Continuous Skyline Queries for Moving Objects in Road Networks

Changyue Shi[1], Xiaolin Qin[1*], Li Wang[1]

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu, China.

* Corresponding author. Tel.: 86+13851642802; email: qinxcs@nuaa.edu.cn.
Manuscript submitted June 20, 2014; accepted August 30, 2014.

Abstract: Skyline queries in road networks have received much attention because of their wide applications in LBS. Due to the increasing demands for services, skyline queries for moving objects in road networks which involve not only static attributes, but also dynamic ones, become more important. This paper studies a problem of how to process the continuous skyline query for moving objects in road networks. We propose an event-based incremental processing algorithm on the strength of tracking dominant relationships between objects with the distances changing and finding the result changing points. Extensive experiments are conducted to evaluate the performance of our proposed algorithm.

Key words: Road networks, moving objects, skyline query, continuous query processing.

1. Introduction

In recent years, various location-based spatial queries have been proposed and researched, such as range queries, nearest neighbor queries [1], skyline queries [2] and so on. Skyline query is a kind of a kind of multi-criterion query. It retrieves the data objects which are not dominated by any other object. The skyline query in road networks involves both non-spatial attributes and the spatial ones, which fuels the demands of location-based applications, like traffic controlling, medical aiding, and advertising recommending.

The existing researches on skyline queries in road networks [3] mainly work on static objects, which have fixed locations in road networks. For example, users can check a skyline query to find out hotels nearby with cheaper price and more stars from many hotels. In this query, the distance between user and hotel is considered as a spatial attribute and the fixed price and stars are considered as non-spatial attributes.

However, dynamic traffic data bring challenges to the existing processing methods on static datasets. Many applications today begin to focus on moving objects in road networks. As an example shown in Fig.1, a man stands by the roadside waiting for a taxi nearby to serve him. In this scenario, we need to consider vehicle performance, driver's experience, user evaluation and the distance from him. In Fig.1, the taxis in solid line boxes $o_1,o_2,...o_5$ show the locations of them at time t_1 and the ones in dotted boxes $o_{1'},o_{2'},...o_{5'}$ are corresponding to their locations 30 seconds later. Considering their locations and non-spatial attributes listed in Table 1, the skyline result set is $\{o_1,o_5\}$ at time t_1, and changes to $\{o_1,o_2,o_3,o_5\}$ 30 seconds later. The distances from user to the taxis constantly change over time since the taxis are moving objects, and the skyline for the taxis is also changing. If we process the continuous skyline query for moving objects with repetitively snapshot queries, it will incur extremely high cost.

Table 1. Non-Spatial Attributes of Taxis

Taxi	o_1	o_2	o_3	o_4	o_5
Performance	9	7	8	6	8
Driver's Experience	5	9	5	4	11
User Evaluation	9.4	8.9	9.2	9.0	9.0

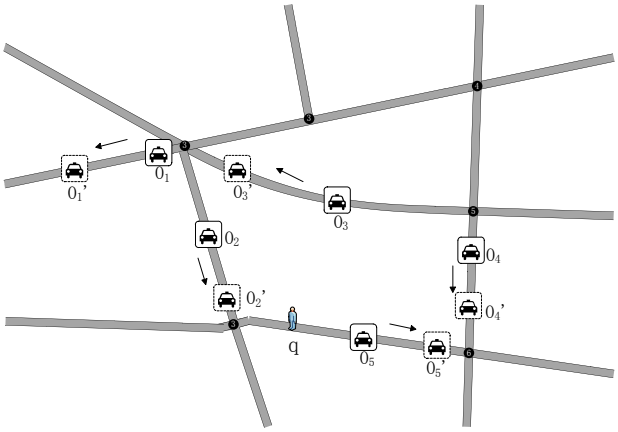


Fig. 1. Taxis' locations in a road network.

To solve the above issue, we study the continuous skyline query for moving objects in road networks and propose an efficient processing algorithm in this paper. We established explicit data models for items in the network environment and analyze the connection between distances of objects and their dominance relationships to find out the time points when the skyline result will change. Then we propose an event-based incremental algorithm to process the continuous skyline query for moving objects in road networks based on our models.

2. Related Work

The skyline query was first introduced by Borzonyi *et al*. as an operator in a relational database [4]. Two algorithms: *Block-Nested-Loops* (BNL) and *Divide- and-Conquer* (D&C) were proposed and laid a foundation of skyline query processing. After that, a lot of scholars devoted to the researches on skyline queries, and successively put forward many variants of skyline query and corresponding efficient algorithms [5]-[7].

Deng *et al*. extended spatial skyline [8] to road network environment and studied the problem of *multi-source skyline query* (MSQ) processing in road networks [3]. They considered the distances from m query points to the object as its m dimensional attributes and proposed three pruning-based algorithms. Md. *et al*. studied a problem of *k dominant skyline for high dimensional space* with a domination power index [9]. Huang *et al*. studied two variants of skyline query: *continuous* d_ε *-skyline query* (Cd_ε-SQ) and *continuous k nearest neighbor skyline query* (CKnn-SQ) [10]. They preferred the skyline points which were closer to the query point. Lin *et al*. studied the range-based skyline query in mobile environments for the reason of privacy protecting and the limited precision of positioning devices [11]. They put forward two algorithms: the index-base I-SKY algorithm and the non-index-based N-SKY algorithm. Mohammad et al. studied the skyline query for spatial objects by utilizing surrounding objects [12]. Cheema et al. studied a problem of monitoring a moving skyline query and proposed a safe zone based approach [13].

The studies mentioned above which focused on skyline queries over static datasets in road networks can't satisfy the increasing demands for location-based services. The collected data involving both static attributes and dynamic ones made the skyline queries for moving objects get more attention. Huang et al. did the first work on continuous skyline query in mobile environment and proposed the corresponding

solution [14]. Tian et al. studied continuous monitoring skyline query over highly dynamic moving objects [15]. They divided the data space into influence region and free region. When some data changed, they updated the influence region and the skyline results dynamically. Later, Chen *et al.* [16] and Ding *et al.* [17] also did some researches on skyline queries for moving objects. However, these existing studies considered objects moving in a free space of 2 or more dimensions, or preset the objects with constant speeds on each dimension. Algorithms on these kinds of scenario assumption can't apply to the limited road networks. The directions and speeds of objects change constantly when they are moving along the roads. So a skyline query processing for such a set of moving objects will be more complex.

3. Skyline for Moving Objects

3.1. Problem Definition

Continuous queries for moving objects have extensive applications in LBS. In order to deal with the result changing, finding the changing points and updating the result will be more effective than snapshot processing in every timestamp. Before we describe the query processing in detail, we give the formal definition of this problem.

In this paper, we consider the query point q as a static point and the data set O as a set of moving objects in a road network. Each object $o_i=<s_1,s_2...s_m...s_n>$ in O has m static non-spatial attributes and n- m dynamic spatial ones. We consider the distance from query point as the only spatial attribute in this paper. For instance, in Fig. 1, the vehicle performance, driver's experience and the user evaluation are static non-spatial attributes because they are invariant. And the distance from query point varying continuously belongs to dynamic attribute.

Definition 1 (road-dominance). Given a query point q and two moving objects o 、 $o' \in O$, we say o road-dominates o' if they met the following two conditions, and denote it as $o \prec_q o'$.

1) For $1 \le i \le m$, $\forall i, o.s_i \le o'.s_i \wedge \exists i, o.s_i < o'.s_i$, it means o non-spatially dominates o', denoted as $o \prec_{ns} o'$;
2) $d(q,o) < d(q,o')$, where $d(q,o)$ and $d(q,o')$ represent the road distances of o and o' from the query point q, respectively.

Skyline query for moving objects in road networks retrieves a subset of objects which are not road-dominated by any other object in O. It can be formally descripted as an object o belongs to the result set if and only if there is no any object $o' \in O$, s.t. $o' \prec_{ns} o$ and $d(q,o') < d(q,o)$.

Because of the locations of moving objects in the road network are changing with time, their distances from the static query point are also changing. A continuous skyline query for moving objects in the road network will retrieve the skyline objects constantly, referred as a MSkyline query for short.

3.2. The Change of Skyline for Moving Objects

According to the description in previous section, we know the MSkyline query involves both static non-spatial attributes and the dynamic distance. Some objects keep in the result set no matter where they are, and the others may enter or leave the result set with the distances changing. In this section, we will analyze the reasons which make skyline change to find the result changing points.

Theorem 1. For a moving object $o \in O$, if o can't be non-spatially dominated by any other object in O, we say it is non-spatial dominator set $Dom_{ns}(o) = \phi$. Then it will keep in the MSkyline result set, which is denoted as *MSK* for short.

Objects satisfying theorem 1. have outstanding non-spatial attributes which make them not road-dominated by others. We say these objects are fixed MSkyline objects, denoted them as permanent MSkyline set MSK_{ns}. Other objects belong to the candidate set $O_{cand} = O - MSK_{ns}$.

Theorem 2. For a moving object $o \in O_{cand}$, whose $Dom_{ns}(o) \neq \phi$, if o is closer to q than any other $o' \in Dom_{ns}(o)$, it will belong to the MSkyline result set MSK.

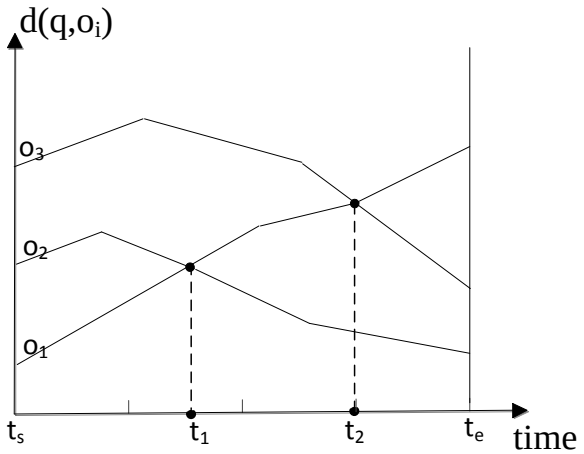


Fig. 2. Example of distance relationships.

Obviously, the distance relationships varying with the objects' moving may cause the *MSK* change. As an example of distance relationships shown in Fig. 2, three objects are moving with a constant distance relationship before time t_1 and the *MSK* remains same during the time interval (t_s, t_1). An intersection (o_1,o_2,t_1) happens at time t_1 when the distance relationship between o_1 and o_2 changes from $d(q,o_1) < d(q,o_2)$ to $d(q,o_2) < d(q,o_1)$. It may cause the object o_1 in *MSK* to leave after time t_1, or the object o_2 not in *MSK* to enter in. We analyze the following three possible cases to determine whether a distance intersection can make the *MSK* change.

Case 1: $o_1 \notin Dom_{ns}(o_2) \wedge o_2 \notin Dom_{ns}(o_1)$

In this case, if there is no non-spatial dominance between o_1 and o_2, there will be no road-dominance either. The distance relationship between o_1 and o_2 will never affect the *MSK*.

Case 2: $o_1 \in Dom_{ns}(o_2) \wedge o_2 \notin Dom_{ns}(o_1)$

In this case, $o_1 \prec_{ns} o_2$ and $d(q,o_1) < d(q,o_2)$ Before time t_1. It means o_1 road-dominates o_2 which makes o_2 not belong to the *MSK*. When it arrives to time t_1, their distance relationship changes to $d(q,o_2) < d(q,o_1)$. It makes o_2 not be road-dominated by o_1 any more. If there is no other object in $Dom_{ns}(o_2)$ road-dominate o_2, o_2 will enter the *MSK* after time t_1.

Case 3: $o_2 \in Dom_{ns}(o_1) \wedge o_1 \notin Dom_{ns}(o_2)$

In this case, the non-spatial dominant relationship between o_1 and o_2 is $o_2 \prec_{ns} o_1$. When it arrives to time t_1, their distance relationship changes to $d(q,o_2) < d(q,o_1)$. It makes o_2 begin to road-dominate o_1. If o_1 belongs to the *MSK* before time t_1, it will leave. Otherwise, it will remain not in after time t_1.

4. An Event-Based Algorithm for MSyline

4.1. Data Model

To clarify the problem we study, we formally define the road network as an undirected weighted graph G(V, E, W). It consists of a set of nodes V 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 W. In addition, each edge also contains the following attributes shown in Table 2.

Table 2. The Attributes of an Edge

Attributes	Descriptions
rid	The identity of the road it belongs to
eid	The edge identity
length	The length of the edge
n_s	The starting node of the edge
n_e	The ending node of the edge

To simplify the problem, we assume the query point as a static point in the road network and represent it as a Trituple q=(qid,eid,dis_s) where qid is the identity of q, eid is the identity of the edge which q resides on, and dis_s is the distance between q and the starting node n_s of the edge. We consider a set of objects moving in the road network. The relevant attributes of moving objects are shown in Table 3.

Table 3. The Attributes of a Moving Object

Attributes	Descriptions
oid	The object identity
eid	The identity of the edge it residing on
dis_s	The distance between the object and the n_s of the edge it residing on
v	The moving speed of it
t_u	The nearest update time
t_{next}	The time it expected to reach the next node

We have the location information (eid,dis_s) and the speed v of the object at update time t_u. A positive v value indicates the object moves with the direction from n_s to n_e on the edge, and a negative v value indicates it moves with the opposite direction. In addition, we use a vector $Dom_{ns}(o)$ to manage the non-spatial dominators for each object o, with a tag Num_Dom_d recording the number of the objects which are closer to q than the o bject o in $Dom_{ns}(o)$.

4.2. Road Network Distance

In order to deal with the continuous skyline query, the road distances between moving objects and the query point need to be constantly calculated. Although the Dijkstra's algorithm can be used to calculate distance between any two points in the network effectively, the constantly calculating with Dijkstra's algorithm will also bring enormous cost. For this reason, we design an optimize distance calculation method in road networks, which can greatly reduce the cost of constantly calculation.

The precomputed shortest distances between every two nodes n_i and n_j in the road network is denoted as $d(n_i,n_j)$ and stored in a distance matrix SD. The value of $d(n_i,n_j)$ in SD can be used to promote the distance calculation between two objects on the edges linked with n_i and n_j.

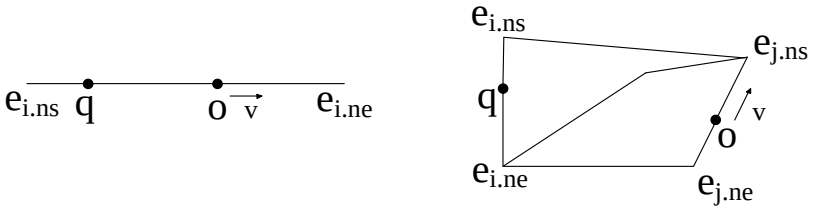


Fig. 3. Two cases of the locations between q and o.

Given a query point q and a moving object o, the distance between them at time t, denoted as $d_t(q,o)$, can be calculated by our method with two cases shown in Fig. 3 based on their locations.

In the first case, they reside on the same edge e_i, and o moves with the speed of v. We can calculate the distance between them by formula (1) as follows. The distance $d_t(q,o)$ obtained by the formula(1) is valid as long as o is moving on e_i, and ensured changing in the range of $[0,e_i.length]$.

$$d_t(q,o)=|q.dis_s-(o.dis_s+o.v\times(t-o.t_u))| \tag{1}$$

In the second case, they reside on different edges e_i and e_j. Since each edge has two nodes, the start node n_s and the end node n_e, the distance $d_t(q,o)$ containing 3 parts can be calculated by formula (2), where $d(q.n_s,o.n_s)$, $d(q.n_s,o.n_e)$, $d(q.n_e,o.n_s)$ and $d(q.n_e,o.n_e)$ have been stored in SD.

$$d_t(q,o)=\min\begin{Bmatrix} d(q.n_s,o.n_s)+q.dis_s+(o.dis_s+o.v\times(t-o.t_u)),\\ d(q.n_s,o.n_e)+q.dis_s+(e_j.length-o.dis_s-o.v\times(t-o.t_u)),\\ d(q.n_e,o.n_s)+(e_i.length-q.dis_s)+(o.dis_s+o.v\times(t-o.t_u)),\\ d(q.n_e,o.n_e)+(e_i.length-q.dis_s)+(e_j.length-o.dis_s-o.v\times(t-o.t_u)) \end{Bmatrix} \tag{2}$$

Obviously, the distance $d_t(q,o)$ obtained by formula (2) keeps in the range of $[d_{\min}(q,e_j),d_{\min}(q,e_j)+e_j.length]$ when o is moving on e_j. Here $d_{\min}(q,e_j)$ refers to the distance along the shortest path from q to e_i. Similar to the first case, when the object o reaches the next node and move to another edge, the distance function will become invalid and need to be recalculated.

4.3. The Algorithm

Given a query point q and a time interval T, the MSkyline query will return the continuous skyline set for a set of moving objects O in the road network during the time period $[t_0,t_0+T]$. We propose an event-based algorithm E_MSKY by tracking the valid events which will cause the MSK change. For example, as shown in Fig.2, the distance intersection (o_1,o_2,t_1) at time t_1 may change the dominant relationship between o_1 and o_2. We call this intersection an event. An event on o_i and o_j at time t_x is denoted as $event(o_i,o_j)=<o_i,o_j,t_x>$. According to the analysis in section 3.2, not all distance intersections can bring change to the MSK. Therefore, we can use the following two pruning strategies to cut down the event set.

Pruning strategy 1: If there is no non-spatial dominance between o_i and o_j, that is $o_i\notin Dom_{ns}(o_j)\wedge o_j\notin Dom_{ns}(o_i)$, the $event<o_i,o_j,t_x>$ is invalid to change the MSK, which can be pruned off.

Proof It is known that the static non-spatial attributes of o_i and o_j have no dominant relationship. No matter where they are moving in the road network, they can't road-dominate each other. The distance intersection between them can't cause any change to the MSK.

Pruning strategy 2: If $t_x>o_i.t_{next}$ or $t_x>o_j.t_{next}$, where $o_i.t_{next}$ and $o_j.t_{next}$ are the expected time o_i and o_j will reach the next node, the $event<o_i,o_j,t_x>$ is a wrong event which can be pruned off.

Proof Because of $t_x>o_i.t_{next}$ or $t_x>o_j.t_{next}$, which means the object o_i or o_j will reach the node and the distance function of o_i or o_j will has changed at time t_x, the $event<o_i,o_j,t_x>$ based on the previous distance function no longer exists.

In order to track the events accurately, we use a time queue Q_t to store the times when moving objects will reach the next node and an event queue Q_e to store the valid events in the order of the trigger time t_x. In the E_MSKY algorithm, we firstly divide the time interval $[t_0,t_0+T]$ into many continuous and disjoint

subintervals $([t_{s1},t_{e1}],[t_{s2},t_{e2}],...[t_{sn},t_{en}])$ based on the time points in Q_t. Then trigger the events in these subintervals sequentially and update the result set MSK. We show the detailed query processing in algorithm 1.

Algorithm1. E_MSKY

Input: query point q, moving object set O and a query time interval $[t_0,t_0+T]$

Output: The MSkyline query result L_{MSKY}

```
    // Initialization
1.  for each o ∈ O
2.      compute Dom_ns(o)
3.      if Dom_ns(o) = φ then
4.          insert the entry (o,∞,∞) into L_MSKY
5.      else
6.          insert o into O_cand
7.  end for
    //Create the events queue Q_e
8.  for each o_i ∈ O_cand
9.      for each o_j ∈ Dom_ns(o_i)
10.         if ∀j ,s.t. d_t0(q,o_i) < d_t0(q,o_j) Then
11.             insert the entry (o_i,t_0,∞) into L_MSKY
12.         compute the intersection time t_x of each event
13.         if t_x < o_i.t_next ∧ t_x < o_j.t_next
14.             enqueue <o_i, o_j,t_x> into Q_e
15.     end for
16. end for
    //Continuous query
17. while t < t_0 + T
18.     pick time interval [t_sk,t_ek] based on t_next in Q_t
19.     handleEvents( q,[t_sk,t_ek] )      // Algorithm 2
20.     add new t_next to Q_t and new events to Q_e
21. end while
22. return L_MSKY
```

The initialization is the first step in E_MSKY algorithm. Only considering the static non-spatial attributes, we can determine the non-spatial dominator set $Dom_{ns}(o)$ for every object o (line2), and distinguish the fixed MSkyline objects MSK_{ns} which always belong to the result set (lines3-6).We implement the result set MSK with a structure of linked list L_{MSKY}. Each node in L_{MSKY} stores a Mskyline object and its entering and leaving time. All the initial Mskyline objects in addition to the fixed ones are added to L_{MSKY} with the entering time t_0 (lines10-11). Then, the event queue Q_e is initialized(lines13-14). After that, we divide the query time interval and handle the events during each subinterval sequentially(lines18-19). After each subinterval, there are some new time points and new events should be added to Q_t and Q_e (line20).

The events handling in each subinterval $[t_s,t_e]$ is presented as algorithm handleEvents in Algorithm 2. For a valid event $e<o_i, o_j,t_x>$ in Q_e, in the case of $o_i \in Dom_{ns}(o_j)$, we update the tag Num_Dom_d of object o_j. If there is no other object in $Dom_{ns}(o_j)$ closer to q than object o_j, add o_j to L_{MSKY} with the entering time

$e.t_x$ (lines3-6). In the other case of $o_j \in Dom_{ns}(o_i)$, we update the tag Num_Dom_d of object o_i. If o_i was a MSkyline point at the moment before time $e.t_x$, update the corresponding node in L_{MSKY} of o_i with the leaving time $e.t_x$ (lines7-10).

Algorithm 2. handleEvents(q,$[t_s,t_e]$)

Input: query point q, event queue Q_e and subinterval $[t_s,t_e]$

Output: The update of L_{MSKY} in $[t_s,t_e]$

```
1.  e = Qe.erase()
2.  While( e.tx < te )
3.      if oi ∈ Domns(oj) then
4.          oj.Num_Dom_d --
5.          if oj.Num_Dom_d == 0 then
6.              insert the entry (oj,e.tx,∞) into LMSKY
7.      else if oj ∈ Domns(oi)
8.          oi.Num_Dom_d ++
9.          if oi.Num_Dom_d <= 1 then
10.             update the entry of oi in LMSKY to (oi,tsi,e.tx)
11.     e = Qe.erase()
12. end while
```

5. Experimental Evaluation

In our experiments, we consider a real network of Oldenburg in Germany [19], which contains 7035 road segments within the space territory of $23572\text{m} \times 26915\text{m}$. We use a generator extracted from [18] to generate moving objects in the road network. The relevant parameter settings are shown in Table 4, the speed and the static attributes of objects randomly determined in the given ranges.

Table 4. Parameters Setting

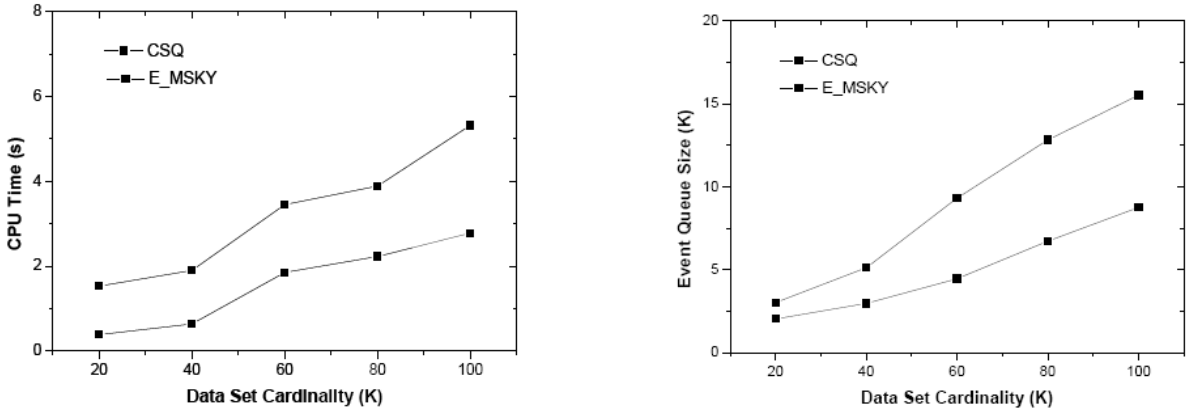
Parameter	Range
Dataset cardinality	20K, 40K, 60K, 80K, 100K
Dimensionality of non-spatial attributes	2, 3, 4, 5, 6
Range of non-spatial attribute value	[0, 500]

Since the continuous skyline query for moving objects studied by Huang et al in [12] is the closest study to this paper up to now. Although they assume the objects moving in the Euclidean space with a fixed speed in both directions, their algorithm *CSQ* also has a strategy for the updating of moving plan which can be used when the objects reach the road nodes in our network environment. So we conduct four sets of experiments to evaluate the performance of our *E_MSKY* algorithm in contrast to *CSQ* algorithm, running on Windows 7 operating system with 3.30 GHz CPU and 2.0GB memory. In order to carry out a fair comparison, the query point is randomly generated and the queries in a comparison of two algorithms are based on the same data sets. What's more, the result is obtained based on averaging the 100 continuous queries for each set of data.

5.1. Effect of Data Set Cardinality

This set of experiments uses the data sets of moving objects including 3 non-spatial attributes and a distance

attribute. The cardinality ranges from 20K to 100K. As shown in Fig.4, we compare the CPU time and the event queue size at any time unit of two algorithms. As shown in Fig.4(b), the *E _MSKY* has a bigger event queue size compared to *CSQ* , while it outperforms *CSQ* in the CPU time cost shown as Fig.4(a). That is because the handling of each event in *E _MSKY* is much simpler than the handling in *CSQ* .

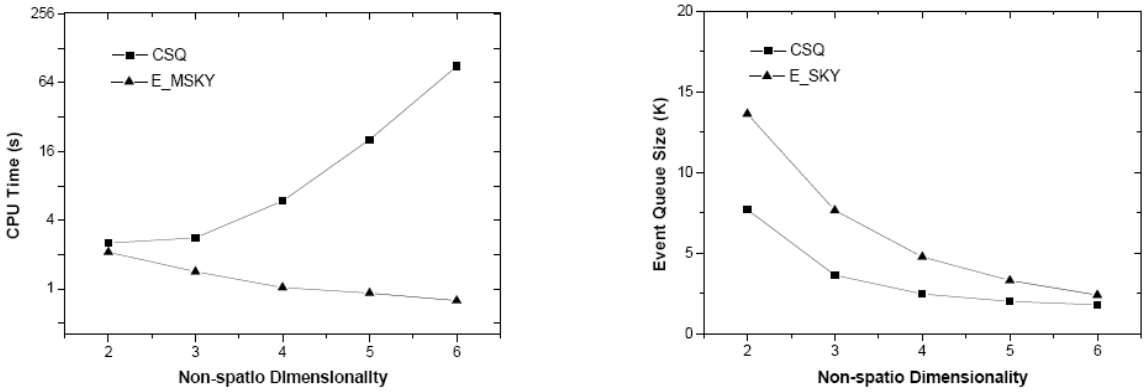


(a) CPU Time (b) Event Queue Size

Fig. 4. Effect of data set cardinality.

5.2. Effect of Non-Spatial Dimensionality

This set of experiments evaluates the performances on data sets of 50K moving objects with the non-spatial dimensionality ranging from 2 to 6. The results are shown in Fig. 5. Fig. 5(a) shows that the CPU time cost of *E _MSKY* decreases with the increase of non-spatial dimensionality, which is opposite to *CSQ* and keeps much less than *CSQ* . The event queue size of *E _MSKY* shown in Fig. 5(b) is also decreases with the increase of non-spatial dimensionality. When more non-spatial dimensionalities of the objects are considered, the possibility that an object can non-spatial dominate or be dominated by other objects will decrease, and there will be more fixed MSkyline objects and less dominators in Dom_{ns} of the candidate objects. This eventually reduces the number of events.



(a) CPU Time (b) Event Queue Size

Fig. 5. Effect of non-spatial dimensionality.

5.3. Effect of Non-Spatial Dimensionality

This set of experiments tests the effect of query time interval. We use a data set of 50K moving objects with 3 non-spatial attributes. The CPU time change with query time interval is shown in Fig. 6. With the query interval length increasing, the CPU time costs of both two algorithms are increasing. Obviously, more objects will reach the road nodes and move to another edge during a longer query interval, which will bring more events to handle

and more update to the results.

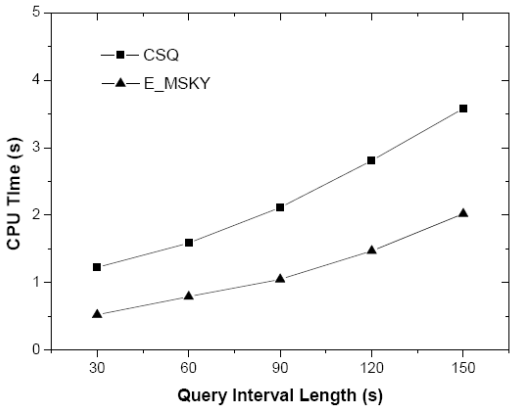


Fig. 5. Effect of query time interval.

6. Conclusion and Future Work

In this paper, we focused on a problem of efficient processing of continuous skyline queries for moving objects in road networks and proposed an event-based incremental algorithm for it. We fully considered the characteristics of the road network and moving objects for modeling and used result turning points to escape from snapshot queries. A lot of experiments we conducted and comparison with algorithm *CSQ* in [14] have verified the superiority of our proposed algorithm.

In addition, the query point we considered in our problem is a static point, Therefore our future work will still focus on the skyline queries for moving objects in road networks, but with a moving query point.

Acknowledgment

This research was supported by the National Natural Science Foundation of China (Grant No. 61373015, 61300052) and the Research Fund for the Doctoral Program of High Education of China（Grant No. 20103218110017）. It was also supported by the Priority Academic Program Development of Higher Education Institutions in Jiangsu Province, China (PAPD) and the Fundamental Research Funds for the Central Universities, NUAA (Grant No. NP2013307).

References

[1] Fang, W.-K. (2013). Finding nearest neighbors in road networks: A tree decomposition method. *Proceedings of the 16th International Conference on Extending Database Technology (EDBT)*.

[2] Lee, K. C.-K., *et al*. (2013). Efficient index-based approaches for skyline queries in location-based applications. *IEEE Transactions on Knowledge and Data Engineering (TKDE), 25(11),* 2507-2520.

[3] Deng, K., Zhou, X.-F., & Shen, H.-T. (2007). Multi-source skyline query processing in road networks. *Proceedings of the 23rd International Conference on Data Engineering (ICDE).*

[4] Borzonyi, S., Kossmann, D., & Stocker, K. (2001). The Skyline operator. *Proceedings of the 17th International Conference on Data Engineering (ICDE).*

[5] Kossmann, D., Ramsak, F., & Rost, S. (2002). Shooting stars in the sky: An online algorithm for skyline queries. *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*.

[6] Papadias, D., *et al*. (2005). Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS), 30(1),* 41-82.

[7] Chen, L., & Lian, X. (2008). Dynamic skyline queries in metric spaces. *Proceedings of the 11th International Conference on Extending Database Technology (EDBT): Advances in Database Technology, ACM.*

[8] Sharifzadeh, M. & Shahabi, C. (2006). The spatial skyline queries. *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*.

[9] Md., A.-S., & Morimoto, Y. (2012). Efficient k-dominant skyline computation for high dimensional space with domination power index. *Journal of Computers (JCP), 7(3),* 608-615.
[10] Huang Y.-K., Chang C.-H., & Chiang L. (2012). Continuous distance-based skyline queries in road networks. *Information Systems(IS),* 37(7), 611-633.
[11] Lin X., Xu J.-L., & Hu H.-B. (2013). Range-based skyline queries in mobile environments. *IEEE Transactions on Knowledge and Data Engineering(TKDE)*, *25(4),* 835-849.
[12] Mohammad, S.-A., *et al*. (2013). Skyline query for selecting spatial objects by utilizing surrounding objects. *Journal of Computers(JCP), 8(7),* 1742-1749.
[13] Cheema, M.-A., *et al*. (2013). A safe zone based approach for monitoring moving skyline queries. *Proceedings of the 16th International Conference on Extending Database Technology (EDBT)*.
[14] Huang, Z.-Y., *et al*. (2006). Continuous skyline queries for moving objects. *IEEE Transactions on Knowledge and Data Engineering (TKDE), 18(12)*, 1645-1658.
[15] Tian, L., *et al*. (2007). "Continuous monitoring of skyline query over highly dynamic moving objects." *Proceedings of the 6nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*.
[16] Chen, N., *et al*. (2012). PRISMO: Predictive skyline query processing over moving objects. *Journal of Zhejiang University-Science C (JZUSC), 13(2),* 99-117.
[17] Ding, X.-F., Jin, H., & Xu, H. (2013). Probabilistic skyline queries over uncertain moving objects. *Computing & Informatics (CAI), 32(5)*, 987-1012.
[18] Brinkhoff, T. (2002). A framework for generating network-based moving objects. *Geo Informatica, 6(2),* 153-180.
[19] Census. Retrieved from http://www.census.gov/geo/www/tiger/

Changyue Shi was born in 1990. She is a master graduate student in Nanjing University of Aeronautics and Astronautics. She majors in computer science and technology and her current research interests are in the management of moving objects.

Xiaolin Qin was born in 1953. He is a professor and a PhD supervisor in Nanjing University of Aeronautics and Astronautics. He is a senior member of China Computer Federation. His main research interests include spatio-temporal databases, secure database, data management and disaster tolerance in distributed environment and information security.

Li Wang was born in 1989, and she is a master graduate student in Nanjing University of Aeronautics and Astronautics. Her current research interests are in the management of indoor moving objects.