World Wide Web
DOI 10.1007/s11280-017-0444-2

Continuous range-based skyline queries in road networks

Xiaoyi Fu[1] · Xiaoye Miao[2] · Jianliang Xu[1] ·
Yunjun Gao[2]

Received: 16 August 2016 / Revised: 21 November 2016 / Accepted: 7 February 2017

Abstract With the ever-growing popularity of smartphone devices in recent years, skyline queries over spatial Web objects in road networks have received increasing attention. In the literature, various techniques have been developed to tackle skyline queries that take both spatial and non-spatial attributes into consideration. However, the existing solutions only focus on solving point-based queries, where the query location is a spatial point. We observe that in many real-life applications, the user location is often represented by a spatial range. Thus, in this paper, we study a new problem of range-based skyline queries (CRSQs) in road networks. Two efficient algorithms named landmark-based (LBA) and index-based (IBA) algorithms are proposed. We also present incremental versions of LBA and IBA to handle continuous range-based skyline queries over moving objects. Extensive experiments using real road network datasets demonstrate the effectiveness and efficiency of our proposed algorithms.

Keywords Skyline queries · Continuous skyline queries · Road network · Location-based services

✉ Xiaoyi Fu
xiaoyifu@comp.hkbu.edu.hk

Xiaoye Miao
miaoxy@zju.edu.cn

Jianliang Xu
xujl@comp.hkbu.edu.hk

Yunjun Gao
gaoyj@zju.edu.cn

1 Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong

2 College of Computer Science, Zhejiang University, 310000 Hangzhou, China

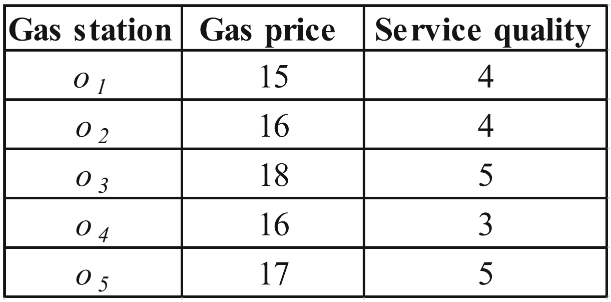
1 Introduction

Given a set of multidimensional spatial Web objects (e.g., restaurants and hotels), a skyline query retrieves the spatial objects that are not dominated by any other object. Specifically, an object p dominates another object p', if p is no worse than p' in each dimension and is better than p' in at least one dimension. The objects retrieved by the skyline query are termed as *skyline objects*. Figure 1 shows an example of a skyline query, where the spatial objects are the gas stations with two properties: gas price and service quality (note that a higher service quality and a lower price are desired). Gas stations o_1 and o_5 are the skyline objects among the five gas stations. The others are dominated by o_1 and o_5 in terms of gas price and service quality. Skyline queries are useful in many real-life applications, such as decision support systems, location-based services, and personalized information search services. Consequently, intensive attention has been paid to the research of skyline queries (e.g., [1, 4, 7, 8, 12, 19, 22]).

More recently, several studies [16, 17, 25] have investigated how to process skyline queries over data objects 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. Nevertheless, the existing works simply use an exact point as the input of the query location. In this paper, we relax this constraint and assume that the query location can be represented by a spatial range. We study a new problem of range-based skyline queries (RSQ) in road networks. Compared to point-based skyline queries, range-based skyline queries in road networks might be more applicable in real life for the following reasons:

- The user wants to know the skyline result with respect to a spatial region (e.g., a school campus or a residential district).
- Because of the limited precision of smart phone devices, only an approximate location of the query issuer is available.
- To protect personal privacy, the user blurs his/her exact location into a spatial region before issuing the query.

Continue with the example in Figure 1. Suppose that the five gas stations are located in a road network shown in Figure 2, where the shadow rectangle R represents the location range of the query issuer who drives a car. In this scenario, the road distance from each gas station to the query point in R (i.e., the distance along the shortest path between them) represents the value of spatial attribute. This spatial attribute and the two non-spatial attributes (i.e., gas price and service quality) are used together to find the skyline objects of the query issuer.



Gas station	Gas price	Service quality
o_1	15	4
o_2	16	4
o_3	18	5
o_4	16	3
o_5	17	5

Figure 1 Example of traditional skyline query

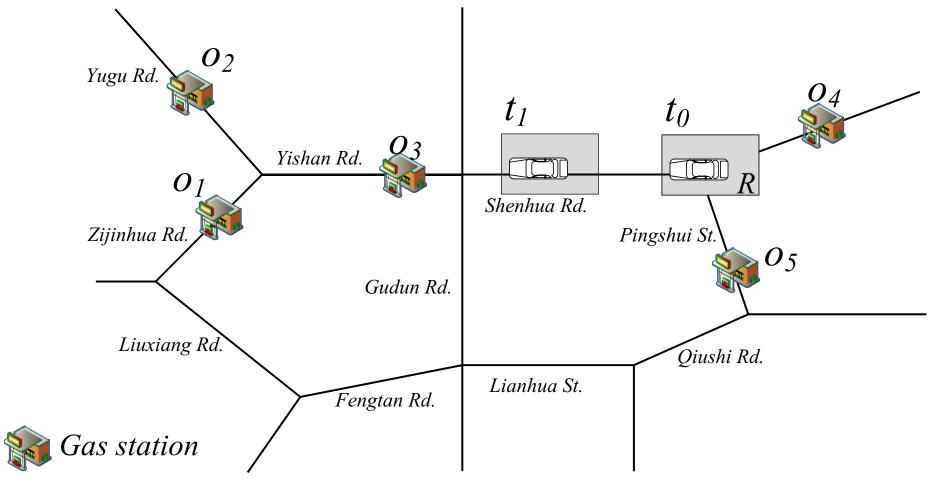


Figure 2 A range-based skyline query in road network

Given a query range R, the service provider should return the union of the skyline objects of all possible query points in R. In this example, the gas station o_4, which is previously dominated by o_1 and o_5 (as shown in Figure 1), is no longer being dominated, as o_4 is closer to R than o_1 and o_5 and is thus added into the set of skyline objects. Hence, the query result becomes $\{o_1, o_4, o_5\}$.

In general, the query issuer *moves with time* in many road network applications, e.g., a driver wants to find a near gas station while driving on a road. For example, the query input location in Figure 2 (i.e., the shadow rectangle R) may move along the road network. Due to the movement of the query location, the query result would inevitably change. If an algorithm for processing snapshot skyline queries is used here, it has to be repeatedly executed to re-evaluate the skyline objects so as to keep producing the new skyline objects. This incurs a tremendous query re-evaluation cost, especially in a highly dynamic environment where the user frequently changes query locations. In this paper, a skyline query with a moving query location is termed as a *continuous skyline query*. We investigate how to efficiently process continuous range-based skyline queries in road networks. This problem is challenging as we may need to update the query result for each timestamp and the number of possible location points in a query range is infinite.

In addition, it would be more challenging when the data objects can move and change their locations frequently. Skyline queries over moving objects have real applications, such as object tracking and monitoring, computer games, and virtual environments [5, 20, 31]. For instance, in a taxi dispatching system, taxis are moving and report their locations to the dispatcher from time to time (e.g., every 5 minutes). To dispatch taxis, the dispatcher needs to consider the taxis' latest locations and their service qualities, then chooses the taxi that is near the customer with good service quality. Another example is to distribute medicines and supplies in the event of natural disasters. The rescue teams move. To select the most suitable rescue team for a rescue area, we need to consider their current locations as well as non-spatial factors such as the amount of supplies they have and the number of wounded they can take. Thus, another objective of this paper is to tackle range-based skyline queries over moving objects. That is, the data objects in the road network may change their locations, and upon such a change we will update the current skyline result.

In summary, our contributions made in this paper include:

- We identify and study a new important problem, that is, continuous range-based skyline queries (CRSQs) in road networks.
- We propose two algorithms, supported by landmark techniques and index structures, to efficiently process CRSQs in road networks.
- We extend the CRSQ problem to moving objects and develop efficient incremental solutions.
- Extensive experiments are conducted to demonstrate the efficiency of our proposed algorithms.

The rest of this paper is organized as follows. Section 2 surveys the related work. Section 3 formalizes our problem definition. We present efficient algorithms to process CRSQs in road networks in Section 4. In Section 5, we extend the algorithms to address range-based skyline queries over moving objects. The proposed algorithms are experimentally evaluated in Section 6. Finally, we conclude the paper with possible directions on future work in Section 7.

2 Related work

Essentially, a continuous range-based skyline query inherits the characteristics of a skyline query, a continuous query, and a range-based query. As such, we review the previous studies on these three queries.

2.1 Skyline queries in road networks

In recent years, processing skyline queries in road networks has received considerable attention. Deng et al. [16] extended the concept of the spatial skyline [28] to road networks and presented the multi-source skyline query (MSQ). An MSQ processes more than one query points at the same time. They proposed three algorithms: the Collaborative Expansion (CE), the Euclidean Distance Constraint (EDC) and the Lower Bound Constraint (LBC) to solve the MSQ problem. However, these algorithms may generate too many candidates and cause unnecessary road distance computation. Hence, Zou et al. [32] proposed the Shared Shortest Path algorithm associated with the Shortest Path Tree to improve performance. The criteria for determining the skyline objects in the above methods are based on the road distances between data objects and query objects. Other studies [17, 25] 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 objects are determined based on these cost values. Huang et al. [9] studied another skyline problem in road networks named in-route query. It assumes that the query issuer's destination and an anticipated route towards the destination are known. Different from these prior point-based studies, in this paper we focus on range-based skyline queries in road networks. Obviously, the existing point-based algorithms are not applicable to our problem as the number of the query points in a spatial range is infinite.

2.2 Continuous skyline queries

There have been some studies [23, 24, 29] considering the problem of processing continuous skyline queries in streaming databases, in which the dimension value of each data object changes over time. As the changing dimension values of each object result in different skyline objects at different time instants, these studies mainly investigated how to efficiently and rapidly update the skyline objects. Zhang et al. [30] further addressed the continuous skyline queries for streaming data with uncertainty. Recently, Huang et al. [11] were the first to take into consideration the distances between data objects during the continuous skyline queries processing. Cheema et al. [3] introduced the idea of *safe zones* for monitoring a moving skyline query, wherein the safe zone of an object is those query positions in which the object is still a skyline object. However, their approach aims at the Euclidean space rather than the 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.

Jang et al. [13] were the first to address the issue of processing continuous skyline queries in road networks. Their idea is to precompute a range R for each data object o such that if the query object is within R, o must be a skyline object. Then, the skyline objects of the query object can be determined based on the pre-computed ranges of all data objects. However, pre-computing all object ranges incurs tremendous overhead, especially for a road network with large size. Hence, Huang et al. [10] presented another type of skyline query named *continuous K-nearest skyline query* (CKNSQ) and proposed an algorithm called CSQ. Assuming the moving route of the query issuer is known in advance, CSQ calculates the skyline objects since the start point and finds the locations on the route where the answer set changes. This avoids processing skyline queries from scratch. [10] also designed a grid index to manage the data objects. However, [10] preferred the skyline objects which were closer to the query point and took a spatial point as the user input, whereas our work focuses on continuous skyline queries in road networks which take a spatial region as the user input.

2.3 Range-based queries

The processing of range-based queries has received notable attention in recent years as the location privacy protection is becoming more and more important. Some previous studies have considered the problem of processing range-based queries. In [27], two complementary techniques, Query Indexing and Velocity Constrained Indexing (VCI), were proposed to efficiently process range-based queries. Kalashnikov et al. [15] presented an in-memory Query-Index technique to evaluate continuous range queries over moving objects. They showed experiment results for several in-memory spatial indexes such as grid index and R*-trees. Cai et al. [2] studied range-monitoring queries and proposed an algorithm, namely *Monitoring Query Management* (MQM), to process such queries. In [2], each mobile object is assigned a resident domain. When an object moves, MQM monitors the object's spatial relationship with its resident domain and the monitoring area inside it.

There also have been studies considering the problem of processing the range-based kNN queries. In [6], Hu et al. proposed an auxiliary index named EXO-tree to handle the range-based kNN queries. Ku et al. studied this problem in spatial networks [18]. In addition, Xu et al. presented a solution to the range-based kNN queries for circular ranges [14]. Lin et al. [21] were the first to study range-based skyline queries and proposed two algorithms: I-SKY and N-SKY. I-SKY is an index-based algorithm. The idea is to compute the skyline scope for each object by their dominance relations before query processing. As the skyline scopes do not depend on the query issuer, [21] uses the MX-CIf quadtree to index the skyline scope and a range-based skyline query can be easily processed by traversing the index tree. While N-SKY is a non-index algorithm, it reduces a range-based skyline query into several segment-based skyline queries. Their approach works for the Euclidean space only. As the road network is very different from the Euclidean space, their approach cannot support our problem and new query processing methods need to be proposed.

3 Problem formulation

In this section, we formalize the continuous range-based skyline query (CRSQ for short) problem and discuss the properties of the CRSQ queries. We consider a data set of objects O. Each object $o \in O$ is associated with one spatial attribute (i.e., location) and several other non-spatial attributes (e.g., gas price and service quality). Table 1 lists the symbols used frequently in the rest of the paper.

Definition 1 (Non-spatial Dominance [21]). Given two objects o and o', if o' is no worse than o in all non-spatial attributes, then we say o' non-spatially dominates o. Formally, it is denoted as $o' \prec o$.

Based on Definition 1, for two objects $o' \prec o$, it is said that, o' is a non-spatial dominator object of o, and o is a non-spatial dominance object of o'. For the sake of simplicity, the set of o's non-spatial dominator objects is denoted as Dom(o), i.e., o is non-spatially dominated by any object in Dom(o). The set of o's non-spatial dominance objects is denoted as Dominance(o). In other words, all the objects in O that are non-spatially dominated by o are included in Dominance(o). Note that, an object o with Dom(o) = Ø must be a non-spatial skyline object, since o is not non-spatially dominated by any other object. Table 2 shows the non-spatial dominance relations of objects o_1 to o_5, whose non-spatial attributes are shown in Figure 1. Objects o_1 and o_5 are not non-spatially dominated by any objects, so Dom(o_1) = Dom(o_5) = Ø. Objects o_2, o_3 and o_4 are non-spatially dominated by $\{o_1\}$, $\{o_5\}$ and $\{o_1, o_2\}$, respectively. As o_3 and o_4 do not non-spatially dominate any other objects, their non-spatial dominance objects sets are empty.

Next we define the concepts of *dominance*, *Point-Based Skyline Query* and *Range-Based Skyline Query*, which are generalized from the definitions for the Euclidean space [21].

Table 1 Symbols and Description

Notation	Description
$G(V, E)$	The road network with a node set V and an edge set E
O	The set objects located in the edge from E
R	The query range of CRSQ
O_R	The set object $o \in O$ located in query range R
SP	The skyline points of pointed based skyline query on road network (PSQ)
Dom(o)	The set of an object o's nonspatial dominator objects
Dominance(o)	The set of an object o's nonspatial dominance objects
FSP	The set objects o whose Dom(o) = Ø
GSP(e)	The global SP of an edge e
dist(p, q)	The shortest path distance from poit p to point q
$H(e)$	The minimum heap containing landmark located in edge e
$SS(o)$	The skyline scope of object o

Table 2 Nonspatial dominance relationship

Object	Dom(o)	Dominance(o)
o_1	Ø	o_2, o_4
o_2	o_1	o_4
o_3	o_5	Ø
o_4	o_1, o_2	Ø
o_5	Ø	o_3

Definition 2 (Dominance in Road Network). Given a query point q and two objects o and o' in a road network, if i) o' non-spatially dominates o, and ii) o' spatially dominates o as well (i.e., o' is closer to q than o in the road network), then we say o' dominates o w.r.t. q in the road network, denoted as $o' \prec_q o$.

Definition 3 (Point-Based Skyline Query in Road Network (PSQ)). Given the object set O, the point-based skyline query in a road network w.r.t. a query point q, denoted as PSQ(q, O), returns a subset of O in which each object is not dominated by any other object in O w.r.t. q.

Definition 4 (Range-Based Skylicne Query in Road Network (RSQ)). Given an object set O and a query range R, the range-based skyline query in a road network, denoted as *RSQ*(R, O)*, returns a set of objects that appear in the skyline set of some point in R. Formally, RSQ*$(R, O) = \{o|\exists q \in R, o \in PSQ(q, O)\}$.

Example 1 Consider a simple scenario where the road network and the objects' non-spatial attributes are shown in Figures 1 and 2, respectively. At time t_0, RSQ(R, O) returns the result set $\{o_1, o_4, o_5\}$ as these objects are the skyline objects of some query point in the road segments covered by the query range R.

Definition 5 (Continuous Range-Based Skyline Query in Road Network (CRSQ)). Given an object set O and a moving query range R, the continuous range-based skyline query in a road network, denoted as CRSQ(R, O), returns the skyline result set of the moving query range R at each time of query range updating.

Example 2 Continue with Example 1. Assume that the query range R moves from Shenhua Rd. to Yishan Rd. and reports its location from time to time. At time t_0, we have the RSQ result as $\{o_1, o_4, o_5\}$, thus CRSQ returns the result set $\{o_1, o_4, o_5\}$ at t_0. At time t_1, CRSQ returns the result set $\{o_1, o_3, o_4, o_5\}$ as it is the RSQ result for R at that time.

Definition 6 (Range-Based Skyline Query in Road Network over Moving Objects (MRSQ)). Given an object set O and a query range R, where the objects in O can move, the range-based skyline query over moving objects in a road network, denoted as MRSQ(q, O), returns the skyline result set of R at each time of object updating.

As formally defined in the previous definitions, RSQ returns a union set of skyline results for every possible query point in the road segments covered by the query range R. For CRSQ, suppose that R is moving and updates its location; then CRSQ continuously returns the query results of RSQ for each query range update. For MRSQ, suppose that the query range R is static and some objects can move on the road network over time (i.e., update their locations at each timestamp); MRSQ returns the current RSQ result upon each object location update. To facilitate query processing, we make the following assumptions:

- A road network is represented as an undirected weighted graph $G(V, E)$. Here V is the set of vertices representing intersections and terminal points and E is the set of edges representing the network edges each connecting two nodes, $E \subseteq V \times V$. The weight of each edge is the length of the edge.
- All the objects are located in the road segments.
- Either the query issuer or the objects may move and update their spatial locations. Note that the problem of monitoring range-based skyline queries where both the query range and objects are moving can be easily addressed based on our proposed algorithms, which will be one of our future work.
- The values of non-spatial attributes remain unchanged throughout the query period.
- Each object has a different location.

Before running into detailed query processing algorithms, we analyze the properties of RSQ problem as follows. These properties were first observed in [21] and are applicable to road networks.

Property 1 If an object o does not have non-spatial dominators, i.e., Dom(o) $= \emptyset$, then o must be an answer of RSQ(q, O).

First, it is obvious that the object is a non-spatial skyline object if Dom(o) $= \emptyset$. Then, for this object o, no other object can dominate it w.r.t. any query point q in the road network where the spatial location is also taken into consideration. Hence, the object o must be an answer of RSQ. Note that, those objects satisfying Property 1 are the fixed skyline objects (denoted as *FSP*) in CRSQ.

Property 2 Given an object set O with the fixed skyline set *FSP*, a query point q, an object $o \in (O - FSP)$ is an answer of PSQ(q, O) if it is closer to q than all its non-spatial dominators in Dom(o). Formally, $o \in$ PSQ(q, O) $\Leftrightarrow$ $p \in$ Dom(o), dist(q, o) $<$ dist(q, p).

Property 2 suggests that we should utilize the set Dom(o) to determine whether o is an answer of RSQ, which minimizes the evaluation cost for each object, thereby boosting the search performance.

Property 3 Given an object set O and a query range R in the road network, any object o inside R must be an answer of RSQ(R, O).

According to Definition 4, RSQ returns the union of PSQ(q, O), where q is a possible query point inside R. When q is located at the same location as the object o, the distance from q to o is 0 and o must be an answer of RSQ(q, O).

Theorem 1 ***(Range-Point Transformation (RPT))*** *Given an object set O and a query range R, if an object $o \in O$ is a result of RSQ(O, R) and o is outside R, o must be a member of the skyline set w.r.t. some intersection point of R and the road network. Formally, it can be expressed as: $o \in$ RSQ(R, O) $\Rightarrow o \in$ PSQ(q, O) $\wedge \exists q \in R \cap E$.*

Proof We prove it by contradiction. Assume that an object o is outside R and $o \in$ RSQ(R, O), while $o \notin$ PSQ(p_1, O) $\cup$ PSQ(p_2, O) $\cup ... \cup$ PSQ(p_n, O), where p_i is an intersection point of R and the edge set E (i.e., $p_i \in R \cap E$). Since $o \in$ RSQ(R, O), according to Definition 4, suppose that o is the skyline object of some query point q inside R, that is

$o \in \text{PSQ}(q, O)$. As $o \notin \text{PSQ}(p_1, O) \cup \text{PSQ}(p_2, O) \cup ... \cup \text{PSQ}(p_n, O)$, there exist objects $o_1, o_2, ..., o_n$ included in $\text{PSQ}(p_i, O)$ such that o_t $(1 \le t \le n)$ dominates o w.r.t. point p_i in the road network. Hence $o_t \in \text{Dom}(o)$ and $\text{dist}(o_t, p_i) < \text{dist}(o, p_i)$. Because every o_t non-spatially dominates o, the necessary condition for o to be a skyline object of q is that $\text{dist}(o, q) < \text{dist}(o_t, q)$ for every o_t. However, the assumption of o being outside R means that there exists at least one object o_t that $\text{dist}(o_t, q) \ge \text{dist}(o, q)$. Thus, o cannot be the skyline object of any point inside R. This leads to a contradiction that $o \in \text{RSQ}(R, O)$. Therefore, the assumption is invalid and the proof completes. □

Recall that according to Definition 5, RSQ returns a union of objects that appear in the skyline result of some point in R. Based on Property 3 and Theorem 1, the result of a range-based skyline query is the union of the objects inside R (denoted as O_R) and the point-based skyline results w.r.t. the intersection points between R and the edges in the road network (i.e., $R \cap E$). Formally,

$$\text{RSQ}(R, O) = \bigcup_{q \in R \cap E} \text{PSQ}(q, O) \cup O_R. \tag{1}$$

It is important to note that, in the rest of this paper, we compute the result of based on (1), for efficient processing of CRSQ.

4 Continuous Range-based Skyline Queries in Road Networks

In this section, we first introduce a baseline algorithm to process CRSQ. Then we propose a landmark-based algorithm (LBA) and an index-based algorithm (IBA) to tackle CRSQ efficiently. We also analyze the complexities of our proposed algorithms.

4.1 Baseline algorithm

With the three properties of RSQ and RPT transformation in Theorem 1, an intuitive solution to CRSQ (termed as Baseline) is to reduce every RSQ (w.r.t. every query range update of CRSQ) into several point-based skyline queries (PSQ) and one range query (based on (1)).

Algorithm 1 Baseline algorithm

```
Input: the object set $O$ on road network $G(V, E)$, moving query range $R$, the fixed skyline
    object set $FSP$ that is pre-computed (based on $\text{Dom}(o)$)
Output: the set of global CRSQ result $result$
 1: while $R$ is moving do
 2:     $result \leftarrow FSP \cup O_R$ // Property 1 and Property 3
 3:     for each point $q \in R \cap E$ do // Theorem 1
 4:         for each object $o' \in O$ do
 5:             compute $\text{dist}(q, o')$
 6:         for each object $o \in (O - FSP)$ do
 7:             $flag \leftarrow$ true;
 8:             for each object $p \in \text{Dom}(o)$ do
 9:                 if $\text{dist}(q, o) \ge \text{dist}(q, p)$ then
10:                     $flag \leftarrow$ false break
11:             if $flag$ = true then
12:                 $result \leftarrow result \cup o$ // Property 2
13:     return $result$;
```

Algorithm 1 gives the pseudo-code of the Baseline algorithm, which takes object set O on road network $G(V, E)$, moving query range R, and the fixed skyline object set *FSP* as inputs. It is assumed that the *FSP* set based on non-spatial attributes is pre-computed. It outputs the global result set of CRSQ. Specifically, for each update of query range, the Baseline algorithm first sets the result set as the union of the *FSP* set and O_R (the objects located in R) based on Properties 1 and 3 (line 1). Then, the Baseline algorithm looks for the skyline objects for every point in $R \cap E$ (i.e., the intersection points of R and the edges in the road network), according to Theorem 1 (lines 3-12). To be more specific, for each point $q \in R \cap E$, Baseline computes the distance from q to every object $o' \in O$ as the spatial value of the object (lines 4-5). Following Property 2, if o is closer to q than all its non-spatial dominators p in Dom(o), we insert o into the result set (lines 6-12). Finally, Baseline outputs the skyline result of RSQ for the current query range.

4.2 Landmark-based algorithm

The Baseline algorithm deals with every update of query range *independently* by reducing each CRSQ into a group of PSQs and a range query based on Property 3 and Theorem 1. Although it reduces the workload for processing each RSQ, Baseline incurs high update overhead for frequently updated query ranges. This is because Baseline processes each RSQ independently, and hence it has to compute the distance between every object to the updated query range for each update of query range. As an improvement, we propose a landmark-based algorithm (LBA) that computes the set of global skyline objects on an edge (termed as the *GSP* set) and leverages a *landmark* technique to process CRSQ efficiently.

In the sequel, we first explain the two main concepts including the *GSP* set and landmark. To begin with, for an edge e, we define GSP(e) as the union set of the skyline objects in PSQ w.r.t. two end nodes of e and the objects located on the edge e. Formally, GSP(e) = PSQ(n_s, O) ∪ PSQ(n_t, O) ∪ O_e, where n_s and n_t are the two end nodes of e and O_e is the set of objects located on e. Hence, GSP(e) is a superset of PSQ(q, O) if the query point q is located on the edge e, as stated in Lemma 1.

Lemma 1 *Given an object dataset O in a road network $G(V, E)$, for a query point q on an edge $e \in E$ connecting two nodes n_s and n_t, it is confirmed that PSQ(q, O) $\subseteq$ GSP(e) (i.e., PSQ(n_s, O) ∪ PSQ(n_t, O) ∪ O_e).*

Proof We prove it by contradiction. Assume that an object $o \in$ PSQ(q, O) but $o \notin$ PSQ(n_s, O) ∪ PSQ(nt, O) ∪ O_e. As $o \notin$ PSQ(n_s, O) ∪ PSQ(nt, O), there exists an object $o' \in$ PSQ(n_s, O) (and an object $o'' \in$ PSQ(n_t, O)) dominating o in the road network. Hence, $o' \in$ Dom(o) (and $o'' \in$ Dom(o)), and the distance dist(n_s, o') $<$ dist(n_s, o) (and dist(n_t, o'') $<$ dist(n_t, o)). Because both objects o' and o'' non-spatially dominate o, the necessary condition for o to be a skyline object of point q is that dist(q, o) $<$ dist(n_s, o') and dist(q, o) $<$ dist(n_t, o''). However, the assumption of $o \notin O_e$ implies that either dist(q, o) $\geq$ dist(q, o') or dist(q, o) $\geq$ dist(q, o''). Hence, $o \notin$ PSQ(q, O). This contradicts to our assumption that $o \in$ PSQ(q, O). Thus, the assumption is invalid and the proof completes. □

Consequently, the GSP(e) sets for every edge e that contains at least one intersection point of $R \cap E$ provide a candidate result set for each RSQ query. Thus, for any update of query range during CRSQ processing, LBA only needs to find the global RSQ result from the collection of these GSP(e) sets. Note that, the purpose of obtaining the GSP(e) set for an edge e is to greatly reduce the number of objects that need to be considered in finding the result set of RSQ for the current timestamp. This is because the objects not in GSP(e)

Table 3 GSP set

e	GSP(e)	e	GSP(e)
$e_1, 8$	o_1, o_3, o_5	$e_6, 7$	o_1, o_3, o_5
$e_1, 2$	o_1, o_3, o_5	$e_7, 8$	o_1, o_3, o_5
$e_2, 3$	o_1, o_3, o_4, o_5	$e_1, 9$	o_1, o_2, o_3, o_5
$e_3, 4$	o_1, o_4, o_5	$e_2, 10$	o_1, o_3, o_5
$e_4, 5$	o_1, o_4, o_5	$e_3, 11$	o_1, o_4, o_5
$e_5, 6$	o_1, o_3, o_4, o_5	$e_2, 6$	o_1, o_3, o_5

can be pruned immediately. Let $e_{s,\ t}$ denote the edge connecting nodes n_s and n_t. Table 3 shows the GSP sets for the edges in the road network shown in Figure 3.

Next, we introduce the second important concept, *landmark*, used in determining whether the query result changes or not, as defined in Definition 7. In other words, for each update of query range, we update the query result of RSQ only if the query result may change (compared with the query result at the last timestamp).

Definition 7 (Landmark). For two objects o and o' in an object set O, if o non-spatially dominates o', i.e., o ∈ Dom(o'), there are two types of landmarks, denoted as L$_{o-o'}$ *and* R$_{o-o'}$, w.r.t. o and o', satisfying dist($L_{o-o'}$, o) = dist($L_{o-o'}$, o') and dist($R_{o-o'}$, o) = dist($R_{o-o'}$, o'). The difference of the two landmarks is described as follows:

(i) For L$_{o-o'}$ landmark, *before the query point q arrives at the landmark $L_{o-o'}$, o cannot dominate o', since dist(q, o') < dist(q, o). When q passes the landmark $L_{o-o'}$, o dominates o'.*

(ii) For R$_{o-o'}$ landmark, *before q reaches the landmark $R_{o-o'}$, o can dominate o'. Once q passes $R_{o-o'}$, o cannot dominate o' because dist(q, o') < dist(q, o).*

For better understanding of the landmark concept, we give an illustrative example below.

Example 3 Consider the road network and data objects in Figures 1 and 3, respectively. Suppose that the query range R moves from node n_3 to node n_1. Consider o_1 and o_4 in Figure 3, as o_1 non-spatially dominates o_4 (referring to Figure 1), there exists a landmark L$_{o_1-o_4}$. In other words, when the query range has not reached L$_{o_1-o_4}$, o_1 cannot dominate o_4. Once it passes L$_{o_1-o_4}$, o_4 is dominated by o_1. In the same way, as o_5 non-spatially dominates o_3, another landmark R$_{o_5-o_3}$ exists as shown in the figure.

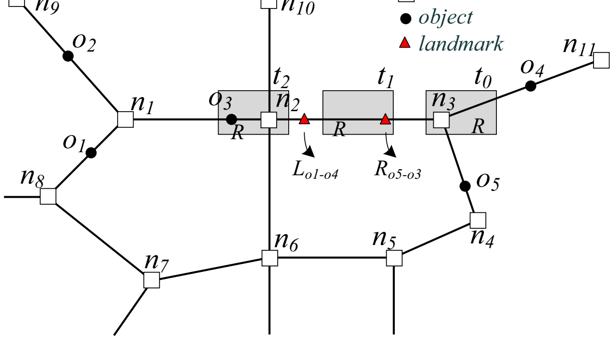


Figure 3 Illustration of the landmark in road network

Note that, given a moving direction of the query range, only one kind of the two landmarks, $\mathrm{L}_{o-o'}$ or $\mathrm{R}_{o-o'}$, exists if $o \in \mathrm{Dom}(o')$.

Algorithm 2 Landmark Examination (LE)

```
Input: landmark $l$, PSQ$(q, O)$
Output: PSQ$(q, O)$ after passing landmark $l$
1: if $l$ is $\mathrm{L}_{o-o'}$ then
2:     if $o' \in$ PSQ$(q,\ O)$ then
3:         return PSQ$(q, O) - \{o'\}$
4: else if $l$ is $\mathrm{R}_{o-o'}$ then
5:     if $o' \notin$ PSQ$(q, O)$ then
6:         if there is no object $o'' \in$ PSQ$(q,\ O)$ dominating $o'$ then
7:             return PSQ$(q, O) + \{o'\}$
8: return PSQ$(q, O)$
```

Based on the landmark concept in Definition 7, we explain how to determine the change of query results using the GSP(e) set. Assume that the landmark of each object pair in GSP(e) has been computed. Algorithm 2 gives the pseudo-code for how to update the result set PSQ(q, O) when the query point q passes a landmark, denoted as l. To be more specific, when the query point q reaches a landmark $\mathrm{L}_{o-o'}$, the object o will dominate the object o' in terms of both the non-spatial and spatial attributes based on Definition 7. If the object o' is currently in PSQ(q, O), then o' is removed from PSQ(q, O) (lines 1-2). For the case where q passes $\mathrm{R}_{o-o'}$, if o' is not in the result set and no object in PSQ(q, O) dominates o', o' will be added into the result set (lines 3-6).

Algorithm 3 Procedure in each update of LBA algorithm

```
Input: the set of objects $O$ on road network $G(V, E)$, query range $R$, last query result $Res_last$
Output: the set of global CRSQ result $result$
1: for each point $q \in R \cap E$ do
2:     if $q$ enters a new edge $e$ connecting two nodes $n_s$ and $n_t$ then
3:         if $H(e) \neq \emptyset$ then //$H(e)$ is a minimum heap of landmarks located in edge $e$
4:             for each landmark$(o,\ o')$ in $H(e)$ do
5:                 if $o \notin$ GSP$(e) \vee o' \notin$ GSP$(e)$ then
6:                     remove landmark$(o,\ o')$ from $H(e)$
7:         for each object $o \in$ GSP$(e)$ do // updat $H(e)$
8:             for each object $o' \in$ GSP$(e) - \{o\} \cap$ Dominance$(o)$ do // update $H(e)$
9:                 compute landmark$(o,\ o')$
10:                if landmark$(o,\ o')$ is on edge $e$ then
11:                    $H(e) \leftarrow H(e) \cup \{$landmark$(o,\ o')\}$
12:                else
13:                    if landmark$(o,\ o')$ is on edge $e'$ (different to $e$) then
14:                        $H(e') \leftarrow H(e') \cup \{$landmark$(o,\ o')\}$
15:        examine the landmarks in $H(e)$ between $n_s$ and $q$ by Algorithm 2
16:        update $result$
17:    else
18:        if $q$ is in the same landmark interval then
19:            $result \leftarrow Res\_last[PSQ(q,\ O)] \cup result$
20:        else
21:            if $q$ enters a new landmark interval then
22:                examine the landmarks in $H(e)$ between last certificated landmark and $q$ by
   Algorithm 2
23:                update $result$
24: return $result \leftarrow result \cup O_R$
```

Now we are ready to present the specific procedure for each update of query range in LBA. Algorithm 3 gives the pseudo-code of the update procedure, where we assume that the sets Dom(o) and Dominance(o) for each object o, and the set GSP(e) for each edge e have been pre-computed. Similar to the Baseline algorithm, for each update of query range, LBA first reduces an RSQ into a range query and several PSQs w.r.t. the intersection points (each denoted as a query point q) of the query range R and the edges of the road network. We also assume that the landmark intervals have been computed in advance to determine the change of PSQ(q, O) for each q. Specifically, a landmark interval is the road segment between two consecutive landmarks. When q moves within a landmark interval, PSQ(q, O) remains unchanged. If q enters a new landmark interval, we need to check the possible update of PSQ(q, O). If q enters a new edge e on which the landmarks have not been computed yet, we need to compute the landmarks on e first. The detailed operations of these three cases for $q \in R \cap E$ are described as follows.

1) q enters a new edge e. Assume that q moves from n_s to n_t. Each edge maintains a minimum heap $H(e)$ for storing the landmarks on it. When q enters e, we first check $H(e)$ and remove the landmarks that not located on e. For each object o in GSP(e), we compute the landmarks related to o and check if they are located on e. Then, the landmarks in $H(e)$ will be considered sequentially to get the result set w.r.t. q.
2) q is within the current landmark interval, the skyline results remain unchanged.
3) q enters a new landmark interval, we certificate the landmark since the last landmark has been checked in last timestamp and get the current query result w.r.t. q.

We give an example to illustrate the query procedure of LBA based on the road network shown in Figure 3.

Example 4 Consider the road network and data objects in Figures 1 and 3, respectively. Assume that the query range R moves from n_3 to n_1. As shown in Figure 3, at time t_0, we assume that all the intersection points of query range R and the road network enter new edges. Thus, we need to compute the landmarks on edges $e_{3,2}$, $e_{3,4}$ and $e_{3,11}$. The locations of landmarks $\mathrm{L}_{o_1-o_4}$ and $\mathrm{R}_{o_5-o_3}$ are shown in Figure 3. Then, we attempt to update PSQ(q, O) of each intersection point from node n_3. The result set at t_0 is $\{o_1, o_4, o_5\}$. Later at time t_1, one of the intersection points (denoted as q_1) passes $\mathrm{R}_{o_5-o_3}$. Hence, we need to examine $\mathrm{R}_{o_5-o_3}$. As the object o_3 does not belong to the current result set of PSQ(q_1, O) and no other object in the current result set of PSQ(q_1, O) dominates o_3, according to Algorithm 2, o_3 is now added into PSQ(q_1, O) and the new result set becomes $\{o_1, o_3, o_4, o_5\}$. At time t_2, three of the intersection points enter new edges (i.e., $e_{2,1}$, $e_{2,3}$, $e_{2,10}$). Then, we compute the landmarks on these three edges. Since there is no landmark located on them, the skyline results of the three intersection points are kept the same as PSQ(n_2, O) (i.e., $\{o_1, o_3, o_5\}$). On the other hand, the right most intersection (denoted as q_2) passes $\mathrm{R}_{o_5-o_3}$ and $\mathrm{L}_{o_1-o_4}$ and we need to examine these two landmarks. By Algorithm 2, o_4 is removed from PSQ(q_2, O) and PSQ(q_2, O) becomes $\{o_1, o_3, o_5\}$. After combining the results of the four intersection points, the new result set at t_2 is $\{o_1, o_3, o_5\}$.

4.3 Index-based algorithm

LBA uses landmarks which accelerate the processing of CRSQ. However, the overhead of computing landmarks may be very high if the query range moves quickly. To avoid the cost of landmark computation in LBA, in this section, we introduce an alternative index-based algorithm (IBA), using *skyline scope index*, for processing CRSQ efficiently.

We first introduce the concept of skyline scope for each object o in a road network. According to Property 1 and Property 2, if Dom(o) is empty, then o must be a skyline object of any query location q in the road network and o is included in the set of fixed skyline objects, denoted as *FSP*. Otherwise, if o is closer to q than all the objects in Dom(o), then o must also be one of the skyline objects of q. Thus, we define the skyline scope [21] of o in the road network as a spatial region in which for any query location q, o is closer to q than any of its dominators, as formalized in Definition 8.

Definition 8 (Skyline Scope in Road Networks). For an object $o \in O$, its skyline scope in a road network $G(V, E)$, denoted as SS(o), is defined as $\{q \in E| \; o \in \text{PSQ}(q, O)\}$, where $o \in \text{PSQ}(q, O)$ means $\forall o' \in \text{Dom(o)}, \text{dist}(o, q) < \text{dist}(o', q)$.

Clearly, the object o is the skyline object of every query point q in o's skyline scope. We borrow the concept of Voronoi cell [31] in order to help us efficiently compute the skyline scope in the road network for every object $o \notin FSP$. Note that, the skyline scope of any object $o \in FSP$ is the whole road network. This is because, wherever the query location q is located, o is always an answer of RSQ(q, O).

Definition 9 (Voronoi Cell in Road Networks [3]). Given an object set O, the Voronoi cell of object o ∈ O, denoted as $V(o, O)$, satisfies that o is the nearest object to p, for ∀ p ∈ $V(o, O)$.

To reduce the computation cost, we can obtain an object o's skyline scope by computing the Voronoi cell of o with the object set containing o and o's non-spatial dominator object. Formally, SS(o) = V(o, O'), $O' = \{o\} \cup \text{Dom}(o)$.

Example 5 Consider the skyline scope of every object in the road network in Figure 4, based on the non-spatial attributes of the objects shown in Figure 1. The non-spatial dominance relations are shown in Table 2. As can be seen, among the five objects, o_1 and o_5 do not have non-spatial dominator (i.e., Dom(o_1) = Dom(o_5) = ∅); thus their skyline scopes cover the whole road network. On the other hand, the skyline scopes of o_2, o_3 and o_4 are obtained by the Voronoi cells in the road network with object sets $\{o_1, o_2\}$, $\{o_3, o_5\}$ and $\{o_1, o_2, o_4\}$, respectively. As shown in Figure 4, the skyline scopes of o_2, o_3 and o_4 are the road segments covered by violet red, blue and red dashed line segments, respectively.

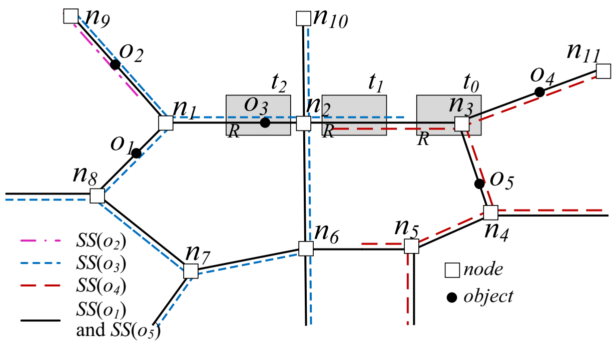


Figure 4 Illustration of skyline scope

As the skyline scopes in the road network are not dependent on the query location, they can be computed and indexed before query processing in order to improve efficiency. To do so, we introduce a new index structure, namely *the skyline scope index* (SSI), which uses a group of B+ trees to index the objects based on their skyline scopes. Specifically, SSI builds a B+ tree for each edge $e \in E$ to index the objects with their skyline scopes intersecting with e. A B+ tree w.r.t. edge e indexes the B+ tree nodes by their distances to the start node of the edge (assume that one of the two end nodes of e has been marked as the start node). Each index node of the B+ tree maintains the information of <*key_value*, *SP*>, where *key_value* stores the distance from this index node to the start node of e and *SP* stores the skyline result of the road segment between this index node and its successor index node.

Algorithm 4 Construction of skyline scope index (SSI)

```
Input: the set of objects $O$ on road network $G(V, E)$
Output: the SSI index w.r.t. $O$ on road network $G(V, E)$
1: for each edge $e$ connecting node $n_s$ and $n_t$ in $G(V, E)$ do
2:     create an empty B+ tree BT$(e, n_s, n_t)$
3: for each object $o \in O$ do
4:     compute its non-spatial dominator objects Dom$(o)$
5:     if Dom$(o) = \emptyset$ then
6:         SS$(o) \leftarrow E$ // cover the whole road network
7:     else
8:         SS$(o) \leftarrow$ V$(o, O')$, $O' = o \cup$ Dom$(o)$
9:     insert $o$ into every BT$(e, n_s, n_t)$ that $e$ intersects with SS$(o)$
10: return SSI index containing all the BT$(e, n_s, n_t)$
```

Algorithm 4 summarizes the procedure of constructing the skyline scope index (SSI). We first construct an empty B+ tree for each edge in the road network. Then for each object o, we need to compute its non-spatial dominator objects Dom(o). If Dom(o) = ∅, that is, o is a member of *FSP*, then o's skyline scope covers the whole road network. If not, we compute o's Voronoi cell with the object set $\{o\} \cup$ Dom(o).

It is obvious that an RSQ can be processed by returning all the objects with their skyline scopes intersecting with the query range R. Thus, the CRSQ processing over an SSI index is straightforward. The IBA algorithm works as follows. First, it reduces the RSQ into a range query and several PSQs w.r.t. the intersection points of $R \cap E$. At each timestamp, IBA computes the intersection points belonging to $R \cap E$. For each intersection point q, IBA traverses the B+ tree that corresponding to the edge on which q is located and gets the skyline objects of q. Finally, the union skyline object sets of all intersection points are returned as the current query result.

To facilitate the understanding of IBA, we give the following example.

Example 6 Consider the road network and data objects in Figures 1 and 4, respectively. At time t_0, the query range R intersects with the skyline scopes of o_1, o_4 and o_5, then the query result at t_0 is $\{o_1, o_4, o_5\}$. Notice that as the skyline scope of o_1 and o_5 is the whole road network, in Figure 4 they are indicated by the black lines, i.e., the original road network. At time t_1, R moves as shown in the figure; the query result becomes $\{o_1, o_3, o_4, o_5\}$ as R intersects the skyline scope of these four objects at t_1. Following a similar method, the query result at time t_2 is $\{o_1, o_3, o_5\}$.

4.4 Complexity analysis

In this section, we analyze the time complexities of our proposed algorithms.

For the Baseline algorithm (Algorithm 1), we use the Dijkstra algorithm to compute network distances with a time complexity of $O(|V|log|V| + |E|)$, where $|V|$ and $|E|$ are the number of nodes and edges in the road network, respectively. For each object in $(O-FSP)$, we compare o with the objects in Dom(o) to find the current query result. The time complexity of this part is $|O-FSP| \cdot |\text{Dom}(o)|$. According to the related work [21], $|\text{Dom}(o)|$ can be estimated as $|O|/2^d$, where $|O|$ is the cardinality of the object set cardinality and d is the dimensionality of non-spatial attributes, and $|O-FSP|$ is $|O|$ in the worst case. Thus, the time complexity of each update of CRSQ is $O(|V|log|V| + |E| + |O|^2/2^d)$.

For LBA, the landmark examination procedure (Algorithm 2) is a constant-time algorithm. As for Algorithm 3, the most expensive operation is the processing when an intersection point q enters a new edge e. The first step is to determine the landmarks of each object in GSP(e) and maintain a minimum heap $H(e)$ to store these landmarks. The time complexity of this step is $O(|\text{GSP}(e)|^2 \cdot \log|H(e)|)$, where $|\text{GSP}(e)|$ and $|H(e)|$ are the cardinality of GSP(e) and $H(e)$, respectively. The second step is to examine the landmarks and to find the skyline objects of q, and the time complexity is $O(|H(e)|)$. Thus, the time complexity for processing one update of CRSQ in the worse case is $O(|\text{GSP}(e)|^2 \cdot \log|H(e)| + |H(e)|)$.

For IBA, the time complexity of the construction of SSI index in Algorithm 4 is $O(|E| + |O| \cdot (|\text{Dom}(o)| + \log N \cdot |E|))$, where $|E|$ is the number of edges in the road network and N is the number of nodes in the B+ tree. As analyzed in the related work [21], $|\text{Dom}(o)|$ can be estimated as $|O|/2^d$, where $|O|$ is the cardinality of the object set and d is the dimensionality of non-spatial attributes. Thus, the time complexity of the construction of SSI index is $O(|E| + |O| \cdot (|O|/2^d + \log N \cdot |E|))$. During the query processing, the main operation is to traverse the B+ tree of each edge that intersects with the query range R. The time complexity for traversing one B+ tree is $O(logcN)$. Suppose that the number of intersection points of query range R and the road network is c; then the time complexity of query processing is $O(c\log N)$.

5 Range-based skyline queries over moving objects

In many skyline query applications, the objects may also move. In light of this, in this section, we extend our query processing techniques to address the RSQ queries over moving objects (MRSQ for short). Note that the Baseline algorithm can be applied directly since it checks all the objects at each timestamp. In the rest of this section, we develop two incremental algorithms, namely LBA-M and IBA-M, based on LBA and IBA algorithms, to solve MRSQ efficiently.

5.1 Landmark-based algorithm for MRSQ

For LBA in MRSQ where the objects are on the move (LBA-M for short), the landmarks may change drastically as the objects move. For each edge e on which the intersection points of $R \cap E$ are located, we need to update the landmarks on e, and then use Algorithm 3 to obtain the query result of each intersection point. Hence, we can get the current MRSQ result by returning the objects inside R and the PSQ query result of each intersection point of $R \cap E$.

Algorithm 5 Update of landmarks

```
Input: a moving object $o$, GSP($e$)
Output: updated landmarks located on $e$
 1: if $o \in$ GSP($e$) before moving then
 2:     if $o \in$ GSP($e$) after moving then
 3:         update landmarks related to $o$ on $e$
 4:     else
 5:         remove landmarks related to $o$ on $e$
 6: else
 7:     if $o \in$ GSP($e$) after moving then
 8:         for each object $o' \in$ GSP($e$) $-$ $\{o\} \cap$ Dominance($o$) do
 9:             insert landmark($o$, $o'$) in $H(e)$
10:     else
11:         mark $o$ as moved
```

It is noteworthy that a moving object o may not affect the query result if o is far away from the query range. Algorithm 5 illustrates how to update the landmarks related to o on edge e efficiently. Once the new location of an object o is reported, if o is a member of the current GSP(e), LBA-M verifies whether o still belongs to GSP(e) at the new location. If yes, LBA-M updates o's landmarks on e, otherwise o's landmarks are removed from $H(e)$. If o was not in GSP(e) and joins GSP(e) after moving, LBA-M computes the landmarks related to o, otherwise o's movement will not influence the current skyline result on e, and there is no need to update o's landmarks.

Example 7 The previous locations of the five objects o_1 through o_5 and the landmarks on the edge e that connects n_1 and n_3 are shown in Figure 3. Suppose R's location will not change after t_2 and o_1 moves as shown in Figure 5. Since $o_1 \in$ GSP(e), the location of landmark $L_{o_1-o_4}$ changes as shown in Figure 5. Then we update the query result of RSQ by examining the landmarks from n_3. Finally, the current query result $\{o_1, o_3, o_4, o_5\}$ is returned.

5.2 Index-based algorithm for MRSQ

When an object o moves, the skyline scope in the road network of o and o's non-spatial dominance objects are all influenced. For MRSQ, at each timestamp, IBA-M (the extended version of IBA) first needs to update the corresponding B+ trees. Then IBA-M computes the intersection points of the query range R and the road network, and for each intersection point, IBA-M searches corresponding B+ tree to get the query result. Finally, the superset

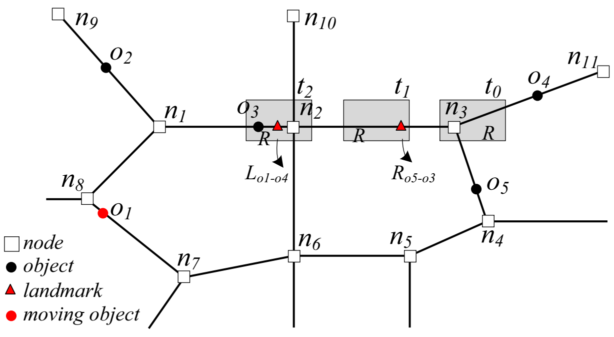


Figure 5 Example of LBA-M when o_1 changes

of query results w.r.t. the intersection points and the objects inside R are returned as the MRSQ query result.

Algorithm 6 Update of skyline scope index

```
Input: moving object $o$, skyline scope index SSI
Output: updated SSI
1: remove SS($o$) from corresponding B+ trees from SSI
2: compute SS($o$) and insert it into corresponding B+ trees in SSI
3: for each object $m \in$ Dominance($o$) do
4:     remove SS($m$) from related BT($e, n_s, n_t$)
5:     recompute SS($m$)
6:     insert SS($m$) into corresponding BT($e, n_s, n_t$)
```

Algorithm 6 gives the procedure of updating the B+ trees of skyline scopes. Specifically, suppose that an object o moves; in order to update the skyline scope index, first we delete o's skyline scope from the corresponding B+ trees and then re-compute o's skyline scope and insert it into the B+ trees. For each object m in Dominance(o), we need to re-compute the skyline scope of m and update the index as well. For illustration, we give an example below.

Example 8 Consider the road network shown in Figure 6. The previous locations of the five objects o_1 through o_5 and their skyline scopes are shown in Figure 4. Suppose that the query range R stops moving after time t_2 and o_1 moves as shown in Figure 6. Since o_1 is a fixed skyline object, its skyline scope is the whole road network and will not change when o_1 moves. The skyline scopes of o_2 and o_4 change as they are non-spatially dominated by o_1, and their new skyline scopes are shown in Figure 6. Suppose that R's location remains unchanged as at t_2, then the current query result becomes $\{o_1, o_3, o_4, o_5\}$ since R now intersects with the skyline scopes of these four objects.

6 Experimental evaluation

In this section, we present a comprehensive experimental evaluation. In what follows, we first present our experimental settings, and then report the experimental results and our findings. All algorithms are implemented in C++, and all experiments are conducted on an Intel Core i5 Duo 3.10GHz PC with 16GB RAM, running Microsoft Windows 7 Professional Edition.

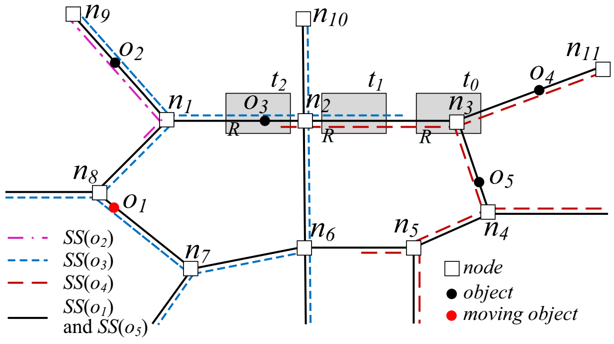


Figure 6 Example of IBA-M when o_1 changes

Table 4 Parameter Settings

Parameter	Range
Object set cardinality $\|O\|$	0.1K, 1K, 20K, 100K, 200K
Number of nonspatial attribute dim	2, 3, 4, 5
Radius of query range r	25, 50, 100, 200, 300
Movement speed of query range(m/s) spd_q	2, 5, 10, 15, 20
Movement speed of object(m/s) spd_o	2, 5, 10, 15, 20

6.1 Experimental settings

In the experiments, we use two real road maps [10] (from http://www.cs.utah.edu/lifeifei/tpq.html: 1) the first road map is the **Oldenburg** Rode Network (a city in Germany) consisting of about 6000 nodes and 7000 edges, 2) the second road map is the **California** Road Network which contains 21,048 nodes and 21,693 edges. There is a set of objects (20,000 objects by default) located in the road network. Each object on the edges of the road network has several non-spatial attributes (2-5 attributes at random) whose values are uniformly distributed in the range [0, 10000]. The non-spatial attributes of the objects are indexed by an R-tree. Table 4 lists the parameter settings, along with their default values and ranges. In our experiments, we measure the average query processing time for 100 updates to evaluate the performance of the proposed algorithms. We use the branch-and-bound skyline (BBS) algorithm [26] to pre-compute the *FSP* set on the R-tree.

In addition, a skyline scope index is built in advance based on the skyline scope concept in the IBA algorithm, as discussed in Section 4.3. Table 5 shows the index size and the index construction time of skyline scope index used in the IBA algorithm for the Oldenburg road network.

The number of buffer misses indicates the I/O cost. In the experiments, all the objects and indexes are stored in the memory except the R-tree that indexes the non-spatial attributes of the objects. Since the R-tree is only used to compute the *FSP* and the dominance relations on the non-spatial attributes between objects in the pre-computation process for all the algorithms under evaluation, their I/O costs are the same. Figure 7 shows the experimental results with $|O|$ varying from 0.1k to 200k. As can be seen, the number of buffer misses increases with $|O|$. This is because the height of R-tree grows when the objects set cardinality becomes larger, and processing queries over the R-tree requires more I/O accesses.

6.2 Results on CRSQ processing

In order to evaluate the impact of different workload settings, i.e., dataset cardinality, the number of non-spatial attributes, the radius of query range and the movement speed of

Table 5 Index Size and Index Construction Time

Object Cardinality	0.1K	1K	20K	100K	200K
Index size	36MB	150MB	251MB	303MB	340MB
Index construction time	1min	4min	2.3h	21.3h	51.1h

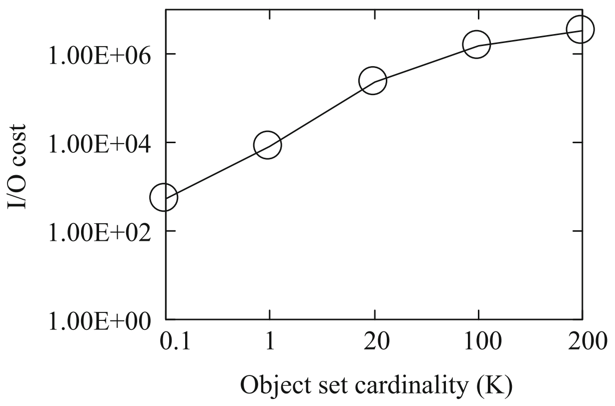


Figure 7 CRSQ I/O cost vs. $|O|$

query range, on our the Baseline, LBA and IBA algorithms, we conduct four sets of experiments on the two real road networks. The experimental results are reported in the following subsections.

6.2.1 Effect of object set cardinality

The first set of experiments evaluates the effect of the number of objects $|O|$ on the performance of the algorithms. Figure 8 shows the CPU time when $|O|$ varies from 0.1K to 200K on the Oldenburg road network (in Figure 8a) and from 0.3K to 200K on the California road network (in Figure 3). As can be seen, IBA is overwhelmingly better than the other two algorithms. As IBA traverse the B+ trees to get the query result at each timestamp, its query cost is much lower than other two algorithms. The efficiency of all three algorithms decreases when $|O|$ increases. For a small $|O|$, LBA is better than Baseline. However, LBA gradually becomes a little worse than Baseline as $|O|$ grows. This is mainly because the

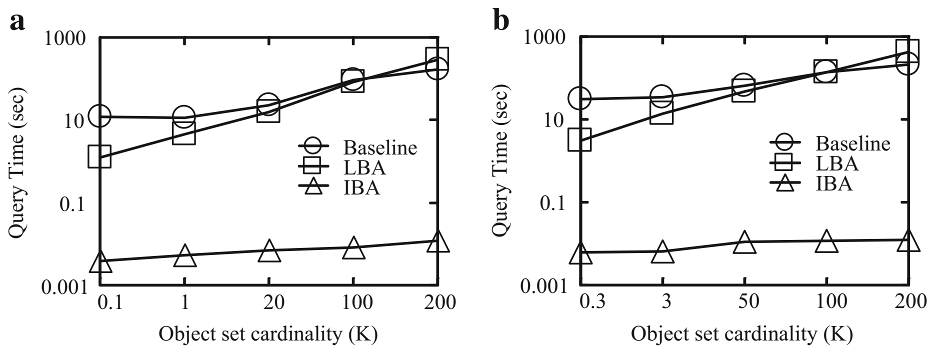


Figure 8 CRSQ cost vs. $|O|$

number of the non-spatial dominance objects of an object increases with $|O|$, which incurs higher overhead of landmark computation in LBA. Consequently, the performance of LBA deteriorates.

6.2.2 *Effect of non-spatial dimensionality*

In the second set of experiments, we investigate the influence of the non-spatial dimensionality dim on the performance of our proposed Baseline, LBA and IBA algorithms. Figure 9 shows the experimental results with dim varying from 2 to 5 on the two road network. It is apparent that IBA is much better than the other two algorithms in all cases as it just needs to perform search operation on the B+ trees for each timestamp. On the other hand, as the number of dimensions dim increases, Baseline and LBA get slightly better. This is because the dominance relations become harder to be satisfied in high dimensions and the number of fixed skyline objects (i.e., the skyline objects only in terms of the non-spatial attributes) grows. As such, for the Baseline algorithm, the number of objects we need to further verify (whether they are query answers) for CRSQ declines. For LBA, the number of non-spatial dominators for each object decreases as dim grows. Thus, the number of possible landmarks reduces. As a result, the computation and examination costs of the landmarks in LBA reduces, which enhances the search performance.

6.2.3 *Effect of query range size*

In the third set of experiments, we investigate the impact of query range size on the efficiency of our presented algorithms. Note that, the size of the query range varies by changing the radius of the query range r from 25 meters to 300 meters, where the center point of the query range is randomly selected from the edges of the road network. The corresponding experimental results on the two road networks are shown in Figure 10. It is observed that IBA performs better than other two algorithms since the main query operation taken for IBA is to search the B+ trees and retrieve the corresponding result, which will not be very high. The performance curve of IBA does not fluctuate very much as the radius r increases.

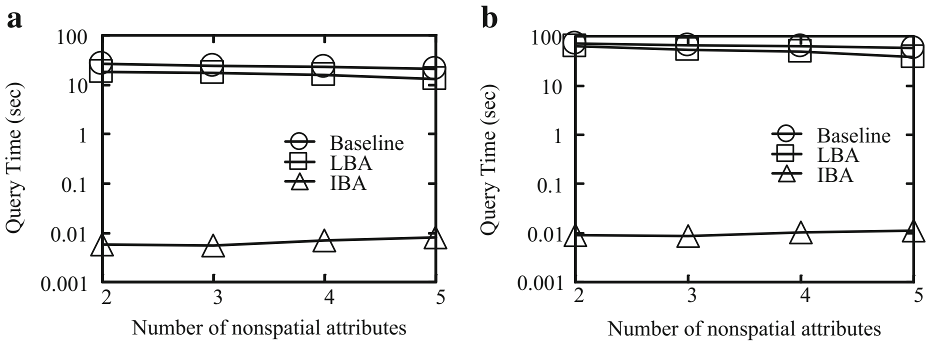


Figure 9 CRSQ cost vs. dim

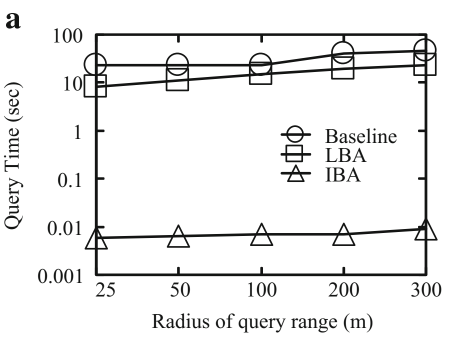
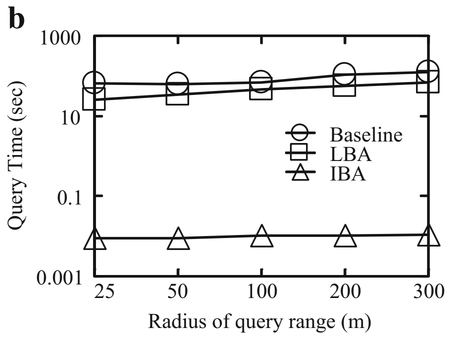


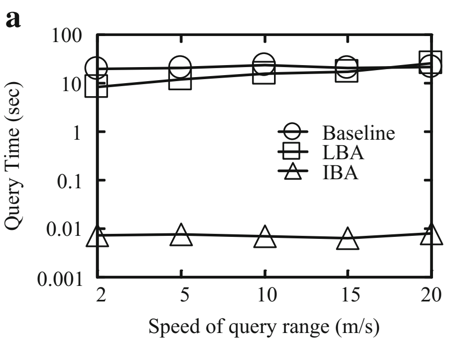
Figure 10 CRSQ cost vs. r



The reason is that the performance of the search operation on the B+ trees depends on the heights of the trees, which do not change when the query range size increases. However, as the radius r increases, the performance of the Baseline algorithm and LBA gets worse. This is because these two algorithms reduce RSQs into PSQs and the number of intersection points grows when r increases.

6.2.4 *Effect of movement speed of query range*

This set of experiments studies the impact of the movement speed of query range spd_q on the performance of our three algorithms. We vary the average movement speed of query range spd_q from 2 to 20 m/s and show the experimental results in Figure 11. It is not a surprise that IBA performs best again as IBA only needs to perform search operations on the B+ trees. The performance of Baseline and IBA does not change very much because the moving of the query range in these two algorithms would not affect the query efficiency. In contrast, for LBA, a higher spd_q implies that the intersection points (query points) would move for a longer network distance and entermore new edges, which leads to more landmark computation for these new edges in LBA. Consequently, when the objects move quickly, LBA becomes worse.



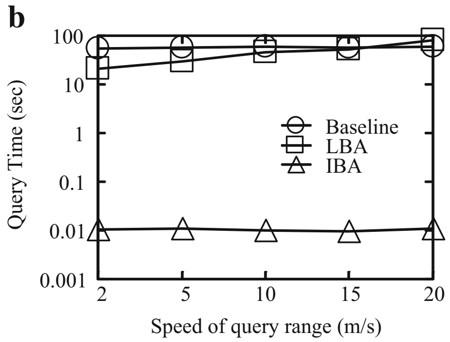


Figure 11 CRSQ cost vs. spd_q

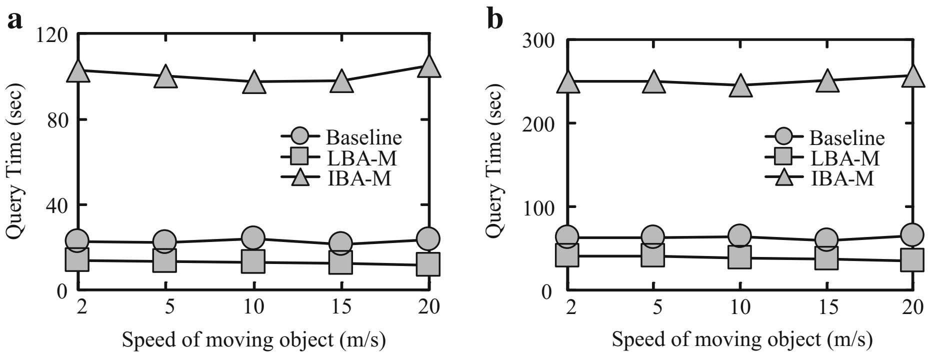


Figure 12 MRSQ cost vs. spd_o

6.3 Results on MRSQ processing

Finally, we investigate the performance of our proposed algorithms for MRSQ where the objects move. We conduct a set of experiments under different speeds of moving objects. Figure 12 shows the result of varying the speed of moving objects spd_o from 2 to 20m/s. We can observe that LBA-M is superior to its competitors. The reason behind it is that when the moving object is far away from the query range, LBA-M does not need to update the landmarks and hence reduces the query processing cost. On the other hand, IBA-M is clearly worse than Baseline and LBA-M. This is because, for each movement, IBA-M needs to update the skyline scopes of the affected objects, which results in high update cost for the IBA-M algorithm. Obviously, LBA-M is more suitable and useful for the situation where the objects move.

7 Conclusions

In this paper, we have studied the continuous range-based skyline query (CRSQ) in road networks, which has a wide range of applications. We have proposed three algorithms, i.e., Baseline, LBA and IBA, to efficiently process CRSQs, where the range-point transformation guarantee, landmark technique, skyline scope index are utilized to boost the search performance. To handle the movement of the objects being queried, we also extend our proposed algorithms to address the range-based skyline query over moving objects (MRSQ). Extensive experiments demonstrate the effectiveness and the efficiency of the proposed algorithms. In particular, IBA performs the best for stationary data objects while LBA-M is superior for moving objects. As for future work, we plan to study range-based skyline queries in a dynamic environment where both the query range and objects can move.

Acknowledgments This work is supported by HK-RGC grants 12244916, 12201615, and 12202414.

References

1. 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 **25**(4), 536–543 (1978)

2. Cai, Y., Hua, K.A., Cao, G.: Processing range-monitoring queries on heterogeneous mobile objects. In: 2004 IEEE international conference on mobile data management, 2004. Proceedings, pp. 27–38. IEEE (2004)
3. Cheema, M.A., Lin, X., Zhang, W., Zhang, Y.: A safe zone based approach for monitoring moving skyline queries. In: Proceedings of the 16th international conference on extending database technology, pp. 275–286. ACM (2013)
4. Gao, Y., Qin, X., Zheng, B., Chen, G.: Efficient reverse top-k boolean spatial keyword queries on road networks. IEEE Trans. Knowl. Data Eng. **27**(5), 1205–1218 (2015)
5. Gedik, B., Wu, K.-L., Yu, P.S., Liu, L.: Processing moving queries over moving objects using motion-adaptive indexes. IEEE Trans. Knowl. Data Eng. **18**(5), 651–668 (2006)
6. Hu, H., Lee, D.L.: Range nearest-neighbor query. IEEE Trans. Knowl. Data Eng. **18**(1), 78–91 (2006)
7. Hu, H., Lee, D.L., Xu, J.: Fast nearest neighbor search on road networks. In: International Conference on Extending Database Technology, pp. 186–203. Springer (2006)
8. Hu, H., Xu J., Lee, D.L.: Pam: an efficient and privacy-aware monitoring framework for continuously moving objects. IEEE Trans. Knowl. Data Eng. **22**(3), 404–419 (2010)
9. Huang, X., Jensen, C.S.: In-route skyline querying for location-based services. In: International Workshop on Web and Wireless Geographical Information Systems, pp. 120–135. Springer (2004)
10. Huang, Y.-K., Chang, C.-H., Lee, C.: Continuous distance-based skyline queries in road networks. Inf. Syst. **37**(7), 611–633 (2012)
11. Huang, Z., Lu, H., Ooi, B.C., Tung, A.K.H.: Continuous skyline queries for moving objects. IEEE Trans. Knowl. Data Eng. **18**(12), 1645–1658 (2006)
12. Jan, C., Parke, G., Jarek, G., Dongming, L.: Skyline with presorting. In: 19th International Conference on Data Engineering, Bangalore, India, p. 717 (2003)
13. Jang S., Yoo, J.: Processing continuous skyline queries in road networks. In: International Symposium on Computer Science and its Applications, pp. 353–356. IEEE (2008)
14. Jianliang, X., Tang, X., Haibo, H., Jing, D.: Privacy-conscious location-based queries in mobile environments. IEEE Trans. Parallel Distrib. Syst. **21**(3), 313–326 (2010)
15. Kalashnikov, D.V., Prabhakar, S., Hambrusch, S.E.: Main memory evaluation of monitoring queries over moving objects. Distributed and Parallel Databases **15**(2), 117–135 (2004)
16. Ke, D., Zhou, X., Tao, H.: Multi-source skyline query processing in road networks. In: 2007 IEEE 23rd international conference on data engineering, pp. 796–805. IEEE (2007)
17. Kriegel, H.-P., Renz, M., Schubert, M.: Route skyline queries: A multi-preference path planning approach. In: 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), pp. 261–272. IEEE (2010)
18. Ku, W.-S., Zimmermann, R., Peng, W.-C., Shroff, S.: Privacy protected query processing on spatial networks. In: 2007 IEEE 23rd International Conference on Data Engineering Workshop, pp. 215–220. IEEE (2007)
19. Kung, H.-T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. J. ACM **22**(4), 469–476 (1975)
20. Lee, M.-W., Hwang, S.-W.: Continuous skylining on volatile moving data. In: 2009 IEEE 25th International Conference on Data Engineering, pp. 1568–1575. IEEE (2009)
21. Lin, X., Xu, J., Hu, H.: Range-based skyline queries in mobile environments. IEEE Trans. Knowl. Data Eng. **25**(4), 835–849 (2013)
22. Lin, X., Xu, J., Hu, H., Lee, W.-C.: Authenticating location-based skyline queries in arbitrary subspaces. IEEE Trans. Knowl. Data Eng. **26**(6), 1479–1493 (2014)
23. Lin, X., Yuan, Y., Wang, W., Lu, H.: Stabbing the sky: Efficient skyline computation over sliding windows. In: 21st International Conference on Data Engineering (ICDE'05), pp. 502–513. IEEE (2005)
24. Morse, M., Patel, J.M., Grosky, W.I.: Efficient continuous skyline computation. Inf. Sci. **177**(17), 3411–3437 (2007)
25. Mouratidis, K., Lin, Y., Yiu, M.L.: Preference queries in large multi-cost transportation networks. In: 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), pp. 533–544. IEEE (2010)
26. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Trans. Database Syst. **30**(1), 41–82 (2005)
27. Prabhakar, S., Xia, Y., Kalashnikov, D.V., Aref, W.G., Hambrusch, S.E.: Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. IEEE Trans. Comput. **51**(10), 1124–1140 (2002)
28. Sharifzadeh, M., Shahabi, C.: The spatial skyline queries. In: Proceedings of the 32nd international conference on Very large data bases, pp. 751–762. VLDB Endowment (2006)
29. Tao, Y., Papadias, D.: Maintaining sliding window skylines on data streams. IEEE Trans. Knowl. Data Eng. **18**(3), 377–391 (2006)

30. Zhang, W., Lin, X., Zhang, Y., Wang, W., Yu, J.X.: Probabilistic skyline operator over sliding windows. In: 2009 IEEE 25th International Conference on Data Engineering, pp. 1060–1071. IEEE (2009)
31. Zhang, Z., Yang, Y., Tung, A.K.H., Papadias, D.: Continuous k-means monitoring over moving objects. IEEE Trans. Knowl. Data Eng. **20**(9), 1205–1216 (2008)
32. Zou, L., Chen, L., Tamer Özsu, M., Zhao, D.: Dynamic skyline queries in large graphs. In: International Conference on Database Systems for Advanced Applications, pp. 62–78. Springer (2010)