8. Huang, Y.K.; Chang, C.H.; Lee, C. Continuous distance-based skyline queries in road networks. *Inf. Syst.* **2012**, *37*, 611–633.
9. Bentley, J.L.; Kung, H.T.; Schkolnick, M.; Thompson, C.D. On the average number of maxima in a set of vectors and applications. *J. ACM* **1978**, *25*, 536–543.
10. Chomicki, J.; Ciaccia, P.; Meneghetti, N. Skyline Queries, Front and Back. *ACM SIGMOD Rec.* **2013**, *42*, 6–18.
11. Hsueh, Y.L.; Hascoet, T. Caching Support for Skyline Query Processing with Partially Ordered Domains. *IEEE Trans. Knowl. Data Eng.* **2014**, *26*, 2649–2661.

12. Kung, H.T.; Luccio, F.; Preparata, F.P. On finding the maxima of a set of vectors. *J. ACM* **1975**, *22*, 469–476.
13. Mortensen, M.L.; Chester, S.; Assent, I.; Magnani, M. Efficient caching for constrained skyline queries. In Proceedings of the International Conference on Extending Database Technology, Brussels, Belgium, 23–27 March 2015.
14. Cheema, M.A.; Lin, X.; Zhang, W.; Zhang, Y. A Safe Zone Based Approach for Monitoring Moving Skyline Queries. In Proceedings of the International Conference on Extending Database Technology, Genoa, Italy, 18–22 March 2013.
15. Zheng, J.; Chen, J.; Wang, H. Efficient Geometric Pruning Strategies for Continuous Skyline Queries. *ISPRS Int. J. Geo-Inf.* **2017**, *6*, 91.
16. Vu, K.; Zheng, R. Efficient Algorithms for Spatial Skyline Query With Uncertainty. In Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL), Orlando, FL, USA, 5–8 November 2013.
17. Deng, K.; Zhou, X.; Shen, H.T. Multi-source skyline query processing in road networks. In Proceedings of the 23rd International Conference on Data Engineering, Istanbul, Turkey, 11–15 April 2007; pp. 796–805.
18. Zou, L.; Chen, L.; Ozsu, M.T.; Zhao, D. Dynamic Skyline Queries in Large Graphs. In Proceedings of the International Conference on Database Systems for Advanced Applications, Tsukuba, Japan, 1–4 April 2010.
19. Kriegel, H.P.; Renz, M.; Schubert, M. Route Skyline Queries: A Multi-Preference Path Planning Approach. In Proceedings of the International Conference on Data Engineering, Long Beach, CA, USA, 1–6 March 2010.
20. Mouratidis, K.; Lin, Y.; Yiu, M.L. Preference Queries in Large Multi-Cost Transportation Networks. In Proceedings of the International Conference on Data Engineering, Long Beach, CA, USA, 1–6 March 2010.
21. Huang, X.; Jensen, C.S. In-Route Skyline Querying for Location-based Services. In Proceedings of the International Workshop on Web and Wireless Geographical Information Systems, Goyang, Korea, 26–27 November 2004; pp. 120–135.
22. Jang, S.M.; Yoo, J.S. Processing Continuous Skyline Queries in Road Networks. In Proceedings of the International Symposium on Computer Science and its Applications, Hobart, Australia, 13–15 October 2008.
23. TIGER. Available online: http://www.census.gov/geo/www/tiger/ (accessed on 28 April 2017).
24. Brinkhoff, T. A Framework for Generating Network-Based Moving Objects. *GeoInformatica* **2002**, *6*, 153–180.
25. Cheema, M.A.; Zhang, W.; Lin, X.; Zhang, Y.; Li, X. Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks. *VLDB J.* **2012**, *21*, 69–95.
26. Guting, R.H.; de Almeida, V.T.; Ding, Z. Modeling and querying moving objects in networks. *VLDB J.* **2006**, *15*, 165–190.
27. Mouratidis, K.; Yiu, M.L.; Papadias, D.; Mamoulis, N. Continuous Nearest Neighbor Monitoring in Road Networks. In Proceedings of the International Conference on VLDB, Seoul, Korea, 12–15 September 2006.
28. Buonanno, A.; D'Urso, M.; Prisco, G.; Felaco, M.; Meliado, E.F.; Mattei, M.; Palmieri, F.; Ciuonzo, D. Mobile Sensor Networks based on Autonomous Platforms for Homeland Security. In Proceedings of the Tyrrhenian Workshop on Advances in Radar and Remote Sensing, Naples, Italy, 12–14 September 2012.
29. Ciuonzo, D.; Buonanno, A.; D'Urso, M.; Palmieri, F.A. Distributed Classification of Multiple Moving Targets with Binary Wireless Sensor Networks. In Proceedings of the International Conference on Information Fusion, Chicago, IL, USA, 5–8 July 2011.
30. Tsiligkaridis, T.; Sadler, B.M.; Hero, A.O. On Decentralized Estimation with Active Queries. *IEEE Trans. Signal Process.* **2015**, *63*, 2610–2622.