Hindawi
Mathematical Problems in Engineering
Volume 2018, Article ID 6749650, 14 pages
https://doi.org/10.1155/2018/6749650

Research Article

Supporting Continuous Skyline Queries in Dynamically Weighted Road Networks

Yingfeng Tang [1,2] **and Shiping Chen** [1]

[1] *Management School, University of Shanghai for Science and Technology, Shanghai, China*
[2] *Academic Affairs Section, Shanghai University of International Business and Economics, Shanghai, China*

Correspondence should be addressed to Yingfeng Tang; tangyf1983@aliyun.com

Received 21 April 2018; Revised 2 August 2018; Accepted 16 August 2018; Published 10 September 2018

Academic Editor: Mahmoud Mesbah

The paper focuses on the design of an optimum method for handling the continuous skyline query problem in road networks. Existing studies on processing the continuous skyline query focus exclusively on static road networks, which are limited because the state of roads in road networks is constantly changing. Therefore, to apply current methods for dynamically weighted road networks, a distributed skyline query method based on a grid partition method has been proposed in this paper. The method adopts the concepts of a distributed computing framework and road network preprocessing computations in which multiple parallel computing nodes are allocated and organized in grids. Using this approach, the road network map is simplified to a hub graph with much smaller scale such that the query load of the central node can be significantly reduced. The theoretical analysis and experimental results both indicate that, by using the proposed method, the system can achieve quick response time for users as well as a good balance between response times and accuracy. Therefore, it can be concluded that using the proposed method is beneficial for handling continuous skyline queries in a dynamically weighted road network.

1. Introduction

Providing location-based services (LBSs) [1] for moving objects in road networks has become an important research topic in traffic informatization and intelligent traffic system development [2] due to the advances in wireless communication and global positioning system (GPS) technologies. Recommendation systems for moving objects have become an increasingly popular function for road networks, for example, for recommending hotels, car parks, and taxi services based on the real-time locations of network users. In recent years, in conjunction with the development of numerous databases, the skyline query [3] has become an important method to solve LBSs for road networks and moving object recommendation systems. One example of the application may be that a moving object set is used to query taxis preferred by users, i.e., cars in good conditions, drivers with long service records, good ratings, and close proximity to a user's location. To achieve this, skyline can be used to conduct queries based on the user's criteria. Results are then presented to the user and they will be asked to make a decision.

The skyline query was first introduced into the database domain in 2001 by Borzsonyi et al., who proposed two basic skyline query algorithms which are block nested loops and divide and conquer (D&C) [3]. Since then, researchers have improved these algorithms and extensively studied skyline queries in a wide variety of scenarios [4–8]. With the continued expansion and development of wireless communication and mobile location technologies, the demand for LBSs has motivated researchers to extend skyline queries to various mobile computing scenarios. Sharifzadeh et al. first proposed the concept of a spatial skyline query [9], and then Deng et al. extended spatial skyline queries to road network scenarios [10]. In [10], Deng et al. measured the distance between the user and interested object as an attribute of the interested object and proposed a multisource skyline query in road networks. To this end, they introduced three algorithms which are collaborative expansion (CE), Euclidean distance constraint (EDC), and lower bound constraint (LBC). More

specifically, CE finds the next nearest neighbor for all query points. Further, EDC first calculates Euclidean distances for all data points and then runs a Euclidean skyline algorithm to output the skyline points, with those points serving as the initial candidate set of the skyline point. Finally, LBC is the same as CE, but LBC uses various optimization techniques, e.g., Euclidean distance computation is used to save network distance calculation. In 2008, Frankenstein et al. studied the continuous skyline query problem of moving objects in road networks [11] and proposed a method that processes a continuous skyline through precomputed shortest range data for targets.

More recently, researchers have studied the continuous skyline query problem of moving objects in road networks from a variety of perspectives. In [12], Huang et al. used a grid structure to index the road network and moving object information to improve the access efficiency of road network data, directly calculating the shortest path between objects using Dijkstra's algorithm. In [13], Shekelyan presented a method for computing a linear path skyline to simplify the query result set, using the multi-Dijkstra algorithm to continuously update the shortest path information. In [14], Prabha et al. focused on the authentication problem of moving objects in a continuous skyline query process, proposing a system that applies a spatial precomputed approach for continuous skyline query. In [15], Safar et al. filtered candidate interest points by calculating the nearest neighbor of each query point, thereby reducing the query space and the number of required shortest path computations wherein a spatial data structure is used to store precomputed shortest path information between nodes of the road network. In [16], Shi et al. approached continuous skyline queries by focusing on location range, transforming interest point sets into Voronoi units, and reducing query times by preprocessing data from the Voronoi units from within the road network. In [17], Fu et al. studied the continuous skyline query problem involving uncertain moving objects by introducing an uncertainty data model in which shortest path information between objects is updated directly according to each event.

In the skyline query process for road networks, the required computation of shortest path between moving objects occupies the main portion of the calculations. Dijkstra's algorithm [18] is a representative network shortest path algorithm. The algorithm propagates a search wavefront from a source vertex until a destination vertex is reached. The A* algorithm [19] is a heuristic method. The key idea of A* is that if there is only one destination, the wavefront expansion process can be optimized towards the direction of the destination vertex. The two algorithms mentioned above are on-the-fly, that is, directly computing the shortest path without any precomputing. Because of the high computing complexity of path search on large-scale road network, the on-the-fly methods cannot meet the requirement for real-time query. Precomputing methods were used to improve query response time of shortest path. There are some studies [20, 21] on the approach with precomputing transitive closure of graph. This type of approach is not suitable for large road networks, because of the huge storage space requirement. And due to huge cost of precomputing for transitive closure, it is also not suitable for dynamically weighted road networks. In [22], Hu et al. proposed a method that stores and indexes the network distances between every vertex and data object with different level of accuracy according to the distance between them. This approach is also not suitable for dynamically weighted road networks due to the high maintenance cost when the network is updated. Another kind of methods with precomputing is based on hierarchical fashion [23–26]. In these methods, a large graph is divided into smaller subgraphs and organizes them in a hierarchical fashion. For each subgraph, the border vertices are the entrances and only the distances between these border vertices are precomputed. Therefore, while computing the network distance, intermediate vertices inside the subgraph are skipped over. However, hierarchical fashion is not suitable for skyline query, since it often requires computing network distances from one source to several destinations. Thus, it is desirable to keep the distance information for the intermediate vertices.

In practice, there are two key challenges when conducting queries using the skyline-based moving object recommendation approach. First, data of moving object is collected by many sensors distributed throughout the road network. Using a traditional method of centralized data processing, large amounts of resources will be wasted in data transmission, thus causing low efficiency. Second, such a centralized processing approach is inadequate for real-time and on-demand application because of the huge numbers and widely scattered moving object data. Finally, road networks are often dynamic rather than static; therefore, the edge weights usually depend on timing because of traffic congestion and road maintenance work being carried out. In general, it is not possible to rely on any precomputation of road network data for determining an optimum route between vertices. Alternatively, computation must be based on real-time data. However, any computation based on a single point will be inadequate in meeting the real-time requirement because of the vast amount of data acquired from road networks.

Summarizing the existing skyline query methods for road network, there are two ways for calculating the shortest distance between moving objects in the skyline query process. The first approach is on-the-fly, that is, to identify the two objects as two vertices in the road network and then to use a shortest path algorithm (e.g., Dijkstra's algorithm) to perform direct calculations. This method can also be applied to a dynamically weighted road network, but the required computation rapidly increases as the scale of the road network increases. The second approach is to precompute and store the shortest distance between vertices within the road network. When querying, the shortest distance between two objects can be obtained by querying the shortest distance between the nearest neighbor vertices of each object. This approach results in faster query response times, but it cannot be applied to a dynamically weighted road network because of the huge cost from precomputation. Therefore, in this paper, grid-skyline query (Gsky) is proposed to strike a balance between the above two methods in large-scale dynamically weighted road networks.

Gsky is based on dividing a dynamic road network into small manageable units of grids. Using the proposed method,

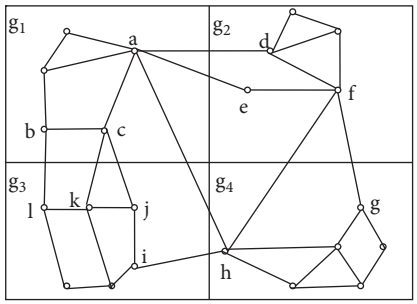

Figure 1: Example road network graph G with four grid partitions.

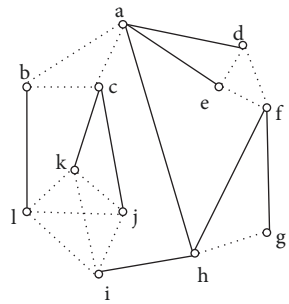

Figure 2: Example hub graph corresponding to the road network presented in Figure 1.

the entire road network is divided into a finite number of grids with a computing node allocated within each grid. This computing node is responsible for collecting and updating information about all moving objects and maintaining a localized distance for all vertices of each grid. Subsequently, a central node is placed for the entire system. When queries are submitted, the central node will gather the required data of moving object and information of localized distance from relevant computing nodes and compute the distance between the moving objects in real time. Finally, users will receive the updated skyline set from Gsky based on changes in distances between moving objects.

Gsky handles moving object information with distributed computing nodes. Therefore, the central node is only activated when there is a query; also, only the interested moving object will be involved in the computation. By using this approach, we can avoid a large amount of data transmission. Generally, parallel computation is performed by computing nodes to update the distances between local vertices. The required information is then fed to the central node only when there is a query. Therefore, the computing workload of the central node should be significantly reduced and managed. Hence the central node only needs to maintain the topology structure of the grids, thereby reducing data maintenance for the central node.

2. Relevant Models

Definition 1 (road network). A road network can be abstractly defined as weighted undirected graph G(V, E, W), where vertex set V denotes all cross-points in the road network, w represents the length of each road with the consideration of traffic congestion, with $w \in W$ and $w > 0$, and E represents all edges. As shown in Figure 1, G is a road network that includes 22 cross-points and 35 edges.

Definition 2 (moving object). A moving object is defined as a point p(m, e, pos) moving along the edge of graph G, where m represents the moving object's nonlocational attributes (i.e., symbol, type, and rating), e represents the edge, and pos identifies the distance between the starting points and the moving object on the edge. Hence, according to certain rules, one end of the road can be set as the starting point.

Definition 3 (grid). The entire road network is divided into grids, each with an equal size $N \times M$. Each grid is defined as $g(V_g, E_g, S_g, W_g)$, where vertex set V_g represents all cross-points in G, edge set E_g represents all roads passing through each grid, and vertex set S_g represents the starting point of a road in a grid with starting point s_g of road e in grid g defined as follows: (1) $s_g = s$, if road e's global starting point $s \in V_g$; (2) if s_g is the first intercept point of road e with border line g, then road e's global starting point $s \notin V_g$. Further, w_g represents the length between e's global starting point s and e's starting point in g s_g, where $s_g \in S_g$, $w_g \in W_g$, and $w_g \geq 0$. As shown in Figure 1, road network G is divided into four grids g_1–g_4.

Theorem 4. *Assume that a moving object is identified as $q(m, e, pos_g)$ in grid g, where n represents the nonlocational information of the moving object, e represents the edge, and pos_g is the distance between s_g and the moving object, with $s_g \in S_g$. And there is a global moving object p(m, e, pos) in graph G corresponding to q; then, $pos = pos_g + w_g$.*

Proof. Theorem 4 is valid based on Definitions 2 and 3. □

Definition 5 (distance). The distance between p and q is defined as the length of the shortest route between p and q in G; this distance is denoted as d(p, q).

Definition 6 (hub graph). For graph G, if $V_g \neq \Phi$, then g is a hub. For edge e, if the two end vertices of e are not in the same grid, then e is called a bridge edge, and its end vertices are called bridge vertices. Connecting bridge vertices with virtual edges e_i from the same hub to vertices in a different hub with bridge edges forms hub graph $G_c(V_c, E_c, E_i, W_c, W_i)$, where V_c represents the bridge vertex set, E_c represents the bridge edge set, E_i represents the virtual edge set, w_c represents the length of the bridge edge, with $w_c \in W_c$ and $w_c > 0$, and w_i represents the length of the virtual edge, i.e., the distance between two ends of a virtual edge within a hub, with $w_i \in W_i$ and $w_i > 0$. The graph in Figure 2 shows an example hub graph of the road network presented in Figure 1.

Theorem 7. *The shortest route between any two vertices in a hub graph corresponds to the shortest route in the original road network.*

Proof. Consider the situation in which the shortest route between any two vertices in the hub graph will not include

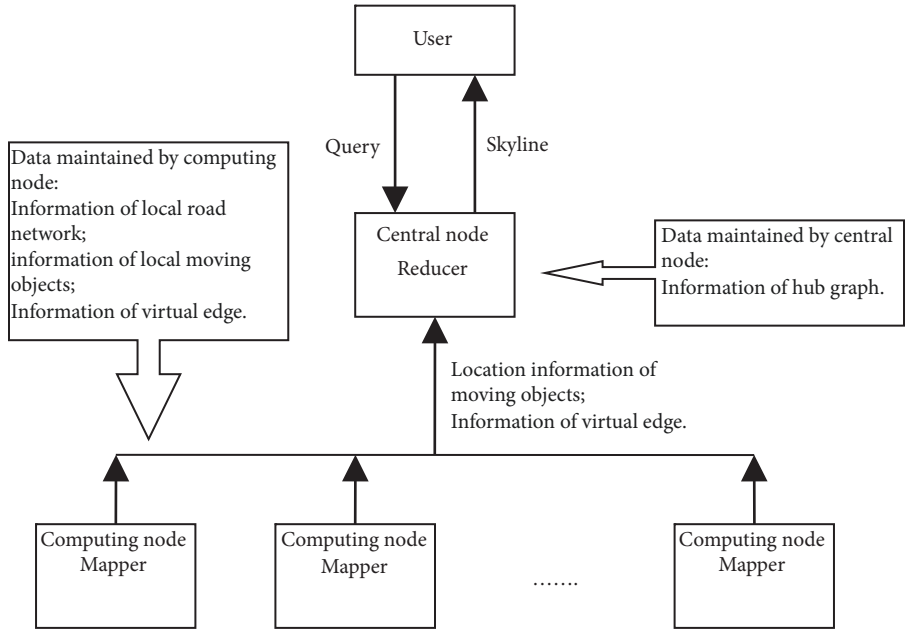

Figure 3: Schema of our Gsky approach.

any virtual edges. On the basis of Definition 6, all edges along this route will correspond to an edge in the original road network. Therefore, this route will be the shortest in the original road network. Consider the other situation in which a virtual edge is included in the computation for the shortest route; assuming that the virtual edge is pq, then on the basis of Definition 6, we can observe that all edges along the shortest route between p and q in the road network will be in the same hub. Given that pq is the shortest route in this hub, we can therefore conclude that pq must be the shortest route between p and q in the original road network. Thus, Theorem 7 is proven. □

Theorem 8. *The shortest route between any two points in a hub graph traverses no more than one virtual edge in the same hub.*

Proof. Assume that the shortest route between vertices p and q in hub graph G_c can be expressed as $r(p \ldots \underline{m_1 \ldots m_n} \ldots q)$, with $n > 1$ and the route $\underline{m_1 \ldots m_n}$ representing a continuous virtual edge (with exactly $n - 1$ such edges) that passes through points of n in hub M. From Definition 6, we can observe that the length of the route $\underline{m_1 \ldots m_n}$ is $l = d_M(m_1,m_2) + \cdots + d_M(m_n - 1,m_n)$, and there is at least one virtual edge, which is $\underline{m_1 m_n}$ with length $l' = d_M(m_1,m_n) \leq l$. Therefore, the shortest route between p and q is $r(p \ldots \underline{m_1 m_n} \ldots q)$ with only one virtual edge. Thus, Theorem 8 is proven. □

From Definition 6 and Theorem 8, it is obvious that a road network can be simplified with a small-scale hub graph for processing the distance between points in order to reduce the computing workload. From Theorem 8, one can know that it is possible to skip some of the routes in the hub graph when computing distance between points; therefore, query domain can be reduced.

Definition 9 (domination). Nonlocational attributes which the user is interested in together with the distance d between the moving object and the user's position form attribute space A and define data point p(a1,a2,...an,d) in A, with a1,a2,...an,d $\in$ A. For any two random points p1 and p2, if $\forall ai \in A$, then p1.ai $\leqslant$ p2.ai and $\exists aj \in A$; therefore, p1.aj $<$ p2.aj, which is defined as p1 dominating p2 and denoted as p1 $\prec$ p2. If p1 $\prec$ p2, then p1 does not dominate p2, which is denoted as p1 $\not\prec$ p2.

Definition 10 (skyline). Given that set C contains information that the user is interested in corresponding to a given road network, all data which are not dominated by other data in C form a skyline set, denoted as SKY(C) = {p1 $\in$ C|$\forall$p2 $\in$ C:p1 $\not\prec$ p2}.

3. Continuous Skyline Query in a Dynamically Weighted Road Network

3.1. Fundamental Structure of Our Method. The overall structure of Gsky approach is presented in Figure 3. As noted above, Gsky uses N × M grids to divide the road network. A computing node is located within each grid and is responsible for processing and maintaining the information, as well as collecting and updating moving object information within the corresponding grid. The computing node is also responsible for computing and updating distance information between points; therefore, there is no communication between points, and asynchronous computing is used to update information corresponding to each point. A central node then processes user's queries; this central node also maintains the hub graph and collects information from each computing node, providing feedback with skyline set which the user is interested in.

```
input interested moving object set P
output moving object set P with updated location information
Steps:
to mapper:
P.get(); // read object assigned to current mapper from input cache
foreach p in P
   p.pos=p.posg+p.e.wg; // compute global location information
P.send(); // send the result to output cache
to reducer:
P.get(); // gather objects from each mapper
```

Algorithm 1: Obtain global location information of a moving object.

The procedure is summarized as follows. When there is no query task, each computing node updates locational information for moving objects on a regular basis, also computing and updating distance information between points depending on changes in relevant weights. Once a query is initiated, the central node requests regional location information for the interested moving object from the computing node and then processes this global location information. Next, the central node computes the distance between each interested moving object and the user's position by gathering information of the virtual edges in each relevant hub from computing nodes. Eventually, the central node generates a skyline set based on the distances between the interested moving objects and the user's position, as well as nonlocational attributes of the moving objects; finally, this skyline set is returned to the user.

The distributed computation structure is based on an approach known as MapReduce [27], which makes use of cloud computing and has the advantages of being highly distributed and error-tolerant. Despite the fact that MapReduce was only able to process batched queries when it was first introduced, many researchers have worked on it extensively to improve its ability to process online data in real time [28–30]. Currently, MapReduce is already able to process real-time information; therefore, in current work, the system uses MapReduce and allocates a mapper to each computing node and a single reducer as the central node.

3.2. Maintaining Location Information for Moving Objects. Maintaining location information for a moving object includes processing for both regional and global location information. The computing node within each grid is responsible for updating regional location information for each moving object. Here, maintaining each data node is based on the road information in the grid within which the computing node is located. According to Definition 3, road information can be described using set (e, V_g, s_g, w_g), where e represents the road, V_g represents the end of the road, s_g represents the starting point of the road within a grid, with the starting point denoted as s when it is within a grid and s_g when it is on the boundary of a grid, and w_g represents the location of s_g, where $w_g = 0$ if s is within the grid (otherwise, w_g is the distance between s_g and s). Moving objects within a grid are described as (m, e, pos_g), where n represents the nonlocational attributes of the moving objects, e represents the road where the moving objects are located, and pos_g represents the distance between the moving objects and s_g. The computing node regularly updates the road and moving object information by receiving signals from sensors within the grids and from corresponding GPSs.

When a query is submitted by a user, the central node requests the global location information from the computing nodes. The global location information of the moving objects can be described as (m, e, pos). According to Theorem 4, it is straightforward to know that the location of a moving object is $pos = pos_g + w_g$.

For example, the system allocates a computing node to maintain the moving object data in g3 shown in Figure 1. Suppose that the cross point between the road cj and the g3 boundary is s, and the length of the cs is 5; then the road cj in g3 can be defined as (cj, c, s, 5), and road kc in g3 can be defined as (kc, k, k, 0). Suppose that there is a moving object o1 on the road cj which falls in g3, and the length between o1 and s is 2; then o1 in g3 can be defined as (m, cj, 2), and the global definition of the o1 in the central node is (m, cj, 7). Suppose that there is a moving object o2 on the road kc which falls in g3, and the length between o2 and k is 3; then o2 in g3 can be defined as (m, kc, 3), and the global definition of the o2 in the central node is (m, kc, 3).

These calculations are processed via MapReduce only once, with the detailed algorithm presented as Algorithm 1.

3.3. Distance Computation between Users and Moving Objects. If the position of a user is fixed on a given road network and the moving objects move with a constant speed, in order to obtain the skyline set which is interested by the user, it is necessary to process the distance between them in the road network. If the location of the user and the interested moving objects are viewed as vertices in the road network, we can then use Dijkstra's algorithm [18] to directly obtain the distance; however, this algorithm requires a large amount of computing resources and therefore cannot meet the needs of live responses. In present work, the system applies a conversion method to process two random points in the road network by which a relatively fixed computation is performed to calculate the distance between vertices in a topological structure. Using appropriate preprocessing of computation,

```
input source vertex s and destination vertex t
output  d(s,t)
Steps:
to reducer:
S.include(s); // define set S,add s to S
T=V-S; // set V include all vertices in the road network
while(T)
  R=GetNeighborRoute(s,S,T); // find all routes between s and neighbor vertices of S in T as R
  foreach r in R
   if r. ConVedgeNum>=2
      R.delete(r); // if r contains more than 2 adjacent virtual edges, delete r from R
      continue();
   else if r. VedgeNum>=1
      r. Vedges.get(); // if r contains virtual edge,sent this virtual edge into input cache of mapper
   r.length.compute(); // compute length of r
  r=R.GetShortest(); // pick the shortest route,add its ends to S
  S.include(r.end);
  T.delete(r.end);
  if(r.end==t) // return result when reaches destination t
     return r.length;
to mapper:
Vedge.get(); // get virtual edges from input cache
Vedge.length=GetLength(Vedge.id);
Vedge.sent(); // sent length of virtual edges into output cache
```

Algorithm 2: Compute the distance between two vertices in a hub graph.

the system can substantially reduce computing and achieve fast response times.

More specifically, assume that we have two random points, p and q, both on roads ab and cd. If the starting point of road ab is point q and the starting point of road cd is point c, then the road lengths by a broad definition of distance for ab and cd are w_{ab} and w_{cd}, respectively. Here, there are four routes between p and q which are

r_1(p,a, c,q);

r_2(p,a, d,q);

r_3(p,b, c,q);

r_4(p,b, d,q).

The shortest length of the four routes is represented as follows:

$$d(r_1) = p.pos + d(a,c) + q.pos;$$

$$d(r_2) = p.pos + d(a,d) + (w_{cd} - q.pos);$$

$$d(r_3) = (w_{ab} - p.pos) + d(b,c) + q.pos;$$

$$d(r_4) = (w_{ab} - p.pos) + d(b,d) + (w_{cd} - q.pos).$$

Therefore, $d(p, q) = \min(d(r_1), d(r_2), d(r_3), d(r_4))$, which indicates that the distance between two points p and q can be obtained by calculating the distance between the endpoints of the two routes.

The distance between two random points in the road network can be processed by the central node in the corresponding hub graph. The length of the bridge edge in the hub graph is regularly updated by the central node, whereas the length of the virtual edge is regularly updated by the computing nodes. When calculating distance, the central node will start from one end using Dijkstra's algorithm and stop once the algorithm reaches the other endpoint; further, when the length of the route needs to be compared, MapReduce will collect length information for virtual edges from the computing node. When calculating the optimum route, Theorem 8 can be applied to remove unqualified routes and therefore substantially reduce the query region.

For example, suppose that in the road network shown in Figure 1, it is necessary to compute the network distance between query point u on road ac and the moving object o on road fg. Then d(a, f), d(a, g), d(c, f), and d(c, g) should be computed first. Taking d (a, f) as an example, when the central node starts computing d (a, f), the data of the 13 virtual edges (namely, ab, ac, bc, de, ef, df, gh, kj, ji, il, lk, lj, and ki) are first obtained from computing node, and then the shortest path algorithm (e.g., Dijkstra's algorithm or A^* algorithm) is used to compute d (a, f) in the hub graph shown in Figure 2. During the computing, if there are more than two consecutive virtual edges on the path, e.g., ac-cb, the path is abandoned.

Algorithm 2 provides the detailed steps.

Finally, note that the computing nodes should update the shortest path tree between each of the nodes and provide length information for the virtual edge back to the central node as necessary. Given the ever-changing weights, the shortest path tree must be reprocessed whenever there is an update; therefore, it is time-consuming and may also cause outdated information to be fed from the computing nodes to the central node. Therefore, instead, an incremental updating algorithm [31] is used to achieve a better performance.

```
input  interested points set A, dominant relationship M1, comparison of distance M2.
output skyline set SKY
Steps:
A.SortByDistance(); // sort points in A by distance attribute
for(i=1;i<=A.MaxNum;i++) // updateM2
   for(j=1;j<=i;j++)
      if(M2[i][j].CompareFlag!=compare(A[i],A[j])) // if the relationship changes, update the relationship
          M2[i][j].CompareFlag=compare(A[i],A[j]);
          M2[i][j].Changed=true;
      else
          M2[i][j].Changed=false;
SKY.include(A[1]); // initialize SKY
for(i=2;i<=A.MaxNum;i++) // compute SKY
   foreach p in SKY
      if(M2[i][p.id]. Changed) // if the relationship of distance attribute changes
         M1[A[i].id][ p.id]=ComputeDominate(A[i],p); // compute dominant relationship and update M1
   if(!SKY. Dominate(A[i],M1)) // judge whether there is a point in SKY that dominates A[i] according to M1
      SKY.include(A[i]); // if no point dominates A[i], add A[i] to SKY
return SKY;
```

Algorithm 3: Update the skyline set.

3.4. Updating the Skyline Set. Once the system finishes computing the distance between the interested moving objects and the user's position, the central node updates the skyline set using the distance attribute and other nonlocational attributes. Often, the numbers of interested moving objects are limited; therefore, it is not necessary to use skyline query algorithm with index. Among nonindexing skyline algorithms, the sort-filter-skyline (SFS) algorithm [32] is efficient and progresses incrementally, which is suitable for the present work. Therefore, a modified SFS algorithm is incorporated to process the skyline set. Further, pruning rule 1 is presented below, which is based on the assumption that nonlocational information does not change when a moving object moves.

Pruning Rule 1. Assume that the relationship between distance attribute d of data points p and q remains unchanged; then, the dominant relationship will remain unchanged between p and q.

Proof. Because the nonlocational attributes of p and q do not change as they move, their relationship will not change. Further, if the relationship between d of p and q remains the same, then relationship between all attributes of p and q will not change. Therefore, on the basis of Definition 9, it can be inferred that the domination between p and q will remain the same. □

To use pruning rule 1 to modify the SFS algorithm, the system establishes two global data tables which are M_1 and M_2. M_1 maintains dominant relationship of data points whereas M_2 maintains the relationship between distance attribute of data points. In the SFS algorithm, all data is arranged first by the distance attribute dimension; at the same time, the system updates M_2 and extracts data for p from the pending skyline set by sorting from the shortest distance to the longest distance. This process consists of the following three steps.

(1) Query M_2: if the relationship between the distance attribute of p and the point p^* in SKY (herein, SKY refers to current skyline set) does not change, the dominant relationship between p and p^* can be obtained from M_1; otherwise, compute the dominant relationship between p and p^* and update M_1.

(2) If there is a point dominating p in SKY, discard p; otherwise, add p to SKY.

(3) Return SKY when all of the points have been processed.

Algorithm 3 provides the detailed procedure for these above steps.

4. Algorithm Analysis

4.1. Computational Analysis. The majority of the computational cycles for Gsky are concentrated on calculating the distance between the moving objects and the user. Therefore, this subsection focuses on describing how the calculation is actually processed. For the convenience of the discussion here, the road network is defined as an $n \times n$ grid network, divided with M square grids of the same size, where $1 \le M \le n^2$. Further, assume that there are N vertices in the road network. Given the above, then

$$N = (n+1)^2. \tag{1}$$

Here, N_g is the total number of vertices in each grid, which can be represented as

$$N_g = \frac{N}{M} = \frac{(n+1)^2}{M}. \tag{2}$$

Bridge vertices are the outermost crossing points, identified as hollow circles in Figure 4.

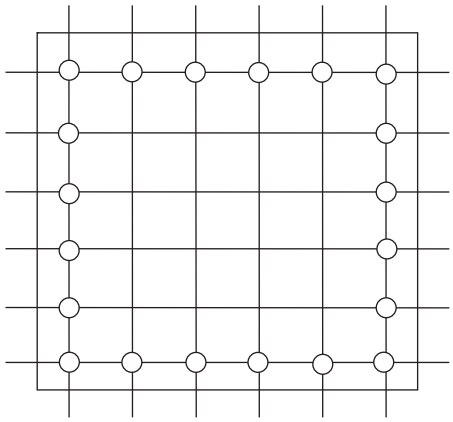

Figure 4: Example of bridge vertices (i.e., the hollow circles) within a grid (i.e., the bounding box).

Here, assuming that the number of bridge vertices within each grid is N_{eg}, then

$$N_{eg} = 4\sqrt{\frac{(n-1)^2}{M}}. \tag{3}$$

Taking the example with road network shown in Figure 4, suppose that the grid network is composed of 11∗11 small grids. The whole network is divided to 4 bigger grids, and one of them is shown in Figure 4. Then the total number of vertices in the network is 12∗12=144. The number of vertices in the bigger grid is 144/4=36. The number of bridge vertices in the bigger grid is 4∗10/2=20.

Further, assuming that the total number of bridge vertices in the hub graph is N_e, then

$$N_e = MN_{eg} = 4(n-1)\sqrt{M}. \tag{4}$$

Next, assuming that the number of interested moving objects is P, the computational capacity for a single computing node is C_s, and the computational capacity for the central node is C_m. On the basis of the complexity of the shortest route method, (2) and (4),

$$C_s = \left(\frac{(n+1)^2}{M}\right)^3 \tag{5}$$

$$C_m = P\left(4(n-1)\sqrt{M}\right)^2 = 16PM(n-1)^2. \tag{6}$$

Next, on the basis of (5) and (6), the computational complexity for a single computing node can be derived as $O((N/M)^3)$; for the central node, the computational complexity is O(PMN).

4.2. Analyzing the Amount of Traffic. Communication in the system consists of two key parts. First, there is the traffic distribution (denoted by D_1) of the central node which collects information from the computing nodes within each grid. This portion of the traffic distribution is equal to the number of interested moving objects; i.e., $D_1 = P$. Second, there is the traffic distribution originating from queries of the distance between the interested moving objects and the user. Here, the central node requests information regarding the virtual edges from computing nodes; this portion of the traffic distribution is denoted as D_2. During the computation, the central node acquires the number of virtual edges with the maximum number of all virtual edges in all grids M, performing such collections for P times. Assuming that the number of bridge vertices in each grid is N_{eg}, then

$$\begin{aligned} D_2 &= PM\frac{N_{eg}\left(N_{eg}-1\right)^2}{2} \\ &\le P\left[8\left(\sqrt{N}-1\right)^2 - 2\left(\sqrt{N}-1\right)\sqrt{M}\right]. \end{aligned} \tag{7}$$

Given (7), the overall traffic D is calculated as

$$\begin{aligned} D &= D_1 + D_2 \\ &\le P + P\left[8\left(\sqrt{N}-1\right)^2 - 2\left(\sqrt{N}-1\right)\sqrt{M}\right] \\ &\le P + 8PN. \end{aligned} \tag{8}$$

From (8), the scale of the traffic for the entire system can be determined to be O(PN), whereas the scale of data is $O(N^2)$ for the road network. Therefore, when $P \le N$, the traffic through the system does not exceed the scale of data for the road network.

4.3. Deciding the Numbers of Grids. From Section 4.1, it is known that the range of number of grids M is $[1, n^2]$. When M = 1, the computational complexity is O(PN) for the central node and $O(N^3)$ for each computing node. Therefore, in this case, Gsky degenerates into a precomputing algorithm on a single computing node. Conversely, when $M = n^2$, the complexity of computation for the central node is $O(PN^2)$ and O(1) for each computing node. Therefore, in this case, Gsky degenerates into a Dijkstra algorithm running on a single central node. To properly balance the computational load over the central node and the computing nodes, a suitable value of M must be determined. Assuming that the computational capacity of the data processing center is similar to that of the computing nodes, then it will achieve this balance when the computational load for the central node is equal to that for each computing node; i.e., from (5) and (6),

$$M = \left(\frac{(n+1)^6}{16P(n-1)^2}\right)^{1/4} = \left(\frac{N^3}{16P\left(\sqrt{N}-2\right)^2}\right)^{1/4}. \tag{9}$$

From (9), grid number M can be determined by the number of vertices in road network N and the number of interested points P.

Taking the example with road network shown in Figure 4, the number of vertices is 144. When the number of interested moving objects is 7, the number of grid is divided into 4, and the performance of the system can be optimized.

Further, the distribution of vertices is not homogenously distributed in real road networks. Therefore, to balance the

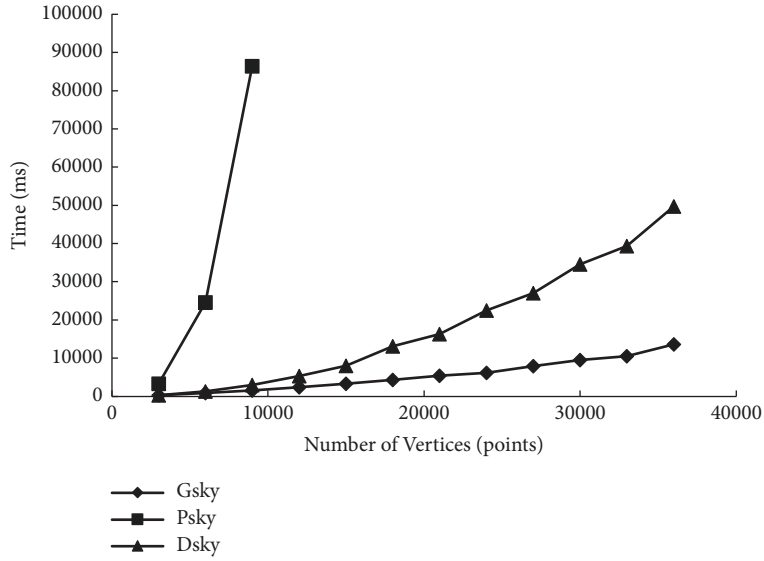

Figure 5: Query response times at different road network scales in synchronous mode (comparison of basic algorithms).

computational load between the nodes, the road network may be divided into grids unevenly such that each grid contains a similar number of vertices. Balanced load of nodes can maximize the efficiency of the entire system.

5. Experiments and Results

For the experiments, six PCs connected using 100 M Ethernet, each with a 2 GHz Intel Core 2 processor, a 2 GB RAM, a 260 GB hard drive, and the CentOS 6.2 operating system, were used. The system used Hadoop Online Prototype (HOP) [29] for Gsky, where one PC served as the central node and the remaining five PCs served as computing nodes. The number of mappers located on the computing nodes depended on how the network was divided into grids; the reducer was also allocated on the central node.

5.1. Comparison of Basic Algorithms. For the experiments, two basic algorithms are used to serve as comparisons for Gsky. First, the Dsky algorithm corresponds to the direct calculation method described in the Section 1. Second, the Psky algorithm corresponds to the precomputing method described in Section 1. In Dsky, the moving objects and user are treated as two vertices in a road network for processing the distance between them. Here, Dijkstra's algorithm is used first, and then the modified SFS is applied to serve as a comparison. In Psky, on the basis of the preprocessing algorithm, the shortest path tree is periodically updated. When processing the distance between the moving object and users, information regarding the related vertices is collected, describing their relative relationships in the road network. Next, the modified SFS is applied as a comparison.

Note that managing moving objects when applying Dsky and Psky occurs through the distributed computing system proposed in our work; however, distance calculation and preprocessing computation are both performed in the central node.

The data used in the experiment were based on a real road network from Beijing [33]. This dataset consisted of 433,391 roads and 171,504 vertices. Each test was performed using a randomly selected region within the road network. Each road was multiplied by a broadly defined length parameter, μ, with a multiplicative range of 1 to 10 and an exponent distribution randomly changing every 5 to 10 s. There were 500 interested moving objects moving in the road network at constant speeds from 10 to 40 km/h. Each moving object had three nonlocational attributes along with location information, thereby making each a four-attribute object. The number of nonlocational attributes ranged from 1 to 500 and remained unchanged as the object moved.

Accuracy and response times were two criteria used to determine the performance of each method. Changes to these two criteria were observed while we changed the scale of the road network (i.e., increasing the number of vertices) for all three algorithms. Average results were based on a 10-time test for each criterion.

Further, both synchronous and asynchronous modes were used to evaluate Gsky and Psky. For the synchronous mode, before processing began, the information of virtual edges and the shortest path tree were updated to help in achieving a high level of accuracy. Conversely, with the information of virtual edges and the shortest path tree being periodically updated, the asynchronous mode, before processing the data, does not consider whether the data is in the latest state and directly uses the current state of data for distance computation to ensure that response times are reasonable.

Figure 5 presents the change in query response times versus the scale of road networks for the three algorithms in synchronous mode. From the figure, it can be observed that the overall trend for all three methods was the same; i.e., the query response time increased as the scale of the road network increased. Query response times increased most significantly for Psky because the computing nodes had

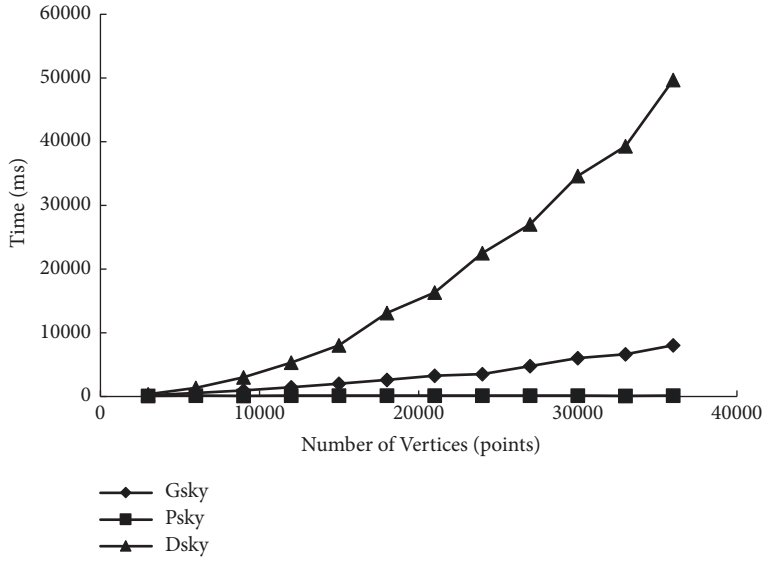

Figure 6: Query response times at different road network scales in asynchronous mode (comparison of basic algorithms).

to update the shortest path tree, thereby creating a large computing load and causing low efficiency. The increase in query response times for Dsky grew only slightly as the scale of the road network increased. For Gsky, the road network was divided into grids, and there were many computing nodes involved in processing the virtual edges, thereby allowing a very large number of preprocessing computations to be distributed; therefore, the impact on query response times as the scale of the road network increased was very small. It can be seen that in synchronous mode, the performance of Psky is the worst, Dsky is better than Psky, and the performance of Gsky is much better than the other two algorithms.

Figure 6 presents the change in query response times versus the scale of road networks for the three algorithms in asynchronous mode. It can be observed for Psky that the response times barely changed as the scale of the road network increased (in fact, the expectation of query response time of Psky keeps around 600 ms) because it removed the task of computing the shortest path tree for vertices and reduced the computing load for distance. Similar to synchronous mode, the response times for Dsky increased significantly. For Gsky, despite the computing load being reduced because of the preprocessing computation, it still took time for the central node to process via the computing nodes in the hub graph. Therefore, the response times increased steadily, and the increase was lower overall than that of synchronous mode. It can be seen that in asynchronous mode, Psky has the best performance, the performance of Dsky is the worst, Gsky is slightly lower than Psky, but far better than Dsky.

Figure 7 presents the change in accuracy versus the scale of the road network for the three algorithms in asynchronous mode. It can be observed that the accuracy keeps close to 100% for Dsky since it computes in real time and does not rely on any preprocessing; however, Psky relies on the preprocessing. Therefore, in asynchronous mode, the time-consuming preprocessing is not capable of catching up with the speed of incoming queries, which means that only a small part of the shortest path tree is actually updated when the query request occurs. Therefore, despite the short response times for Psky, the accuracy dropped significantly as the scale of the road network increased. Gsky also relies on preprocessing, but it takes a much shorter amount of time, which means that most of the updates for the virtual edges were completed when the central node was trying to calculate the distance. Therefore, the accuracy of Gsky only dropped slightly as the scale of the road network increased. According to Figures 6 and 7, it can be seen that in asynchronous mode, Dsky has the highest accuracy, but the response time is higher; the response time of Psky is the lowest, but the accuracy is poor; compared to the other two algorithms, Gsky keeps high accuracy with lower response time.

In summary, in synchronous mode, the response times for Gsky increased steadily as the scale of the road network increased, indicating that Gsky can perform well even in relatively large-scale road networks. Therefore, we can conclude that Gsky is the best method of the three evaluated methods in synchronous mode. In asynchronous mode, the response times for Gsky were not as short as in Psky, and the accuracy of Gsky was not as good as Dsky; however, overall, Gsky still performed the best when we consider both criteria. Therefore, we conclude that Gsky is the most suitable for the real-world application.

5.2. Comparison of Existing Algorithms. In order to compare the performance of Gsky and existing methods, two representative existing algorithms have been implemented under the experimental environment above. The first one is the LBC algorithm in [10], which is an on-the-fly method without any precomputing. The method minimizes the cost of network distance computation in skyline points computing using the *path distance lower bound* and reduces the searching space of skyline points with the optimized methods. The method has good performance for multisource skyline query in road network. The second algorithm was presented in

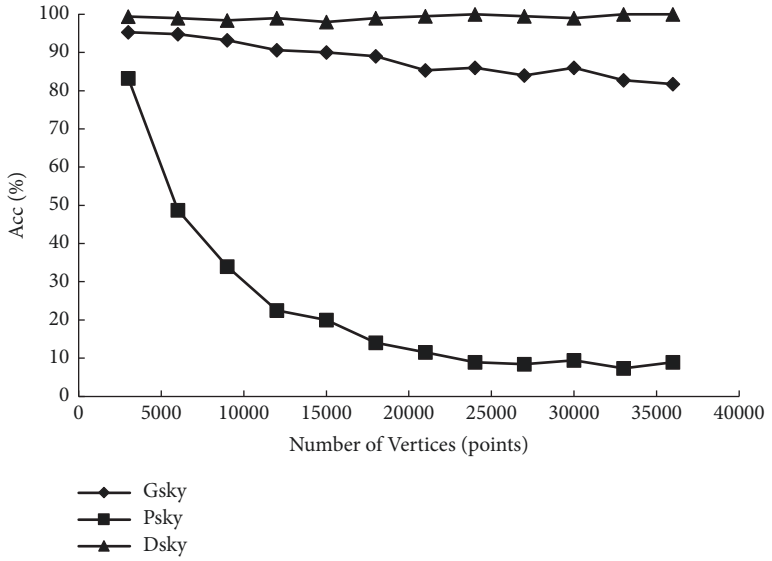

Figure 7: Accuracy at different road network scales in asynchronous mode (comparison of basic algorithms).

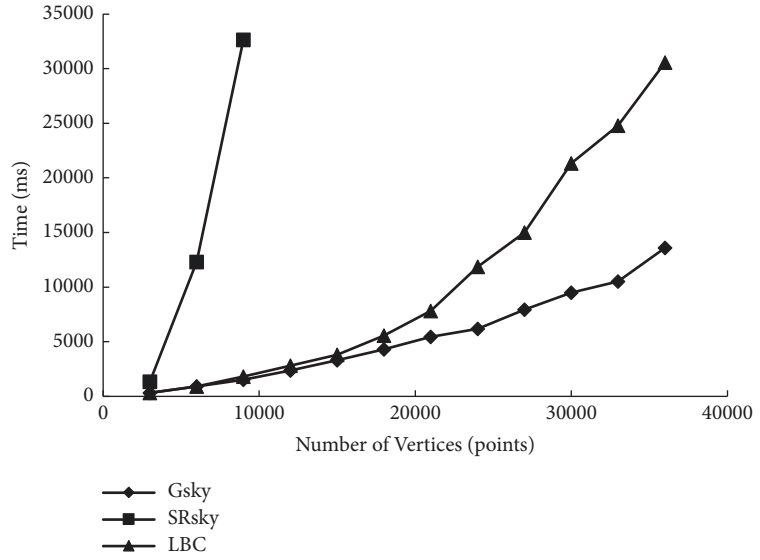

Figure 8: Query response times at different road network scales in synchronous mode (comparison of existing algorithms).

[11]. This method processes a continuous skyline through precomputed *shortest range* data of targets and has good performance for continuous skyline query in static road network. For convenience of discussion, the method is named as SRsky.

For LBC, the query points have been set to 1, and one computing node has been set for maintaining moving objects and road network data. The computing node is also responsible for maintaining the *path distance lower bounds*. One central node has been set for computing the shortest paths and skyline set for users. For SRsky, one computing node has been set for maintaining road network, moving objects, and their *shortest range* data, also one central node for the shortest paths and skyline set for users.

Figure 8 presents the change in query response times versus the scale of road networks for the three algorithms in synchronous mode. From the figure, it can be observed that in synchronous mode, LBC has better performance than Dsky because of the optimization of network distance computation. SRsky has the worst performance because the load for computing shortest range data grows as fast as the increase of network scale. Gsky has the best performance among the three algorithms.

Figure 9 presents the change in query response times versus the scale of road networks for the three algorithms in asynchronous mode. It can be observed that the query response time of SRsky keeps the lowest level (keeps around 500 ms) due to the precomputed shortest range data. The query response time of LBC is the same as that of the synchronous mode. The query response time of Gsky is lower than LBC, because the load for computing network distance in the hub graph is lower.

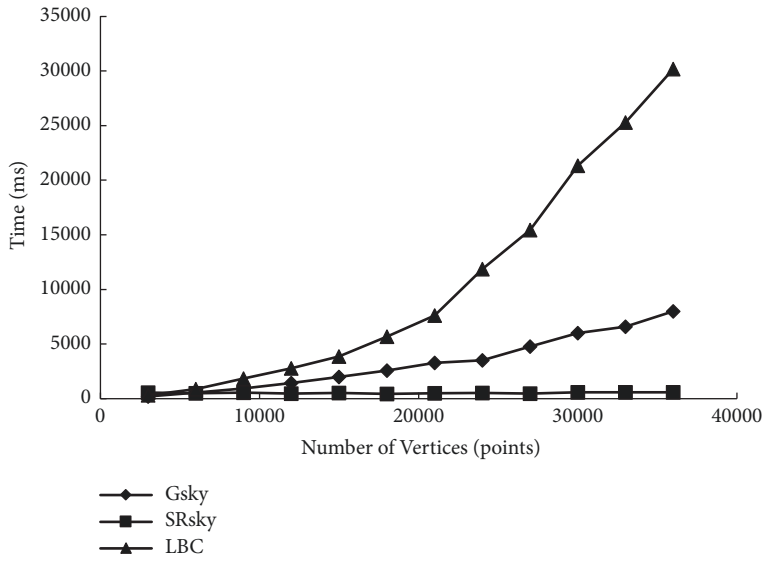

Figure 9: Query response times at different road network scales in asynchronous mode (comparison of existing algorithms).

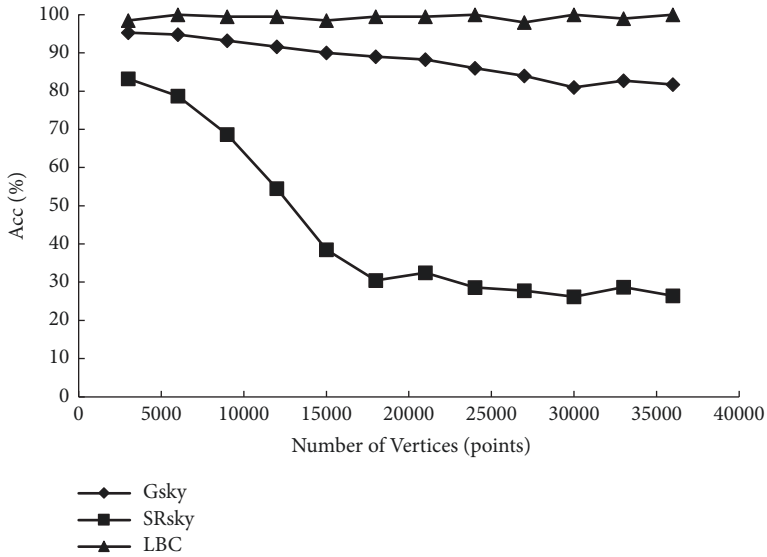

Figure 10: Accuracy at different road network scales in asynchronous mode (comparison of existing algorithms).

Finally, Figure 10 presents the change in accuracy versus the scale of the road network for the three algorithms in asynchronous mode. It can be observed that as an on-the-fly algorithm, LBC keeps the highest accuracy (keeps around at 100%). The accuracy of SRsky is greatly improved compared with Psky, because the shortest range data which needs to be precomputed is much smaller than the shortest path tree of all vertices. The accuracy of Gsky is between LBC and SRsky, and keeps a relatively high level.

In summary, compared with the existing algorithms, the Gsky algorithm can maintain reasonable response time and balance the response time with high levels of accuracy in large-scale dynamically weighted road network.

6. Conclusions

In this paper, Gsky algorithm was proposed to process moving objects in a dynamically weighted road network. Using the approach, the road network is first divided into grids such that it can be simplified and treated as a small-scale hub graph. In a hub graph, the skyline set of moving objects is then processed and recommended to users. This method has the advantage of supporting dynamically weighted road networks and applies a distributed computing structure, thereby distributing very large computing load from vertices to computing nodes to reduce the computing load of the central node and improve query response times. In the work here, the system applies a new computing-focused-around-data approach in which a local road network is

managed locally and information is collected only when necessary such that traffic throughout the system is substantially reduced. Through the work, analysis and experiments show that, compared to current methods, the Gsky algorithm and approach can maintain reasonable response time and balance the response time with high levels of accuracy, even in very large-scale dynamically weighted road networks.

In Gsky algorithm, since the vertices are not homogenously distributed in real road networks, dividing road network into grids will make the number of vertices in each grid inhomogenously distributed. This leads to the unbalanced load of the computing nodes and reduces the efficiency of the system. The future research will focus on finding a better road network partition method, which can make a more balanced partition of the road network, and apply the method to the presented distributed system.

Nomenclature

G: A road network which is defined as a weighted undirected graph
V: Vertex set of G
E: Edge set of G
W: Length of each edge
p, q: Positions of moving objects which are defined as points on edges
m: The moving object's nonlocational attributes
v: Vertices in V
e: Edges in E
w: Lengths in W
pos: Length between the starting points and position of the moving object on the edge
g: Grid
V_g: Vertices which falls in grid g
E_g: All edges passing through grid g
S_g: Starting points of edges which is defined in grid g
s_g: Points in S_g
s: Global starting point of e
w_g: Length between s and s_g
W_g: Set of w_g
pos_g: Length between s_g and position of the moving object on the edge
d(p, q): The length of the shortest route between p and q in G
G_c: Hub graph of G
v_c: Bridge vertices
V_c: Set of v_c
e_c: Bridge edges
E_c: Set of e_c
e_i: Virtual edges
E_i: Set of e_i
w_c: Length of bridge edge
W_c: Set of w_c
w_i: Length of virtual edge
W_i: Set of w_i
r(p,q): Route from p to q in G
P: Data points
A: Attribute space
a: Attributes in A
C: Set of P
SKY(C): Skyline set of C
n: Number of grids in the grid network
M: Number of grids which divide the grid network
N: Number of vertices in the grid network
N_g: Number of vertices in one grid
N_{eg}: Number of bridge vertices in one grid
N_e: Number of bridge vertices in in the hub graph
C_s: Computational capacity for a single computing node
C_m: Computational capacity for the central node
D: Traffic distribution.

Abbreviations

LBSs: Location-based services
GPS: Global positioning system
D&C: Divide and conquer
CE: Collaborative expansion
EDC: Euclidean distance constraint
LBC: Lower bound constraint
Gsky: Grid-skyline query
SFS: Sort-filter-skyline
HOP: Hadoop online prototype
Dsky: Direct skyline query with Dijkstra's algorithm
Psky: Skyline query with precomputing
SRsky: Skyline query with shortest range.

Data Availability

The datasets analyzed during the current study are available in the website https://figshare.com/articles/Urban_Road_Network_Data/2061897 or from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

The authors would like to thank the University of Shanghai for Science and Technology and Shanghai University of International Business and Economics for supporting this work and Dr. Nian X. Zhang for translating the manuscript. This work was supported by the National Natural Science Foundation of China (No. 61170277 and No. 61472256). The authors would like to thank Enago (www.enago.cn) for the English language review.

References

[1] A.-Y. Zhou, B. Yang, C.-Q. Jin, and Q. Ma, "Location-based services: architecture and progress," *Chinese Journal of Computers*, vol. 34, no. 7, pp. 1155–1171, 2011.

[2] W. Guo-feng, S. Peng-fei, and Z. Yun-li, "Review on development status and future of intelligent transportion system," *Highway*, vol. 05, pp. 217–222, 2012.

[3] S. Borzsonyi, K. Stocker, and D. Kossmann, "The Skyline operator," in *Proceedings of the 17th International Conference on Data Engineering*, pp. 421–430, IEEE, Heidelberg, Germany, April 2001.

[4] K. L. Tan, P. K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *Proceedings of the International Conference on Very Large Data Bases*, pp. 301–310, Morgan Kaufmann Publishers Inc., 2001.

[5] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: an online algorithm for skyline queries," in *Proceedings of the International Conference on Very Large Data Bases*, VLDB Endowment, pp. 275–286, 2002.

[6] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 467–478, June 2003.

[7] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 41–82, 2005.

[8] B. Chen and W. Liang, "Progressive skyline query processing in wireless sensor networks," in *Proceedings of the 5th International Conference on Mobile Ad-hoc and Sensor Networks*, pp. 17–24, December 2010.

[9] M. Sharifzadeh and C. Shahabi, "The spatial skyline queries," in *Proceedings of the International Conference on Very Large Data Bases (DBLP '06)*, pp. 751–762, Seoul, Korea, September 2006.

[10] K. Deng, X. Zhou, and H. T. Shen, "Multi-source skyline query processing in road networks," in *Proceedings of the 23rd International Conference on Data Engineering (ICDE '07)*, pp. 796–805, April 2007.

[11] Z. Frankenstein, J. Sperling, R. Sperling et al., "Processing continuous skyline queries in road networks," *International Symposium on Computer Science and ITS Applications*, pp. 353–356, 2008.

[12] Y.-K. Huang, C.-H. Chang, and C. Lee, "Continuous distance-based skyline queries in road networks," *Information Systems*, vol. 37, no. 7, pp. 611–633, 2012.

[13] M. Shekelyan, G. Jossé, and M. Schubert, "Linear path skylines in multicriteria networks," in *Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE '15)*, pp. 459–470, April 2015.

[14] P. S. S. Prabha and B. Jayanthi, "A nover secure location based skyline query processing," *International Journal of Engineering Sciences & Research Technology*, vol. 4, 2015.

[15] M. Safar, D. El-Amin, and D. Taniar, "Optimized skyline queries on road networks using nearest neighbors," *Personal and Ubiquitous Computing*, vol. 15, no. 8, pp. 845–856, 2011.

[16] S. Chang-yue, Q. Xiao-lin et al., "Location range-based skyline query in road networks," *Computer Science*, 2014.

[17] F. Shi-chang, D. Yi-hong et al., "Continuous probabilistic skyline queries based on road network for uncertain moving object," *Computer Science*, vol. 38, pp. 152–156, 2011.

[18] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[19] R. Kung M, E. Hanson, Y. Ioannidis et al., "Heuristic search in database systems," in *Proceedings of the 1st International Workshop on Expert Database Systems*, pp. 537–548, Benjamin-Cummings Publishing Co. Inc., 1986.

[20] R. Agrawal, S. Dar, and H. V. Jagadish, "Direct transitive closure algorithms: design and performance evaluation," *Association for Computing Machinery. Transactions on Database Systems*, vol. 15, no. 3, pp. 427–458, 1990.

[21] S. Dar and R. Ramakrishnan, "A performance study of transitive closure algorithms," *ACM SIGMOD Record*, vol. 23, no. 2, pp. 454–465, 1994.

[22] H. Hu, D. Lee L, and S. Lee V C, "Distance indexing on road networks," in *Proceedings of the International Conference on Very Large Data Bases (VLDB Endowment '06)*, pp. 894–905, 2006.

[23] S. Jung and S. Pramanik, "An efficient path computation model for hierarchically structured topographical road maps," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 5, pp. 1029–1046, 2002.

[24] R. Geisberger, P. Sanders, D. Schultes et al., "Contraction hierarchies: faster and simpler hierarchical routing in road networks," in *Proceedings of the Experimental Algorithms, International Workshop (Wea '08)*, pp. 319–333, Provincetown, MA, USA, June 2008.

[25] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou, "Shortest path and distance queries on road networks: towards bridging theory and practice," in *Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD '13)*, pp. 857–868, June 2013.

[26] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter, "Exact routing in large road networks using contraction hierarchies," *Transportation Science*, vol. 46, no. 3, pp. 388–404, 2012.

[27] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[28] T. Condie, N. Conway, P. Alvaro et al., "Online aggregation and continuous query support in MapReduce," in *Proceedings of the SIGMOD*, pp. 1115–1118, ACM Press, Indianapolis, IN, USA, June 2010.

[29] J.-H. Böse, A. Andrzejak, and M. Högqvist, "Beyond online aggregation: parallel and incremental data mining with online map-reduce," in *Proceedings of the Workshop on Massive Data Analytics on the Cloud (WWW '10)*, ACM Press, April 2010.

[30] Q. Kai-yuan, H. Yan-bo et al., "MapReduce intermediate result cache for concurrent data stream processing," *Journal of Computer Research and Development*, vol. 50, pp. 111–121, 2013.

[31] L. Dai-bo, H. Meng-shu et al., "Efficient dynamic algorithm for computation of shortest path tree," *Computer Science*, vol. 38, pp. 96–99, 2011.

[32] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proceedings of the 19th International Conference on Data Ingineering (ICDE '03)*, pp. 717–719, March 2003.

[33] https://figshare.com/articles/Urban_Road_Network_Data/2061897.