Dynamic Skyline Maintaining Strategies for Moving Query Points in Road Networks

Jiping Zheng[1,2,3], Shunqing Jiang[1], Jialiang Chen[1], Wei Yu[1], Siman Zhang[1]

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China
[2] Collaborative Innovation Center of Novel Software Technology and Industrialization, China
[3] Department of Computer Science and Technology, Nanjing University, China
{jzh, jiangshunqing, chenjialiang, echoyu, zhangsiman}@nuaa.edu.cn

Abstract

Skyline query processing in Location-based Services has been investigated extensively in recent years. In this paper, we address the issue of efficient evaluation of Continuous Range Skyline Queries (CRSQ) in road networks where the query points are moving and the interest points are within a certain range. We develop efficient skyline maintaining strategies to answer continuous range skyline queries. First, we propose a novel method named Dynamic Split Points Setting (DSPS) dividing a given path in road networks into several segments. Second, for each segment, we adopt the Progressive Incremental Network Expansion (PINE) technique based on Network Voronoi Diagrams (NVD) to calculate candidates of skyline interest points. After that, when the query point moves, the spilt points are dynamically set by DSPS strategies to ensure that when the query point moves within a segment, skyline points remain unchanged and only need to be updated while moving across the split points. Finally, extensive experiments show that our DSPS strategies are efficient compared with previous approaches.

Keywords: Continuous range skyline queries, Progressive incremental network expansion, Dynamic split points setting, Network voronoi diagrams, Location-based services, Road networks

1 Introduction

Due to the exponentially increasing usage of smartphones and the availability of inexpensive position locators, location-based services (LBS) are increasingly popular where skyline queries based on the current location of the user is an important type of location-based query that can provide useful information and has a wide range of real applications. Skyline query [1-2] is an important operator for applications involving multi-criteria decision making. Given two multi-dimensional points u and v, u dominates v *iff* u is preferred to v in all dimensions, but strictly better than v in at least one dimension. In recent years, the research of skyline queries in road network has received considerable attention [3-9]. Skyline queries in road networks not only take the inherent static attributes of targets into consideration, but also consider spatial attribute (network distances between query point and targets). Figure 1 shows an example where 5 points (p_1-p_5) represent hotels with two inherent static attributes: *price* and *ranking* and a query point q represents a user's location in a road network (shown in Table 1). When issuing a skyline query: *find cheaper, higher ranking and also closer hotels*, the network distance from each hotel to q becomes one of the dimensions. For detailed hotel information as shown in Table 1, the skyline query results are (p_3, p_5) for they are cheaper, higher ranking, and closer than others (p_1, p_2, p_4).

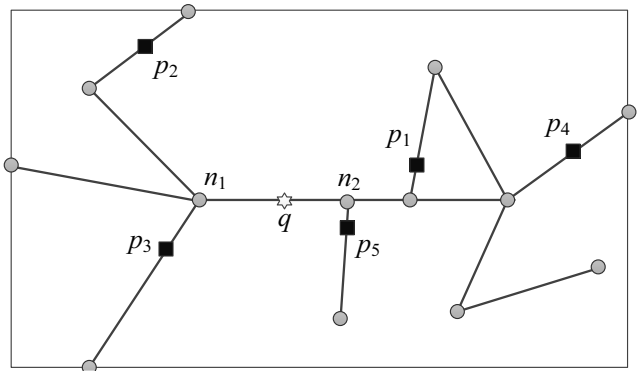


Figure 1. Skyline query in road networks

Table 1. Attributes of interest points

Hotels	Price	Ranking	Distance
p_1	7	2	10
p_2	3	3	11
p_3	3	6	7
p_4	5	4	15
p_5	4	5	4

*Corresponding Author: Jiping Zheng; E-mail: jzh@nuaa.edu.cn
DOI: 10.3966/160792642019092005004

In our *hotel-finding* example, if the query point q is a tourist in a moving car from n_1 to n_2 in the segment n_1n_2, she may only interest in the hotels which network distances to her are within a certain range, e.g. 6 *km*. We can see that hotel p_3 is not a skyline point. This is because the distance between p_3 and q is greater than 6.

There are researches [5, 10-13] on continuous query processing. They all assume the distances between query point and data points are Euclidean distance which are not suitable for real road network applications. Skyline queries in road networks and continuous querying with skylines in road networks are considered in [5-6, 14]. However, in [5], the computation of network distance based on Grid index is time-consuming due to the limitation of Grid index. [6] only considered the query point is static while [14] assumed that the road network distance is city block distance (*i.e.* L_1-distance or Manhattan distance). Also, the determination of split points is not efficient.

In this paper, we address the problem of efficient Continuous Range Skyline Queries (CRSQ) processing in road networks [5]. To evaluate CRSQ on a path, we first determine global candidate skyline points by using Progressive Incremental Network Expansion (PINE*, which is an improved version of PINE technique first proposed by [15] based on Network Voronoi Diagrams (NVDs)). Then we determine the split points using our DSPS strategy and update skyline points according to the location of the query point as well as the locations of split points. In general, the contributions of this paper are summarized as follows:

- We introduce improved Progressive Incremental Network Expansion (PINE*) technique for efficient network distance computation to answer range skyline queries in road networks.
- To evaluate continuous range skyline queries, we propose a Dynamic Split Points Setting (DSPS) strategy which can return skyline points immediately according to the locations of the query point and split points.
- We give experimental evidence that the solutions we proposed in this paper to address CRSQ problem are effective and clear.

The rest of the paper is organized as follows. In Section 2, we discuss related work. Some important pieces of knowledge are discussed in section 3. In Section 4 we present how our CRSQ method and DSPS strategy can be used to answer continuous range skyline queries in road networks. In Section 5, extensive experiments are conducted to show the efficiency of the proposed method. Finally, we conclude our work and point out possible future work in Section 6.

2 Related Work

The skyline operator has been introduced to the database community in 2001 [1], and consequent researches focus on efficient skyline query processing. There have been some work considering the problem of processing skyline queries in road networks, in which the dynamic attribute of road network distance has been involved. [3] first investigated spatial skyline query problems which were first introduced in [16] in road networks and presented multi-source skyline queries which considered several query points at the same time. [4] proposed in-route skyline processing in road networks. [17] computed all linearly non-dominated paths denoted as linear path skyline in bicriteria networks. Processing skyline queries over moving objects is an emerging research topic in recent years. [10] introduced the continuous skyline query problem in the context of LBSs, and an algorithm called *CSQ* been proposed for moving clients. [11] considered that due to privacy concerns and limited precision of localization devices, the input of the user location is often a spatial range, they studied a problem of how to process range-based skyline query in mobile environments. [14] considered the road network distance as Manhattan distance and proposed Manhattan spatial skyline queries. [18] focused on the continuous skyline computation on moving data with an arbitrary number of dynamic queriable dimensions. Some other studies in road network skyline query focus on finding the skyline paths [7-8, 19]. Recently, [20] also considered the skyline trips of multiple POIs categories query problem, pre-computed and stored the distances between POIs and some geographical regions to produce near optimal results. [6] first addressed continuous skyline queries in road networks. The most related work to continuous range skyline queries is Cd_ε-SQ^+ based on Grid index proposed by [5]. Given a path, Cd_ε-SQ^+ broke the path to several segments and each segment had no intersections. Cd_ε-SQ^+ to answer continuous range skyline queries on a segment includes two phases: a global skyline points determination phase, and a result turning point determination phase. However, Cd_ε-SQ^+ is not efficient due to time-consuming network distance computation and determination of split points.

There are also many research based on Voronoi Diagrams. In past several years, calculating road network distances based on Voronoi diagrams is popular for *kNN* queries in road networks. [20] first used Network Voronoi Diagrams (NVD) to solve *k*NN queries in road networks. They proposed a novel approach called VN^3. VN^3 partitions a large network into smaller network Voronoi polygons (NVPs), then pre-compute distances across each NVP. [15] proposed a novel approach, termed Progressive Incremental Network Expansion (PINE). PINE has less disk access and CPU time than VN^3, and PINE is applicable for all kinds of the density and distribution of the interest points. [9] addressed some spatial queries in road networks based on PINE, include *k*NN and C*k*NN

queries to show the effectiveness of PINE technique. [21] applied VN^3 approach to address Continues k nearest neighbor queries (CkNN) in spatial network databases. Also, [22] used Voronoi diagrams to solve the reverse nearest neighbor query problem on spatial networks. In this paper, we take full virtues of PINE and NVD to calculate the network distance to answer continuous range skyline queries. We also adopt Voronoi diagram techniques to compute the network distances for our continuous range skyline queries.

3 Preliminaries

3.1 Problem Definition

In this paper, we consider the query point q as a moving point and the data set P as a set of static interest points in a road network. Each interest point $p_i = <s_1, s_2,..., s_m, d>$ in P has m static non-spatial attributes and a dynamic spatial one d. d is the dynamic distance between the query point and p_i.

Definition 1. (Range Skyline Point) Given a query point q and a range d_r in a road network, a point p ($p \in P$) in the road network is considered to be a range skyline point if it satisfies the following two conditions:

1. $\forall\, p' \in P$, p' does not dominate p;
2. $p_d < d_r$, it means the distance between q and p is less than d_r.

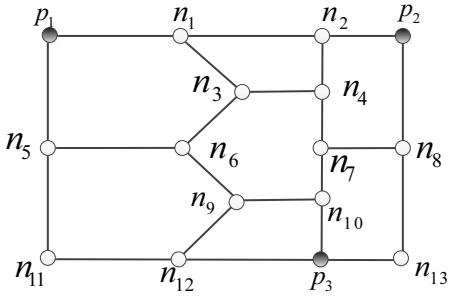
Definition 2. (Continuous Range Skyline Query) CRSQ is defined as: given a set of spatial interest points, a distance range value, and a query point moves on a given path, retrieve Range Skyline Point (*RSP*) set to the query point. The *RSP* is updated according the location of the query point on the path. And at any time, the network distance of each interest point of *RSP* to query point is less than or equal to the range distance.

3.2 Change of Skyline for a Moving Query Point

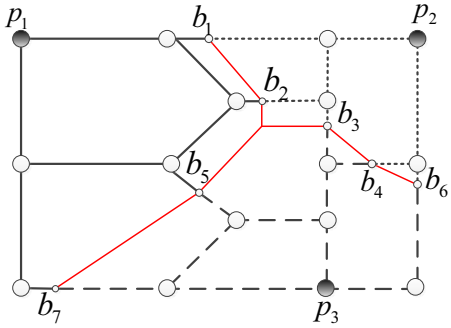
According to the description in previous section, as the query point moves on the path, some interest points may enter or leave the *RSP* for the change of distances. We adopt split points to represent the points to change the skyline results. For CRSQ on a segment, the approach that introduced by [5] computed all split points of the segment at the beginning, as the query point moves, just to update the *RSP* when the split point arrives at. This process needs amount of time to compute split points and stores them. In this paper, our approach (DSPS) adopts a novel approach dynamically setting the split points and updating the *RSP* as the query point moves.

3.3 Data Model

To clarify the problem we study, we formally define a road network as an undirected weighted graph $G(N, P, E)$. It consists of a set of nodes N and a set of edges E which correspond to the road intersections and the road segments, respectively. The lengths of the edges are stored in E. P is a set of interest points. Figure 2(a) depicts a road network model $G(N,P,E)$, where $N=\{n_1, n_2,..., n_{13}\}$ are the intersections of the road, $P=\{p_1, p_2, p_3\}$ are the interest points, and the edges set $E=\{n_1n_2, n_1n_3, ..., p_3n_{13}\}$ represents the segments of the road. Figure 2(b) shows the NVD of the road network in Figure 2(a) which divides the network into *network Voronoi polygons*, namely $NVP(p_i)$. Interest points p_1, p_2 and p_3 are generators. The set $B=\{b_1, b_2, ..., b_7\}$ is called border point set. A border point is the midpoint of a shortest path from one generator to another one, e.g. b_1, $d_N(b_1, p_1) = d_N(b_1, p_2)$.



(a) Road network model



(b) Voronoi diagram of (a)

Figure 2. Network Voronoi diagrams

3.4 Pre-calculated Information of NVDs

After partitioning a road network into NVPs according to the interest points, we need to pre-calculate the information for each NVP such as the network distances between any two border points and the network distances between border points and generated points before we run our PINE* algorithm to obtain the CSP of each segment. For example, for NVP(p_1) in Figure 2(b), we need to pre-calculate the following distances: $d_N(p_1, b_1)$, $d_N(p_1, b_2)$, $d_N(p_1, b_5)$, $d_N(p_1, b_7)$, $d_N(b_1, b_2)$, $d_N(b_1, b_5)$, $d_N(b_1, b_7)$, $d_N(b_2, b_5)$, $d_N(b_2, b_7)$, $d_N(b_5, b_7)$.

If two network Voronoi polygons NVP(p_i) and NVP(p_j) have the same border point, we say that NVP(p_i) is adjacent with NVP(p_j) and vice versa. For

example, in Figure 2(b), NVP(p_1) is adjacent with NVP(p_2) and PINE* algorithm needs the adjacent information of the network Voronoi polygon for each interest point to run. Thus the adjacent information should be pre-calculated and stored which can be obtained when generating the NVDs. We adopt tabular-like data structure in [21] to store these information as illustrated in Figure 3 which corresponds to the NVD in Figure 2(b).

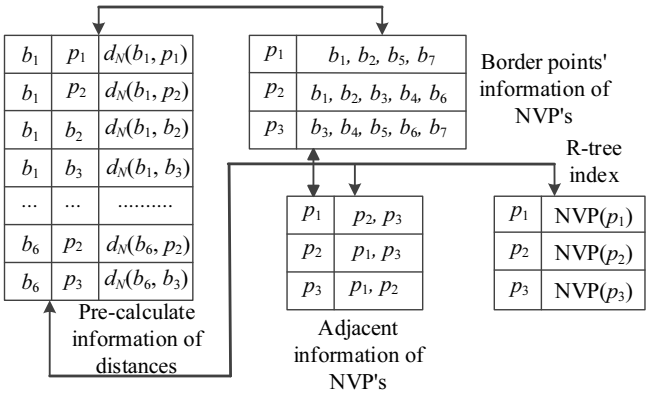


Figure 3. The data structure to store the pre-calculated information of NVDs

4 Proposed CRSQ

Algorithm 1 shows the pseudo codes of our DSPS strategy that used to process CRSQ (D-CRSQ). At the beginning, we divide path P (the given path in CRSQ) into several segments not containing any interest point or road network intersection (line 1). For each segment, we first acquire the candidate skyline points *CSP* by PINE*, then use our dynamic split points setting strategy to implement CRSQ on this segment (lines 2-4). Lemma 1 shows the basic idea of acquiring the *CSP* of a segment.

Lemma 1. Given a segment n_in_j of a road network G, the length of n_in_j is L, a query point q moves from n_i to n_j, distance range value d_r, the set CSP_{ninj} is a collection of interest points, and $d_N(p, n_i)<L+d_r$, $p\in CSP_{ninj}$. Wherever the query point q is located on n_in_j, if $p' \in P \cap p' \notin CSP_{ninj}$, then, $d_N(p', q)>d_r$.

Proof. Because $p'\notin CSP_{ninj}$, so, $d_N(p', n_i) > L+d_r$, we prove $d_N(p', q)>d_r$. There are two situations: 1). If $d_N(p', n_i)\leqslant d_N(p', n_j)$, we have $d_N(p', q) = d_N(p', n_i) + d_N(n_i, q)$, because $d_N(p', n_i)>L+d_r$, so $d_N(p', q)>d_r$. 2). If $d_N(p', n_i)<d_N(p', n_j)$, so, $d_N(p', n_i)=L+d_N(p', n_j)$. Because $d_N(p', n_i) >L+d_r$, we can obtain $d_N(p', n_j)>d_r$. $d_N(p', q)=d_N(p', n_j)+d_N(n_j, q)$. That is, $d_N(p', q)>d_r$.

According to Lemma 1, when we acquire the *CSP* of segment n_in_j, we just find all the interest points $p(p \in P)$ that, $d_N(p, n_i)<L+dr$, L is the length of n_in_j, d_r is the distance range value.

Algorithm 1. DSPS strategy of CRSQ queries (D-CRSQ)

1. Divide Path P_A into several segments each of which does not contain any interest point or road network intersections;
2. for each segment P_{segi} of P_A do
3. Find candidate skyline points of P_{segi} at the starting node of P_{segi};
4. Dynamically acquire split points and update the *RSQ* according to the candidate skyline points of P_{segi} as the query point q moves.

In the next two subsections, we first illustrate how to efficiently acquire the *CSP* of a segment by using PINE* technique, then introduce our proposed method to process dynamic continuous range skyline queries.

4.1 Calculation of Candidate Skyline Point Set of a Segment

Our PINE* approach to acquire candidate skyline points exploits the properties of the Network Voronoi Diagrams (NVDs) and the pre-computed distances for each cell of the NVD. Each candidate skyline interest point p_i of segment n_in_j is in the form of (p_i, $d_N(n_i, p_i)$), and sorted according to $d_N(n_i, p_i)$ in ascending order.

The main process to acquire candidate skyline point set of a segment can be summarized as performing network-segment expansion in the first polygon, and then bulk loading a set of NVPs recursively, until all objects of interest within the expected searching range are retrieved.

Algorithm 2 shows the pseudocodes of acquiring the CSP_{ninj} of the segment n_in_j.

Algorithm 2. Acquire CSP_{ninj} based on PINE*

Input: n_i, d_r, length of segment n_in_j L.
Output: CSP_{ninj}

1. $CSP_{ninj} = \phi$, $H = \phi$;
2. $NVP(p_i)$ is the cell containing n_i in *NVD;*
3. b_{i1}, b_{i2}, ..., b_{ik} is the borders point of $NVP(p_i)$
4. calculate the road network distance from n_i to p_i and b_{i1}, b_{i2}, ..., b_{ik};
5. insert (p_i, $d_N(n_i, p_i)$) into CSP_{ninj};
6. insert (b_{i1}, $d_N(n_i, b_{i1})$),(b_{i2}, $d_N(n_i, b_{i2})$), ..., (b_{in}, $d_N(n_i, b_{in})$) into H;
7. remove first entry (b, $d_N(n_i, b)$) from H;
8. while $d_N(n_i, b) < d_r + L$ do
9. p_k is the other generator point of border point b;
10. $d_N(n_i, p_k)= d_N(n_i, b)+ d_N(b, p_k)$;
11. if $d_N(n_i, p_k) < d_r + L$ then
12. insert (p_k, $d_N(n_i, p_k)$) into CSP_{ninj};
13. for each border point b_j of $NVP(p_k)$ do
14. $d_N(n_i, b_j)= d_N(n_i, b)+ d_N(b, b_j)$;
15. insert (b_j, $d_N(n_i, b_j)$) into H;
16. if H is not empty then
17. remove first entry (b, $d_N(n_i, b)$) from H;
18. return CSP_{ninj}.

4.2 Processing DSPS for CRPQ on a Segment

In this section, we discuss dynamic split points setting (DSPS) strategy for CRSQ queries (D-CRSQ) in segment n_in_j. The result of D-CRSQ for a query point q in segment n_in_j includes the skyline result set and a split point. Table 2 summarizes the mathematical notations frequently used in the rest of this paper.

Table 2. Summary of notations

Notation	Definition
$p_i \prec p_j$	p_i dominates p_j in terms of static and dynamic attributes
$p_i \prec_{static} p_j$	p_i dominates p_j only in terms of static attributes
$p_i \nprec p_j$	p_i cannot dominates p_j
RSP	range skyline points set q
RSP^{up}	increasing group of RSP
RSP^{down}	decreasing group of RSP
CSP_{ninj}	the candidate skyline points of segment n_in_j
CSP^{up}_{ninj}	increasing group of CSP_{ninj}
CSP^{down}_{ninj}	decreasing group of CSP_{ninj}
CSP^{R}_{ninj}	interest points group within a distance range d_r of CSP_{ninj}

After acquiring the CSP_{ninj} of segment n_in_j, for each point p_i in CSP_{ninj}, we need to monitor the changing trend of the network distance between p_i and the query point q. We divide the interest points in CSP_{ninj} into increasing and decreasing groups [23]. The increasing group (decreasing group) of CSP_{ninj} is represented as CSP^{up}_{ninj} (CSP^{down}_{ninj}). If $p \in CSP^{up}_{ninj}$ (CSP^{down}_{ninj}), as the query point q moves from n_i to n_j, the network distance $d_N(p, q)$ is increasing (decreasing). And when we get CSP_{ninj} utilizing the approach that introduced in previous section, we can easily find the shortest path between n_i and each point in CSP_{ninj}. For an interest point $p_i \in CSP_{ninj}$, if the shortest path from p_i to n_i passes through n_j, we insert it into CSP^{up}_{ninj}. Otherwise, we add it into CSP^{down}_{ninj}. Hence, we can directly determine whether a point belongs to increasing or decreasing groups. Similarly, RSP is classified into two sets RSP^{up} and RSP^{down}.

When obtaining the changing trend for each interest point from CSP_{ninj}, we can further delete the interest points that will not belong to the skyline result set when the query point is located on the segment n_in_j from CSP_{ninj} according to Lemma 2 and Theorem 2.

Lemma 2. If $p_i \in CSP^{up}_{ninj}$ and $d_N(p_i, n_i) > d_r$, p_i will never become skyline point.

Proof. As the query point q moves from n_i, $d_N(p_i, q) = d_N(p_i, n_i) + d_N(p_i, q)$. For $d_N(p_i, n_i) > d_r$, $d_N(p_i, q) > d_r$ is hold all the time. Hence, p_i will not become skyline point.

Theorem 2. *The points in CSP^{up}_{ninj} and CSP^{down}_{ninj} being dominated will never become skyline points.*

Proof. We first prove the situation that the points in CSP^{up}_{ninj}. For two points $p_i, p_j \in CSP^{up}_{ninj}$, suppose $p_j \prec p_i$, so $d_N(p_i, q) > d_N(p_j, q)$. As the query point q moves, the distances between p_i, p_j and q have the same changing trend, and the static attributes of them keep the same. That p_i is dominated by p_j all the time. Thus, p_i will never become skyline point. Proof for the points in CSP^{down}_{ninj} is similar.

Based on the above Lemma 2 and Theorem 2, we delete corresponding interesting points from CSP^{up}_{ninj} and CSP^{down}_{ninj}. This step could improve the efficiency of CRSQ queries processing.

Now, we could acquire the initial skyline points (ISP) of q (starting moving at the node n_i). We take skyline domination test in CSP^{R}_{ninj}, where $CSP^{R}_{ninj} \subset CSP_{ninj}$, for each $p \in CSP^{R}_{ninj}$, $dN(p, q) < d_r$, and if $p' \in (CSP_{ninj} - CSP^{R}_{ninj})$, $d_N(p', q) > d_r$. The non-dominated interesting points of CSP^{R}_{ninj} is ISP.

When ISP to q (moving starts at node n_i) from CSP_{ninj} is obtained, we could return RSP=ISP to the user. Next, we apply Dynamic Split Points Setting (DSPS) strategy for CRSQ queries. Before this, we introduce the generation rules for four types of split points.

$exit_p$. Select interest point p_{max}, $p_{max} \in RSP^{up}$ and $d_N(p_{max}, q) > d_N(p', q)$. $\forall p'$, $p' \in (RSP^{up} - p_{max})$, if $d_r - d_N(p_{max}, q) < L$, then p_{max} generates split point: ($exit_{pmax}$, $d_r - d_N(p_{max}, q)$).

in_p. Select interest point p_{min}, $p_{min} \in ((CSP_{ninj} - CSP^{R}_{ninj}) \cap CSP^{down}_{ninj})$ and $d_N(p_{min}, q) < d_N(p', q)$, $\forall p'$, $p' \in ((CSP_{ninj} - CSP^{R}_{ninj}) \cap CSP^{down}_{ninj} - p_{min})$. If $d_N(p_{min}, q) - d_r < L$, p_{min} generates split point: (in_{pmin}, $d_N(p_{min}, q) - d_r$).

$exit_{pj/pi}$. For each point p_i in RSP^{up} and p_j in RSP^{down}, if $p_i \prec_{\text{static}} p_j$, $(d_N(p_j, q) - d_N(p_i, q))/2 < L$. Then, p_i and p_j generate split point: ($exit_{pj/pi}$, $(d_N(p_j, q) - d_N(p_i, q))/2$).

$in_{pi/pj}$. For each point p_i in RSP^{up} and $p_j \in (CSP^{R}_{ninj} \cap CSP^{down}_{ninj})$ but $p_j \notin RSP$, if $p_i \prec p_j$, $(d_N(p_j, q) - d_N(p_i, q))/2 < L$, then, p_i and p_j generate split point: ($in_{pi/pj}$, $(d_N(p_j, q) - d_N(p_i, q))/2$).

We use the split point generation rules mentioned above to find the locations of split points that the query point (start at n_i) will arrive at in the future. We use a priority queue SPQ to store the split points calculated and the split points in SPQ are stored in the form of: (sp, $d_N(sp, q)$) where sp denotes the type of split points with their generating points (e.g. $in_{pi/pj}$, the type is in and the generating points are p_i, p_j) and $d_N(sp, q)$ is the distance between sp and the query point q. At this time, we should calculate the four types of candidate split points respectively according to the updated CSP_{ninj} and RSP. After acquiring each type split point sp, if $d_N(sp, n_i) < L$, then we insert it into split point priority queue SPQ. Note that at the process of determining first type of split points, we only select interest point p_{max}, $p_{max} \in RSP^{up}$ and $d_N(p_{max}, q) > d_N(p', q)$, $\forall p'$, $p' \in (RSP^{up} - p_{max})$ to generate $exit_{pmax}$. This is because the query point q will always arrive $exit_{pmax}$ before $exit_{p'}$. Similarly, when determining second type of split points we only select

interest point p_{min}, $p_{min} \in ((CSP_{ninj} - CSP^{R}_{ninj}) \cap CSP^{down}_{ninj})$ and $d_N(p_{min}, q) < d_N(p', q)$, $\forall p', p' \in ((CSP_{ninj} - CSP^{R}_{ninj}) \cap CSP^{down}_{ninj} - p_{min})$. In this way, we could reduce the calculation of split points as far as possible. After *RSP* and *SPQ* are obtained, we take DSPS strategy for processing CRSQ queries and the details are shown in Algorithm 3.

Algorithm 3. DSPS strategy for processing CRSQ queries on segment n_in_j

Input: d_r, n_in_j, L, CSP_{ninj}, q(located at n_i), *RSP*, *SPQ*
Output: Processing CSPQ queries of segment n_in_j

1. MD=0;
2. while $MD<L$ && *SPQ* is not empty do
3. remove the top entry *sp* from *SPQ*;
4. return (*RSP*, *sp.dis*);
5. $MD = sp.dis$;
6. add *sp.dis* to pairs of CSP^{up}_{ninj} and subtract *sp.dis* from pairs of CSP^{down}_{ninj};
7. subtract *sp.dis* from each split point of *SPQ*;
8. switch *sp* do
9. case $exit_p$
10. delete the interest point p from *RSP* and CSP_{ninj};
11. delete *sp'* which generated by p from *SPQ*;
12. determine $exit_{p'}$ and insert it into *SPQ*;
13. case in_p
14. for each interest point p_i in RSP^{up} do
15. if $p_i \prec p$ then
16. determine split point $in_{pi/p}$;
17. if $d_N(p, q)–d_N(p_i, q)/2<L$ then
18. insert ($in_{pi/p}$, $d_N(p, q)–d_N(p_i, q)/2$) into *SPQ*;
19. if there is no interest point p^*, $P^* \in RSP$ and $p^* \prec p$ then
20. insert the interest point p into *RSP*;
21. determine split point $exit_{pj/p}$ and insert it into SPQ;
22. determine in_{p^*} and insert it into SPQ;
23. case $exit_{pj/pi}$
24. delete the interest point p_i from *RSP* and CSP_{ninj};
25. delete *sp'* which generated by p_i from *SPQ*;
26. case $in_{pi/pj}$
27. if there is no $in_{pk/pj}$ exist in *SPQ* then
28. insert the interest point p_j into *RSP*;
29. for each interest point p in RSP^{up} do
30. if $p_j \prec_{\text{static}} p$ then
31. determine split $exit_{pj/p}$;
32. if $d_N(p_j, q) – d_N(p, q)/2 < L$ then
33. insert ($exit_{pj/p}$, $d_N(p_j, q) – d_N(p, q)/2$) into *SPQ*;
34. end switch
34. end while;

We use the segment n_in_j of Figure 4 to describe the walkthrough of processing continuous range skyline queries where the query point q moves from n_i to n_j. The length of n_in_j is 5, and the value of d_r is 4. The static attributes of each interest point in Figure 6 are shown in Table 3.

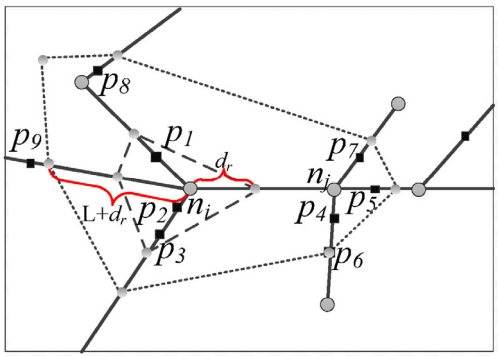


Figure 4. Example of dynamic CRSQ

Table 3. Attributes of interest points

Hotels	Price	Ranking	Distance
p_1	5	4	2.0
p_2	4	4	0.5
p_3	6	3	2.0
p_4	8	6	7.0
p_5	7	2	5.5
p_6	4	3	8.0
p_7	3	5	6.0
p_8	4	4	8.0

First, we get the CSP_{ninj} of segment n_in_j and ensure if each point of CSP_{ninj} is an increasing or decreasing point. The sequence is as follows: {(p_2, 0.5, ↑), (p_1, 2.0, ↑), (p_3, 2.0, ↑), (p_5, 5.5, ↓), (p_7, 6.0, ↓), (p_4, 7.0, ↓), (p_6, 8.0, ↓), (p_8, 8.0, ↑)}.

According to Lemma 2 and Theorem 2, we delete the interest points which will not be the skyline points when the query point q is moving along with the segment n_in_j from CSP_{ninj}. After this step, the CSP_{ninj} are {(p_2, 0.5, ↑), (p_3, 2.0, ↑), (p_5, 5.5, ↓), (p_7, 6.0, ↓), (p_4, 7.0, ↓), (p_6, 8.0, ↓)}.

After that, when q's moving starts at n_i, CSP^{R}_{ninj} = {p_2, p_3}, combining the distance with static attributes of Table 3, we can see that they cannot dominate each other. So, the initial skyline result set is {p_2, p_3}.

Next, we calculate the split points that q will encounter first. In the initial skyline result set: {p_2, p_3}, p_3 has the maximum distance to n_i and $p_3 \in CSP^{up}_{ninj}$. $exit_{p3}$ is generated by p_3 at a distance of 4.0-2.0=2.0 from q (at the beginning, q is located at n_i). In $CSP_{ninj} – CSP^{R}_{ninj}$ ={(p_5, 5.5, ↓), (p_7, 6.0, ↓), (p_4, 7.0, ↓), (p_6, 8.0, ↓)}, p_5 has the minimum distance to n_i and $p_5 \in CSP^{down}_{ninj}$. in_{p5} is generated by p_5, at a distance of 5.5–4.0=1.5 from q. Insert (in_{p5}, 1.5), ($exit_{p3}$, 1.75) into the candidate split point priority queue *SPQ*. In *RSP* and (CSP^{R}_{ninj} –*RSP*), there is no decreasing point. So q will

not encounter the third and fourth types of the split points until arriving at in_{p5}.

When the query point q moves, the candidate skyline point set CSP_{ninj}, split point priority queue SPQ and skyline result set $S(Q)$ are maintained as follows (shown in Table 4, the first column is the distance that q moves).

Table 4. DSPS strategy for the example in Figure 4

Moving	CSP_{ninj}	SPQ	S(Q)
0	{(p_2, 0.5, ↑), (p_3, 2.0, ↑), (p_5, 5.5, ↓), (p_7, 6.0, ↓), (p_4, 7.0, ↓), (p_6, 8.0, ↓)}	{(in_{p5}, 1.5), ($exit_{p3}$, 2.0)}	{p_2, p_3}
1.5	{(p_2, 2.0, ↑), (p_3, 3.5, ↑), (p_5, 4.0, ↓), (p_7, 4.5, ↓), (p_4, 5.5, ↓), (p_6, 6.5, ↓)}	{($in_{p3/p5}$, 0.25), ($exit_{p3}$, 0.5), (in_{p7}, 0.5), ($in_{p2/p5}$, 1.0)}	{p_2, p_3}
0.25	{(p_2, 2.25, ↑), (p_3, 3.75, ↑), (p_5, 3.75, ↓), (p_7, 4.25, ↓), (p_4, 5.25, ↓), (p_6, 6.25, ↓)}	{($exit_{p3}$, 0.25), (in_{p7}, 0.25), ($in_{p2/p5}$, 0.75)}	{p_2, p_3}
0.25	{(p_2, 2.5, ↑), (p_5, 3.5, ↑), (p_7, 4.0, ↓), (p_4, 5.0, ↓), (p_6, 6.0, ↓)}	{($in_{p2/p5}$, 0.5), ($exit_{p7/p2}$, 0.75), (in_{p4}, 1.0), ($exit_{p2}$, 1.5)}	{p_2, p_7}
0.5	{(p_2, 3.0, ↑), (p_5, 3.0, ↑), (p_7, 3.5, ↓), (p_4, 4.5, ↓), (p_6, 5.5, ↓)}	{($exit_{p7/p2}$, 0.25), (in_{p4}, 0.5), ($exit_{p2}$, 1.0)}	{p_2, p_5, p_7}
0.25	{(p_5, 2.75, ↑), (p_7, 3.25, ↓), (p_4, 4.25, ↓), (p_6, 5.25, ↓)}	{(in_{p4}, 0.25)}	{p_5, p_7}
0.25	{(p_5, 2.5, ↑), (p_7, 3.0, ↓), (p_4, 4.0, ↓), (p_6, 5.0, ↓)}	{(in_{p6}, 1.0)}	{p_4, p_5, p_7}
1.0	{(p_5, 1.5, ↑), (p_7, 2.0, ↓), (p_4, 3.0, ↓), (p_6, 4.0, ↓)}	Φ	{p_4, p_5, p_6, p_7}

- Get the first split point in_{p5} from SPQ, and the distance from this split point to n_i is 1.5.
- q moves 1.5 forward and reaches the split point in_{p5}, the skyline result set will change.
- Update the distances of CSP_{ninj} and SPQ.
- Take skyline domination test between p_5 with ISP={p_2, p_3} (the static attributes of them show in Table 3), we can see that $p_5 \prec p_2$, p_3, $S(Q)$ remains unchanged.
- Determine new split points (in_{p7}, 0.5), ($in_{p3/p5}$, 0.25), ($in_{p2/p5}$, 1.0) and insert them into SPQ.
- Repeat above process, as the movement of the query point q, the changes of CSP_{ninj}, SPQ and $S(Q)$ are shown in Table 4.

4.3 Discussion

When we answer CRSQ queries in segment n_in_j, there are two differences between our proposed method and Grid index method in [5] which can improve the efficiency greatly.

The first difference lies in the calculation of candidate skyline points. In [5], in order to acquire candidate skyline points, they need to process range queries at n_i and n_j respectively. Then the union of the query results will be the candidate skyline points. However, our method only needs to process range queries at n_i to obtain candidate skyline points by utilizing PINE* technique and guaranteed by Lemma 1. It is obvious that [5] needs more time to determine the candidate skyline points. Furthermore, we adopt Voronoi diagram instead of Grid index to search candidate skyline points in road networks. We can see that even in a simple example [5, Figure 4], CRSQ query needs to access 5 Grid cells while in our method we only access one NVP cell by R-tree index and extend accessed network nodes to maintain the skyline results. Because all related information of each NVP cell is pre-calculated and stored, the efficiency is improved to a great extent.

The other main difference of [5] and our proposed method is the calculation of split points on a segment. In [4], split points are all calculated beforehand which is time-consuming. Instead, we compute the split points dynamically with the moving of query point by DSPS strategy. Furthermore, our DSPS strategy ensures that when the query point moves at two adjacent split points, the skyline results remain unchanged which also works well to process CRSQ queries by avoiding repeated computation (guaranteed by Lemma 2 and Theorem 2).

5 Experiments

We conduct three sets of experiments for the proposed approaches in this section. Firstly, we study the effect of the CPU cost on the performance of range queries that used to acquire the candidate skyline result set. The second one investigates the performance of the D-CRSQ by measuring the CPU time. Finally, the last set of experiments studies the performance of our approach for larger road networks.

5.1 Calculation of Candidate Skyline Point Set of a Segment

Experiments are implemented in Visual C++ 2010, and all algorithms are executed on a Windows 7 Service Pack 1 PC with 3.3 GHz Intel Core i5-4590

CPU and 4GB memory. We use two different graphs[1] on which skyline queries are performed: (1) the road map of California Road Network that consists of 21,048 nodes and 21,693 edges. (2) the road map of San Francisco that consists of 174,956 vertices and 223,001 edges. The first two sets of experiments conducted on California road network. The moving query point follows a random path, e.g. when a query point moves on an edge with a given speed to the end node, it randomly selects a neighbor edge and continues to move on it.

Table 5 shows the system parameters used in our experiments: interest points (varying from 0.5K to 10K) are generated from the road network edges randomly, the percentage of query range is set from 1% to 10% of the entire space. Each interest point has several static attributes (varying from 2 to 6), whose values satisfy Gaussian distribution (μ=5, σ^2=3). The numbers of segments of each path are from 1 to 20. Bold values are the default used in our experiments.

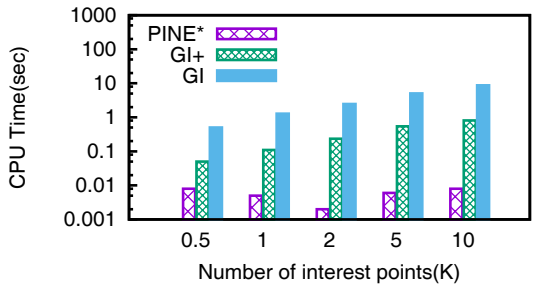
Table 5. Parameters and values

Parameters	Values
Number of interest points(K)	0.5, 1, **2**, 5, 10
Percentage of query range	1%, 2%, **3%**, 5%, 10%
Number of static attributes	2, 3, **4**, 5, 6
Segments of every query path	1, 5, **10**, 15, 20

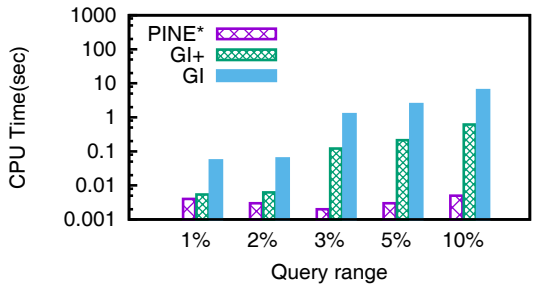
5.2 Performance of Range Query to Acquire the Candidate Skyline Result Set

We first evaluate the efficiency of three approaches for range queries to acquire the candidate skyline interest points of a segment: our PINE* approach, Grid Index based (GI) approach that proposed in [5] for network distance computation of Cd_ε-*SQ* and the further improved version of GI named GI+ of Cd_ε-*SQ*+.

Figure 5(a) shows the average CPU time under different numbers of interest points (varying from 0.5K to 10K), and for each experiment, the query range is set as 3% of the total road map. We can see that PINE* need much less CPU time than GI and GI+. When the number of interest points is 2K, the numbers of network nodes expanded are 28, 69 and 98 for PINE*, GI+ and GI, respectively. For our PINE* approach, from the experiment results, we can find that when the distribution of the interest points become denser, the CPU time of PINE* decreases at the beginning, and then increases as the number of interest points is larger than 2K. This is because when the density of interest points is sparse, the size of each Voronoi polygon (NVP) is large. At the beginning of PINE* approach, it will spend much time to calculate the distances between the query point and the generated points, the border points of the NVP that including query point.



(a) Number of interest points



(b) Query range

Figure 5. Efficiency of range queries

When the number of interest points is more than 2K, the density of NVP is large, the processing time of expanding NVP will increase. When the number of interest points is 2K, the average CPU time is minimum.

Figure 5(b) shows the average CPU time under the different query range (varying from 1% to 10%) in road networks and the number of interest points is set to 2K. It is obvious that as the query range increase, every approach will expand more nodes and interest points. Therefore, the processing time will increase. When the percentage of the query is set to 3%, the numbers of networks nodes expanded are 21, 60 and 97 for PINE*, GI+ and GI, respectively. And the experiment results show that our PINE*-based approach significantly outperforms Grid index based GI and GI+ approaches.

5.3 Performance of *D-CRSQ* Method

In this subsection, we compare the *D-CRSQ* method with the approaches introduced in [9] in terms of the CPU time. Five experiments are conducted to investigate the effects of five important factors on the performance of processing CRSQ queries. These important factors are the number of interest points, the size of query range, the number of static attributes, the length of query path and the distribution of interest points in the road map.

Figure 6(a) shows the average processing time under different number of interest points (varying from 0.5K to 10K). We can see that as the number of the interest points increases, the processing time of *D-CRSQ*

[1] http://www.cs.utah.edu/~lifeifei/SpatialDataset.htm

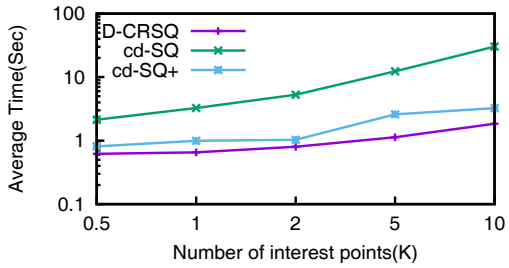
increases slightly. When there are a large number of interest points, *D-CRSQ* is much better than Cd_ε-*SQ* and Cd_ε-*SQ*+ algorithms. When the number of interest points is 2K, the numbers of network nodes expanded are 327, 439 and 645 for D-CRSQ, cd-SQ+ and cd-SQ, respectively. This is because our *D-CRSQ* method reduces expensive shortest path computations greatly. As the increasing number of interest points, our DSPS strategy also decreases the number of calculation of split points to a large extent.

Figure 6(b) shows the average processing time under different ranges (varying from 1% to 10%). When the percentage of query range is 3%, the number of network nodes expanded are 345, 357 and 368 for D-CRSQ, cd-SQ+ and cd-SQ, respectively. As the query range increases, our PINE* technique to answer D-CRSQ queries could acquire *CSP* efficiently. Thus, the query range has little influence on our DSPS strategy.

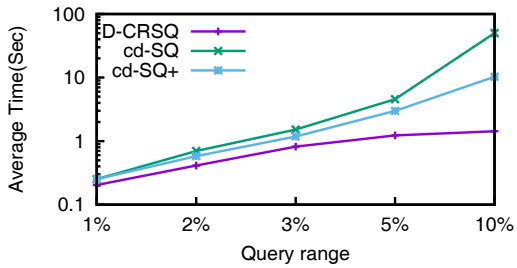
Figure 6(c) shows the average CPU time under static attributes (varying from 2 to 6). When the number of static attributes is 4, the number of network nodes expanded are 301, 325 and 722 for D-CRSQ, cd-SQ+ and cd-SQ, respectively. More attributes of each interest point result in more skyline points so that more dominance tests are performed. *D-CRSQ* method can reduce the number of skyline domination test and the settings of split points. So the efficiency of *D-CRSQ* is better than Cd_ε-*SQ* and Cd_ε-*SQ*+.

Figure 6(d) shows that the average processing time under different query paths. It is obvious that as the query path number increases, the processing time will increase. When the number of segments is 10, the numbers of network nodes expanded are 348, 362 and 674 for D-CRSQ, cd-SQ+ and cd-SQ, respectively. For the same reason mentioned above, our DSPS strategy for CRSQ queries is also much better than Grid index based methods.

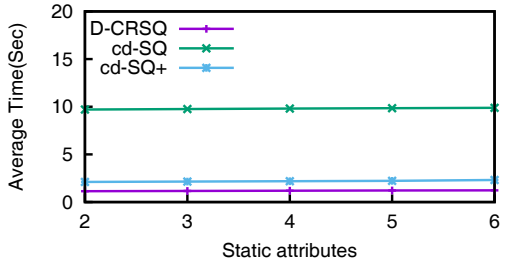
Finally in this subsection, we study how the distribution of interest points in the road map affects the performance of the Cd_ε-*SQ* algorithm, Cd_ε-*SQ*+ algorithm and *D-CRSQ*, by considering two types of interest point distributions. The two interest point distributions are the *uniform* distributions and *Gaussian* distribution. As shown in Figure 6(e), the CPU time of three approaches for the Uniform distribution is slightly lower than that for the Gaussian distribution. For uniform distribution, the numbers of network nodes expanded are 147, 345 and 714 for D-CRSQ, cd-SQ+ and cd-SQ, respectively. While for Gaussian distribution the numbers of network nodes expanded are 213, 378 and 732 for D-CRSQ, cd-SQ+ and cd-SQ, respectively. This is because for the Gaussian distribution, the interest points are closer to each other (comparing to the uniform distribution). Therefore, more NVPs need to be expanded for *D-CRSQ*, and more landmarks need to be examined in the split point determination phase for all approaches, so that it takes more CPU time to process the CRSQ queries.



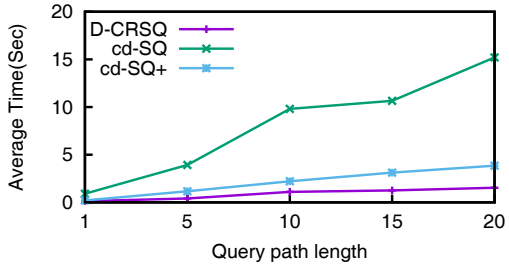
(a) Number of interest points



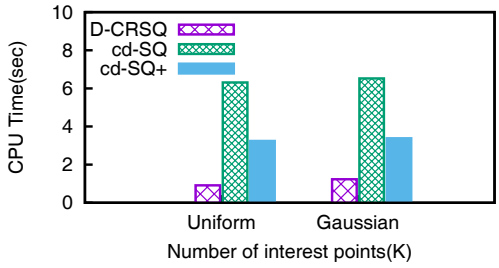
(b) Query range



(c) Static attributes



(d) Query range length



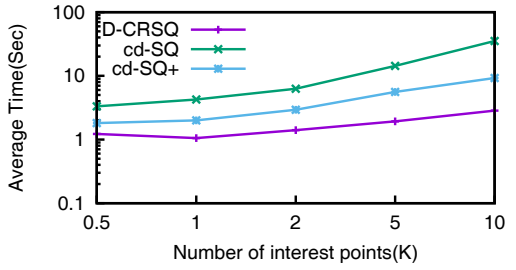
(e) Interest points distribution

Figure 6. CRSQ performance

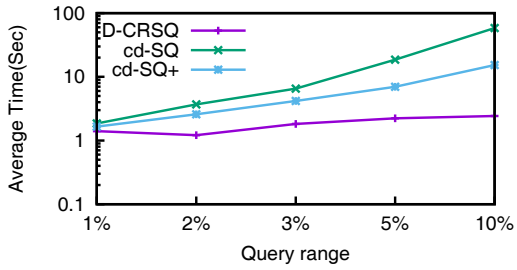
5.4 Performance for Larger Road Networks

In this subsection, we conduct two experiments to study how well the proposed *D-CRSQ* work for a larger road network. The road maps of San Francisco which consists of 174,956 vertices and 223,001 edges.

Figure 7(a) and Figure 7(b) measures the CPU time for our CRSQ applied in the larger road networks, as functions of the number of interest points and query range. When the number of interest points is 2K, the numbers of network nodes expanded are 478, 573 and 658 for D-CRSQ, cd-SQ+ and cd-SQ, respectively. While when the percentage of query range is 3%, the number of network nodes expanded are 438, 514 and 537 for D-CRSQ, cd-SQ+ and cd-SQ, respectively. Compared to the experimental results in Figure 6(a) and Figure 6(b) (which shows the performance for a small road network, California Road), the CPU time overhead for the three approaches is increasing. The reason for this is the road connectivity of the larger road networks is more complicated so that more time is required for computing the road distance.



(a) Number of interest points



(b) Query range

Figure 7. CRSQ performance for the larger road networks

6 Conclusion

In this paper, we address continuous range skyline queries in road networks. Due to the network distance computation is expensive, we utilize PINE* technique based on NVDs to calculate candidate skyline interest points. Further, we propose DSPS strategy to set the split points for efficient processing continuous range skyline queries. Experiments show that our proposed method to answer CRSQ queries is efficient for calculating and maintaining skyline in road networks. Possible future work includes two aspects. The first is to consider skyline paths [7, 8, 19] instead of skyline points in road networks. The idea adopting PINE* to deal with CRSQ queries can be modified and extended to deal with skyline path queries in road networks. The other aspect is to concern dynamic road networks and changing data objects. For real-world applications in road networks, data object attributes and road conditions vary inevitably with time [24, 25], extending our proposed method or developing novel techniques to satisfy the requirements to process CRSQ queries is an important issue in the future.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China under grants U1733112, 61702260. The authors also would like to thank the anonymous reviewers for their helpful suggestions.

References

[1] S. Borzsony, D. Kossmann, K. Stocker, The Skyline Operator, *Proceedings of the 17th International Conference on Data Engineering*, Heidelberg, Germany, 2001, pp. 421-430.

[2] H. Li, S. Jang, J. Yoo, An Efficient Architecture for Parallel Skyline Computation over Large Distributed Datasets, *Journal of Internet Technology*, Vol. 15, No. 4, pp. 577-588, July, 2014.

[3] K. Deng, X. Zhou, H. T. Shen, Multi-source Skyline Query Processing in Road Networks, *Proceedings of the 23th International Conference on Data Engineering*, Istanbul, Turkey, 2007, pp. 796-805.

[4] X. Huang, C. S. Jensen, In-route Skyline Querying for Location-based Services, *Proceedings of the 4th International Conference on Web and Wireless Geographical Information Systems*, Goyang, Korea, 2004, pp. 120-135.

[5] Y.-K. Huang, C.-H. Chang, C. Lee, Continuous Distance-based Skyline Queries in Road Networks, *Information Systems*, Vol. 37, No. 7, pp. 611-633, November, 2012.

[6] S. Jang, J. Yoo, Processing Continuous Skyline Queries in Road Networks, *International Symposium on Computer Science and its Applications*, Hobart, ACT, Australia, 2008, pp. 353-356.

[7] H. P. Kriegel, M. Renz, M. Schubert, Route Skyline Queries: A Multi-preference Path Planning Approach, *Proceedings of the 26th International Conference on Data Engineering*, Long Beach, CA, USA, 2010, pp. 261-272.

[8] K. Mouratidis, Y. Lin, M. L. Yiu, Preference Queries in Large Multi-cost Transportation Networks, *Proceedings of the 26th International Conference on Data Engineering*, Long Beach, CA, USA, 2010, pp. 533-544.

[9] M. Safar, D. El-Amin, D. Taniar, Optimized Skyline Queries

on Road Networks Using Nearest Neighbors, *Personal and Ubiquitous Computing*, Vol. 15, No. 8, pp. 845-856, December, 2011.

[10] Z. Huang, H. Lu, B. C. Ooi, A. K. H. Tung, Continuous Skyline Queries for Moving Objects, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, No. 12, pp. 1645-1658, December, 2006.

[11] X. Lin, J. Xu, H. Hu, Range-based Skyline Queries in Mobile Environments, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 25, No. 4, pp. 835-849, April, 2013.

[12] J. Zheng, J. Chen, H. Wang, Efficient Geometric Pruning Strategies for Continuous Skyline Queries, *International Journal of Geo-Information*, Vol. 6, No. 3, Article 91, March, 2017.

[13] X. Guo, B. Zheng, Y. Ishikawa, Y. Gao, Direction-based Surrounder Queries for Mobile Recommendations, *The VLDB Journal*, Vol. 20, No. 5, pp. 743-766, October, 2011.

[14] W. Son, S.-W. Hwang, H.-K. Ahn, Mssq: Manhattan Spatial Skyline Queries, *Information Systems*, Vol. 40, pp. 67-83, March, 2014.

[15] M. Safar, K Nearest Neighbor Search in Navigation Systems, *Mobile Information Systems*, Vol. 1, No. 3, pp. 207-224, October, 2005.

[16] M. Sharifzadeh, C. Shahabi, The Spatial Skyline Queries, *Proceedings of the 32th International Conference on Very Large Data Bases*, Seoul, Korea, 2006, pp. 751-762.

[17] M. Shekelyan, G. Jossé, M. Schubert, H.-P. Kriegel, Linear Path Skyline Computation in Bicriteria Networks, *Proceedings of the 19th International Conference on Database Systems for Advanced Applications*, Bali, Indonesia, 2014, pp. 173-187.

[18] M. Lee, S. Hwang, Continuous Skylining on Volatile Moving Data, *Proceedings of the 25th International Conference on Data Engineering*, Shanghai, China, 2009, pp. 1568-1575.

[19] Y. Tian, K. C. Lee, W.-C. Lee, Finding Skyline Paths in Road Networks, *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Seattle, Washington, 2009, pp. 444-447.

[20] S. Aljubayrin, Z. He, R. Zhang, Skyline Trips of Multiple POIs Categories, *Proceedings of the 20th International Conference on Database Systems for Advanced Applications*, Hanoi, Vietnam, 2015, pp. 189-206.

[21] M. Kolahdouzan, C. Shahabi, Voronoi-based k Nearest Neighbor Search for Spatial Network Databases, *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, Canada, 2004, pp. 840-851.

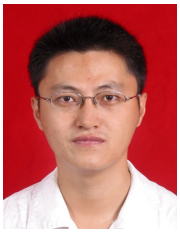
[22] M. Safar, D. Ibrahimi, D. Taniar, Voronoi-based Reverse Nearest Neighbor Query Processing on Spatial Networks, *Multimedia Systems*, Vol. 15, No. 5, pp. 295-308, October, 2009.

[23] M. R. Kolahdouzan, C. Shahabi, Alternative Solutions for Continuous k Nearest Neighbor Queries in Spatial Network Databases, *Geoinformatica*, Vol. 9, No. 4, pp. 321-341, December, 2005.

[24] S. Jiang, J. Zheng, J. Chen, W. Yu, K-th Order Skyline Queries in Bicriteria Networks, *The 18th Asia Pacific Web Conference*, Suzhou, China, 2016, pp. 488-491.

[25] Y. K. Huang, Within Skyline Query Processing in Dynamic Road Networks, *International Journal of Geo-Information*, Vol. 6, No. 5, Article 137, May, 2017.

Biographies



Jiping Zheng is now an associate professor of the College of Computer Science & Technology, Nanjing University of Aeronautics & Astronautics, and a visiting fellow of the Department of Computer Science and Technology, Nanjing University. His research interests include skyline computing, sensor data management, spatial indexes and database security.

Shunqing Jiang received the B.S. degree from Yancheng Teachers University, Yancheng, in 2014 and the M.S. degree from College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, in 2017. His research interests include skyline queries in road networks and spatial data index.

Jialiang Chen received the B.S. and M.S. degrees from Nanjing University of Aeronautics & Astronautics, Nanjing, in 2014, 2017, respectively. His research interests include continuous and spatial skyline queries.

Wei Yu received the B.S. degree from Taiyuan Institute of Technology, Taiyuan, in 2014 and the M.S. degree from College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, in 2017. Her research interests include skyline queries in road networks and Voronoi diagrams.

Siman Zhang received the B.S. and MS degrees from Nanjing University of Aeronautics & Astronautics, Nanjing, in 2015, 2018, respectively. Her research interests include skyline computation in social networks.