International Journal of Geo-Information

MDPI

Article

Efficient Geometric Pruning Strategies for Continuous Skyline Queries

Jiping Zheng [1,2,3,*], Jialiang Chen [1] and Haixiang Wang [1]

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China; chenjialiang@nuaa.edu.cn (J.C.); wanghaixiang@nuaa.edu.cn (H.W.)
[2] Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 211106, China
[3] School of Computer Science and Engineering, The University of New South Wales, Sydney, NSW 2052, Australia
* Correspondence: jzh@nuaa.edu.cn; Tel.: +86-25-8489-6490 (ext. 16116)

Academic Editors: Norbert Bartelme and Wolfgang Kainz
Received: 15 November 2016; Accepted: 19 March 2017; Published: 22 March 2017

Abstract: The skyline query processing problem has been well studied for many years. The literature on skyline algorithms so far mainly considers static query points on static attributes. With the popular usage of mobile devices along with the increasing number of mobile applications and users, continuous skyline query processing on both static and dynamic attributes has become more pressing. Existing efforts on supporting moving query points assume that the query point moves with only one direction and constant speed. In this paper, we propose continuous skyline computation over an incremental motion model. The query point moves incrementally in discrete time steps with no restrictions and predictability. Geometric properties over incremental motion denoted by a kinetic data structure are utilized to prune the portion of data points not included in final skyline query results. Various geometric strategies are asymptotically proposed to prune the querying dataset, and event-driven mechanisms are adopted to process continuous skyline queries. Extensive experiments under different data sets and parameters demonstrate that the proposed method is robust and more efficient than multiple snapshots of I/O optimal branch-and-bound skyline (BBS) skyline queries.

Keywords: continuous skyline queries; incremental motion model; geometric properties; event driven mechanisms; grid file index

1. Introduction

The skyline query [1,2] is a useful operation for many important applications, including multi-criteria optimal decision making. Given two certain and multi-dimensional tuples u and v, u dominates v *iff* u is no worse than v in all dimensions, and strictly better than v in at least one dimension. Due to the exponentially increasing usage of smartphones and the availability of inexpensive position locators, location-based services (LBS) are increasingly popular where skyline queries are based on the current location of the user, which changes continuously as the user moves. Taking an example of a tourist looking for restaurants, she/he may be interested in the restaurants close to her/his location that are cheap and have good reputations. Since the distances between the tourist and the restaurants are changing as the tourist travels, the skyline needs to be updated continuously. In addition, the tourist may have a destination in her/his mind and she/he moves towards the place. As shown in Figure 1, the user at time t_0 (i.e., at the place $q(t_0)$) moves in an upright direction (though users may move in arbitrary directions, in reality one user at a time only moves in one direction). At the next time t_1, the user arrives at the place $q(t_1)$. The ideal route may

be the red one. However, for various reasons, such as picking up a friend, carriage maintenance, and traffic congestion etc., the actual route may be the yellow or the blue one.

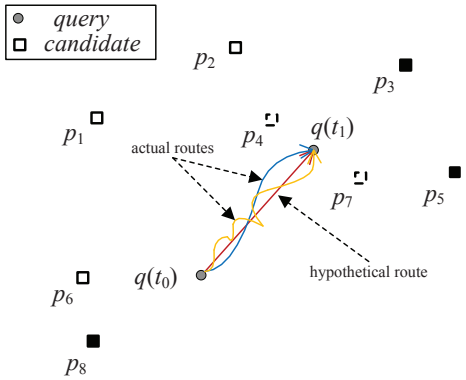

Figure 1. Routes for finding restaurants while moving.

In other real-time applications such as e-games and digital battle systems, the route between a player and her destination may not be straight. Instead, the route is tortuous, as in the routes in the previous restaurant finding example. When the fighting player moves, she should keep her eyes on those guardians who are close and most dangerous to her in terms of energy, weapon, etc. To find suitable restaurants during travel or to escape guardians while moving to the player's destination, the intuitive approach to updating the skyline query results is to recalculate the skyline results using efficient algorithms from scratch, such as branch-and-bound skyline (BBS) [3,4]. However, the relatively effective solution is to cache the last computed skyline results and to only calculate those data objects that may enter or leave the skyline results.

Note that existing approaches always assume that the motion of the query point is continuous and exactly calculable. Huang et al. [5] assumed that the query point was moving consecutively and the velocity of query point was known as (v_x, v_y). Lin et al. [6] and Guo et al. [7] considered the motion of the query point a to be a line, and Lin et al. [6] also assumed the query point was moving within a certain range. For location privacy consideration, the user query point is sometimes assumed to vary in a disk region [8]. In this paper, we address the motion typically presented incrementally over a series of discrete time steps, which is more practical for real applications.

Since discrete motion patterns are more suitable for moving points, we utilize the incremental motion model for the continuous skyline queries. That is, query points are moving incrementally in discrete time steps. Unlike existing motion models in [5–8], given the drift error bound and velocity of the drift, the incremental motion model places no restrictions on the motion or on its predictability (although the direction in this paper is given, we have no restrictions on moving directions). Under the incremental motion model, we utilize the geometric properties to prune the region in which the data objects will not be in the final skyline query results. To avoid calculating the skyline results from scratch while query points are moving, we maintain a data structure similar to kinetic data structures (KDSs) [9,10], which are famous in the area of computational geometry. The KDS keeps the desired relationship between data by storing all those data in some structures specific to the relationship. The contents in KDS do not change unless the relationship between some data points has been changed. The data structure includes a list for skyline query results at each time step. When the query point moves, the data structure decides whether the data object(s) enter(s) the skyline results or goes out of the result set. The implementation of the data structure is based on event-driven mechanisms. The framework of processing continuous skyline queries under an incremental motion model is illustrated as shown in Figure 2. First, the input dataset is pruned using geometric properties based on the incremental motion model. Then, we calculate the initial skyline results on the pruned dataset.

Event-driven mechanisms are adopted to compute the continuous skyline results when the query point is moving. To sum up, the key contributions are as follows:

- We adopt an incremental motion model for the continuous skyline queries which neither restricts the motion nor makes any predictions.
- By utilizing the geometric properties as a query point moves under incremental motion model, we prune skyline non-result-related data points, which can accelerate the processing of continuous skyline queries.
- Instead of accurate dominance, we propose probabilistic dominance under an incremental motion model for two data points which possibly dominate each other. By giving different thresholds of possible dominance, we decide the final query results, which are more actual in real applications.
- We demonstrate the efficiency of the geometric pruning strategies under an incremental motion model with extensive experiments on large-scale datasets.

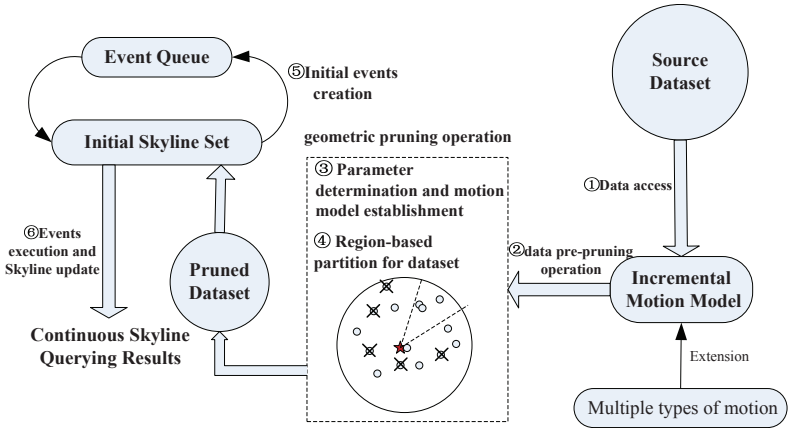

Figure 2. Continuous skyline processor.

The rest of the paper is organized as follows. Section 2 introduces related work. Preliminaries are given in Section 3. Section 4 presents the geometric pruning strategies. Data structures and event-driven processing mechanisms are provided in Section 5. Extensions to specific motion patterns are proposed in Section 6. Results of comprehensive performance studies are discussed in Section 7. Section 8 concludes the paper.

2. Related Work

Skyline Queries. The skyline operator was introduced to the database community by Borzsonyi et al. [1] in 2001. Consequent studies have focused on efficient skyline query processing. Tan et al. [11] developed bitmap and index techniques. Chomicki et al. [12] developed the sort-filter-skyline algorithm (SFS), which improved Block Nested Loop (BNL) by pre-sorting the dataset. Several optimizations to the SFS algorithm (e.g., [13]) increase its efficiency. Kossmann et al. [14] presented a nearest neighbor algorithm (NN) which allowed users to change preferences during runtime. Papadias et al. [4] proposed a progressive algorithm called branch-and-bound skyline (BBS), based on a nearest neighbor search technique supported by R-trees. Variations of the skyline operator have also been explored, such as in a distributed environment, road networks, skyline cubes, reverse skylines, and approximate skylines, just to name a few. See [2] and [15] for more extensions. However, the above studies only consider static query points on static attributes. Sharifzadeh et al. [16] first introduced the spatial skyline queries which are also named multi-source skyline queries in road networks by Deng et al. [17]. They differentiate the attributes of the data objects to two categories: spatial attributes and non-spatial attributes, which are also called dynamic and static dimensions or

static attributes and dynamic attributes. They also only considered static query points on dynamic attributes.

Motion Modeling and Skyline Processing. Another related area is monitoring continuous motions using kinetic data structures (KDS) in computational geometry. Basch et al. [9] first proposed a conceptual framework for KDS to continuously maintain evolving attributes of mobile data. A basic assumption in the KDS framework is that the object trajectories are known. More recently, several efforts have been made to deal with data in much less restricted models of motion. Mount et al. [18] studied the maintenance of geometric structures in a setting where the trajectories are unknown. They separated the concerns of tracking the points and updating the geometric structure into two modules: *the motion processor* (*MP*) and *the incremental motion algorithm* (*IM*). They further presented a simple online model in which two agents named *observer* and *builder* cooperated to maintain the incremental motion [19].

Motion considered in skyline queries includes a moving query point and moving data objects. Huang et al. [5] proposed a kinetic-based data structure to update the skyline results. The query point was moving along with predefined motion patterns (i.e., uniform motion in a straight line). Lee et al. [20] also studied a similar problem. However, both of the attempts rely on the assumption that the velocities of the moving points are known. Unfortunately, this assumption does not hold in many real-world applications where the points (e.g., cars, tourists) frequently change their motion patterns (e.g., speed and direction). Furthermore, the extension of their techniques is non-trivial for the scenarios where velocities are unknown. Lin et al. [6] assumed that the query point is in a predefined spatial range instead of an exact location or moving along with a line segment. To handle the movement of the querying objects, the incremental version of the line-based skyline solution has been devised to reduce both the result set size and the computation cost. Hsueh et. al [21] presented an algorithm to update the skyline when the data objects change their attribute values. Cheema et al. [22] proposed a safe zone-based approach for monitoring moving skyline queries which allows queries to move in an arbitrary fashion. The query results are required to be updated only when the query leaves its safe zone. In the framework, when a query point moves in the time interval between two adjacent events, its trajectory belongs to a safe zone. Vu et al. [23] introduced uncertainty both on the query point and Points Of Interest (POIs) for spatial skyline queries. The uncertainties here are represented as square disks with varied radii. Compared to this method, our incremental motion model considers not only the uncertainties (represented by the disk whose radius is the maximum speed), but also the directions. Another difference from these studies is that we prune the data points not belonging to the final skyline results based on geometric properties as a preprocessing step, which accesses less data points and thus saves more CPU time. The authors in [3,4,24–26] studied dynamic skylines. However, [3,4,24] used branch and bound method (BBS) to calculate the skyline results from scratch. References [25,26] utilized a caching mechanism to accelerate the dynamic/constraint skyline query processing. However, for continuous skyline queries, they are based on past queries not fully utilizing the query results of last moment. In addition, the methods of Sacharidis et al. [25] are not practical due to the shortcomings of bitmap coding mechanisms.

3. Preliminaries

When query points move, the distances between the query point and data points need to be estimated for further processing of skyline queries. In this section, we first estimate the location of the query point on incremental motion model, and then evaluate distance dominance between the data points for further calculating skyline results. Table 1 summarizes the notations frequently used throughout the paper.

Table 1. Summary of notations.

Notation	Meaning
$p_i \prec p_j$	p_i dominates p_j
$p_i \prec_{dist} p_j$	p_i dominates p_j in distance
p_t, p_i, p_j, q_s	data points, the center of the Euclidean ball
s_n, s_i, s_j, sk, sk'	skyline point
sp_f	the farthest skyline point
S_{sta}	the static partial skyline
S_{chg}	the volatile partial skyline
S,S_0	the complete skyline
$MB(M)$	the boundary of the motion M
L	vertical bisector of p_i and p_j
L_1, L_2	the tangent lines of an incremental motion model (a series of Euclidean balls)
LL	a bidirectional linked list to store skyline points
Q	a priority queue to hold events

3.1. Problem Definition

Given n data points in the dataset $\mathcal{D}$, each point has d-dimensional non-spatial attributes, also called static attributes. Each point p_i is stored as $(\mathbf{p}(p_i), p_{i,1}, p_{i,2}, ..., p_{i,d})$, and $\mathbf{p}(p_i)$, $p_{i,j}$ are the location of p_i and jth-dimensional static attribute of p_i, respectively. Thus, p_i can be represented as $p_i = (p_{i,1}, p_{i,2}, ..., p_{i,d}, p_{i,d+1})$, where $p_{i,d+1}$ is the distance of the query point q and data point p_i.

Definition 1. *(Static Dominance) For two data points p_i, p_j and all attributes of p_i except distance attribute, $\forall k, k = (1, 2, ..., d)$, $p_{i,k} \leq p_{j,k}$ and at least one $<$ holds, we say that p_i statically dominates p_j, represented as $p_i \prec_{sta} p_j$.*

Definition 2. *(Complete Dominance) For two data points p_i, p_j, $\forall k, k = (1, 2, ..., d+1)$, $p_{i,k} \leq p_{j,k}$ and at least one $<$ holds, we say that p_i completely dominates p_j, denoted as $p_i \prec p_j$.*

Although the skyline processing involves spatial and static attributes in our problem, some data points could be always in the skyline no matter how the query point moves. This is because these data points have the dominating nonspatial attributes which guarantee that no other data points can dominate them. We denote this subset of skyline points as $\mathcal{S}_{sta}$, the final skyline query results as $\mathcal{S}$, and the difference of the two sets $\mathcal{S} - \mathcal{S}_{sta}$, as $\mathcal{S}_{chg}$ for data points in $\mathcal{S}_{chg}$ might not be skyline points when query point moves. It is obvious that $\mathcal{S}_{chg}$ contains two parts: the first part is $\mathcal{S}_{in}$, which includes the data points from non-skyline dataset $\mathcal{D} - \mathcal{S}$. These points become skyline points when the query point moves. The second part is $\mathcal{S}_{remain}$, which contains the skyline points not moving out of $\mathcal{S}$ when the query point moves.

Instead of snapshot skyline queries like I/O optimal BBS method, we define the continuous skyline queries as follows.

Definition 3. *(Continuous skyline queries) Given the skyline results $\mathcal{S}_0$ at time t_0, the skyline result $\mathcal{S}_1$ at next time t_1 is based on $\mathcal{S}_0$ and only considers two varied sets: $\mathcal{S}_0 - \mathcal{S}_1$, the data points move out of skyline results of time t_0, and $\mathcal{S}_1 - \mathcal{S}_0$, the data points from $\mathcal{D} - \mathcal{S}_0$ become skyline points.*

From References [1,5,20,21], we know that the skyline result $\mathcal{S}_0$ is small compared with $\mathcal{D}$. However, the dataset $\mathcal{D} - \mathcal{S}_0$ is also too large. Fortunately, not all data points in this dataset have the possibility to become skyline points. In practice, according to the motion of query point q, only a very small part of $\mathcal{D} - \mathcal{S}_0$ can be skyline points. So, the computation overhead of the continuous skyline queries will be decreased to some extent compared with snapshot skyline queries.

3.2. Incremental Motion Model

The incremental motion model was first proposed by Mount et al. [18]. Later, Cho et al. [19] utilized this model to maintain net and net tree structures under an incremental motion model. We give a brief introduction to this model with adaptive modifications to our problem.

3.2.1. Query Point Position

Let q be a query point. The position of q in Euclidean space at time t is denoted as $\mathbf{q}(t)$. The motion of point q, $\mathcal{M}$, is a finite sequence of point positions sampled at discrete time instances: $\mathcal{M} =< \mathbf{q}(t_0), \mathbf{q}(t_1), ..., \mathbf{q}(t_N) >$, where $t_{i-1} < t_i$. The interval between two consecutive instances is a time step: $s = [t_{i-1}, t_i]$. Let v denote the estimation of a query point's current velocity. The estimated displacement of the point over this step is sv, and its actual displacement is given by the vector $\mathbf{u} = \mathbf{q}(t_i) - \mathbf{q}(t_{i-1})$. Let $|\mathbf{u}|$ denote the Euclidean length of vector $\mathbf{u}$. we use drift of this point at time t_i to represent the relative error between the actual and estimated displacements:

$$e_{drift} = \frac{|\mathbf{u} - s\mathbf{v}|}{|s\mathbf{v}|} \tag{1}$$

Let δ be the drift bound; we say that the motion $\mathcal{M}$ satisfies this motion estimate if for all time steps the drift of $\mathcal{M}$ relative to the velocity estimate v is at most δ. Given the velocity estimate v and given any time t, the estimated location of the point after an elapsed time of s is defined to be $\hat{\mathbf{q}}(t, s) = \mathbf{q}(t) + s\mathbf{v}$. For example, the estimated location of $q(t_{i-1})$ after step s is $\hat{q}(t_{i-1}, s)$ (see Figure 3a). Let $B(q, r)$ denote a Euclidean ball of radius r centered at point q. From the above definitions, we have

$$\mathbf{q}(t_i) \in B(\hat{\mathbf{q}}(t_{i-1}, s), s\delta|\mathbf{v}|) \tag{2}$$

Let T be a time interval of duration starting at time t, $T = [t, t + s]$. If a motion $\mathcal{M}$ satisfies a given motion estimation, then for each time instance $t + s' \in T$, $0 \leq s' \leq s$, the query point $q(t + s')$ lies within a Euclidean ball centered at $\hat{q}(t, s')$ and the boundary of $\mathcal{M}$ is determined by a series of the balls mentioned above (see Figure 3b). The following equation specifies the constraints on its position at any time:

$$\mathrm{MB}(M) = \bigcup_{1 \leq s' \leq s} B(\hat{\mathbf{q}}(t, s'), \delta s'|\mathbf{v}|) \tag{3}$$

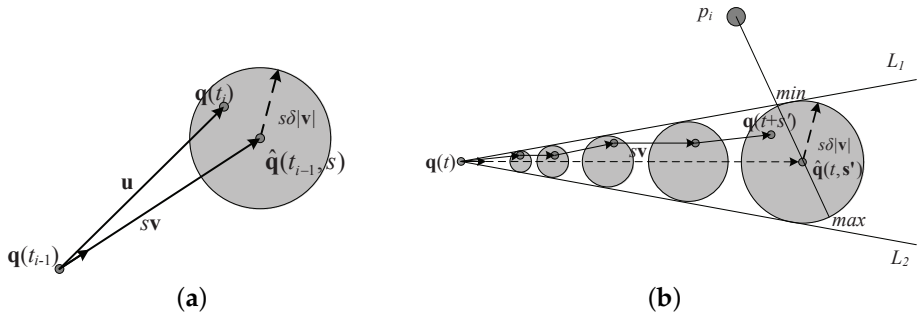

Figure 3. Incremental motion (IM) model. (**a**) Error bounds; (**b**) Location estimation.

3.2.2. Time Parameterized Distance Function

In our problem, the velocity of a query point is determined by the positions of the point at different time steps:

$$\mathbf{v} = \frac{\mathbf{q}(t_i) - \mathbf{q}(t_{i-1})}{t_i - t_{i-1}} \tag{4}$$

For simplicity, we assume that the query point q moves in $2d$ space, $v = (v_x, v_y)$, and the position of q at time t_i is $q(t_i) = (x_i, y_i)$. We use q_s to represent the center of the Euclidean ball. At time t_i,

$q_s = (x_i, y_i)$. After an elapsed time of $s = [t_i, t_{i+1}]$, the center of the incremental motion model q_s changes to (x_{i+1}, y_{i+1}) and equals to $(x_i + sv_x, y_i + sv_y)$. Then, for a data point p located at (x_p, y_p), at time t_{i+1}, the estimated distance between q_s and p can be expressed as follows:

$$\hat{D}(p, q_s) = \sqrt{(x_i + sv_x - x_p)^2 + (y_i + sv_y - y_p)^2} \tag{5}$$

By Equation (3), the estimated location of the query point is $q(t_{i+1}) \in B(\hat{\mathbf{q}}(t_i, s), s\delta|\mathbf{v}|)$. Since the estimated location of a query point after an elapsed time s is in a Euclidean ball, the actual distance between $q(t_{i+1})$ and p is constrained in a certain range (see Figure 3b). We use $I(p, q(t_{i+1}))$ to denote the distance between p and $q(t_{i+1})$:

$$I(p, q(t_{i+1})) = [\hat{D}(p, q_s) - s\delta|\mathbf{v}|, \hat{D}(p, q_s) + s\delta|\mathbf{v}|] \tag{6}$$

Besides, $dist(p, q(t_{i+1}))$ is used to denote the actual distance between p and $q(t_{i+1})$, which is in the range of $I(p, q(t_{i+1}))$ denoted by Equation (6).

3.3. The Dominance Relationship of Distance

As the query point moves constantly, the distance between the moving point and each data point changes, which is inaccurate because we cannot predict the actual position of the query point exactly. However, we are sure that the distance is in a certain range: $I(p_i, q_s)$. In addition, if $dist(p_i, q_s) < dist(p_j, q_s)$, p_i is closer to the query point than p_j. The dominance relationship between $dist(p_i, q_s)$ and $dist(p_j, q_s)$ is not confirmed, because both of them are inaccurate but within a known range. So, we need a rational method to compute the probability that p_i is better than p_j in distance.

Assume that after an elapsed time s, the position of a query point is shown in Figure 4, the radius of the incremental motion model is $s\delta|v|$, and L denotes the perpendicular bisector of p_ip_j which divides the ball into two regions: D_i and D_j. Then, we use A and B to denote the two intersections of L and the ball. When the query point q is within region D_i, $dist(p_i, q_s) < dist(p_j, q_s)$.

Lemma 1. *Assume that the position of query point q is uniformly distributed in the Euclidean ball. The angle between the estimated position of query point q to the intersections of the perpendicular bisector of p_ip_j and the bound of the incremental motion model is 2θ, then the probability that p_i is better than p_j in distance, denoted by $Pr(p_i <_{dist} p_j)$, can be calculated by*

$$Pr(p_i <_{dist} p_j) = \frac{\theta - \sin\theta\cos\theta}{\pi^2} \tag{7}$$

Proof. As long as the query point is in the region D_i, p_i is closer to the query point than p_j; consequently, the probability that p_i is better than p_j in distance is equal to the probability that the query point falls in the region D_i. Assuming that the possible positions of the query point are uniformly distributed in the ball, we only need to compute the proportion of region D_i to the whole ball.

The angle between the estimated position of query point q to the intersections of the perpendicular bisector of p_ip_j is 2θ, so the area of fan-shaped AqB is:

$$S(\triangledown AqB) = \frac{2\theta}{2\pi}\pi R^2 = \theta R^2 \tag{8}$$

where R denotes the radius of the ball.

In addition, the area of triangle ABq is:

$$S(\triangle ABq) = \frac{1}{2}|AB| \times |qC| = \frac{1}{2}2R\sin\theta R\cos\theta = \frac{1}{2}R^2\sin(2\theta) \tag{9}$$

So, the area of D_i can be expressed as follows:

$$S(D_i) = S(\triangledown AqB) - S(\triangle ABq) = \frac{(2\theta - \sin(2\theta)R^2}{2\pi} = \frac{(\theta - \sin\theta\cos\theta)R^2}{\pi} \tag{10}$$

Then, we can obtain that:

$$Pr(p_i <_{dist} p_j) = \frac{S(D_i)}{\pi R^2} = \frac{\theta - \sin\theta\cos\theta}{\pi^2} \quad (11)$$

□

From the above analysis we can see that with the movement of a query point the distance between the moving point and each data point is not certain, and is related to the estimated position of the query point. Next, we will give the definition of dominance in distance. Lemma 1 has shown how to compute the probability that p_i is better than p_j in distance.

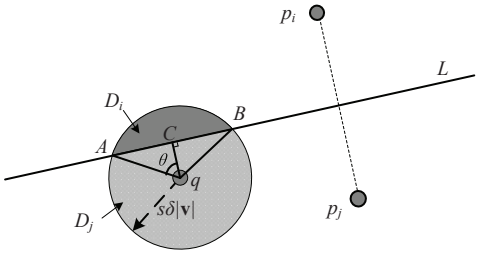

Figure 4. Dominance of distance for two data points.

Definition 4. *(Dominance in Distance) If the probability that p_i is better than p_j in distance is greater than a predefined threshold—That is, $Pr(p_i <_{dist} p_j) > \tau$—, we say that p_i dominates p_j in distance (i.e., $p_i \prec_{dist} p_j$).*

Definition 5. *(Skylines based on Thresholds) If $1 - \tau \leq Pr(p_i <_{dist} p_j) \leq \tau$ or $1 - \tau \leq Pr(p_j <_{dist} p_i) \leq \tau$, p_i and p_j cannot dominate each other in distance, and if no other data point can dominate p_i or p_j, they both belong to the skylines.*

The probability here is defined by the ratio of two areas (see Figure 4 and Equation (11)); the skyline queries are not probabilistic skyline queries as in Pei et al. [27] where probabilities denote the probable occurrence of each possible world from the PWS (possible world semantics) space composed by uncertain data objects.

4. Evaluating Skyline Changes under Incremental Motion

4.1. Pruning Using Geometric Properties

The final skyline results consist of the points which are not dominated by any other skyline, both in distance and all static dimensions. So, if p_i cannot dominate p_j in static dimensions, p_i cannot dominate p_j after considering the distance dimension. That is, p_i can dominate p_j only if p_i dominates or at least equals p_j in all static dimensions. Thus, we can use the static skyline results and the earliest spatial relations of the data points to minimize the scale of data and reduce unnecessary data accesses.

Lemma 2. *For a query point q at time t, if sp_f is the farthest point in $\mathcal{S}_{sta}$ to the query point q, then any point p_t that cannot dominate sp_f in distance is not in $\mathcal{S}$.*

Proof. Obviously, $p_t \notin \mathcal{S}_{sta}$, thus $\exists sp \in \mathcal{S}_{sta}$, and sp dominates p_t in static dimensions. Since sp_f is the farthest point in $\mathcal{S}_{sta}$ to the query point q, sp dominates sp_f in distance and sp_f dominates p_t in distance; thus, p_t is dominated by sp when considering distance and static dimensions, $p_t \notin \mathcal{S}$. □

Lemma 2 indicates a search bound when processing skyline queries. We can prune the portion of the unqualified data points before query processing: points that are dominated in distance by all points in $\mathcal{S}_{sta}$ can be eliminated. Furthermore, if the drift error bound δ and the estimated velocity of a query point $\mathbf{v}$ are given, the range of the ball in the incremental motion model is determined.

Lemma 3. *As shown in Figure 5, the tangent lines of the ball of the increment motion model are L_1, L_2 (see Figure 3b). Through query point q we draw lines H_1H_1', H_2H_2' vertical to L_1 and L_2, respectively. All these lines partition the entire area space into three parts: region A, region B, and the remaining area. Then, for any data point p in region B, if there exists a data point sk in region A and satisfies $sk \prec p$ while the query point resides in the region L_1qL_2, p cannot be in the final skyline.*

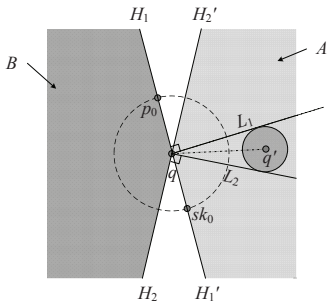

Figure 5. Pruning under IM Model.

Proof. Initially, p is dominated by sk, so sk dominates (or is equal to) p in all static dimensions, and $dist(sk, q) \leq dist(p, q)$. There may be two situations in the future:

1. sk is always in the skyline.
2. sk leaves the skyline at some time step.

For situation 1, consider the extreme case: sk lies on the boundary of region A (qH_1' or qH_2'), then we take q as the center and $dist(sk_0, q)$ as the radius to draw a circle (see Figure 5). We only need to prove that any data point p in region B or outside of the circle, if p is dominated by sk in initial, it will still be dominated by sk in the future. p_0 is one of the extreme cases: through point q we draw a line vertical to p_0sk_0 (i.e., the line L_1), because the whole estimated region of q is on the right side of the line, so sk_0 is nearer to the query point than p_0 in the future and sk_0 still dominates p_0.

For situation 2, if sk leaves the skyline, there exists a point $sk' \prec sk$; in this case, sk' dominates (or is equal to) sk in all static dimensions, and $dist(sk', q) \leq dist(sk, q)$. Referring to situation 1, $sk \prec p$ still holds, so $sk' \prec p$ and p can be safely pruned. □

We use Lemma 3 to divide a dataset into several regions, and verify each data point by the region it belongs to, then we prune the points that have no potential to enter the skyline $\mathcal{S}$ in the future.

Lemma 4. *On the basis of Lemma 3, we add following constraint: the angle of the incremental motion model $\alpha \leq 60°$ as shown in Figure 6a. L is the reverse extension line of the angular bisector of $\angle L_1qL_2$, H_2H_2' and L divide the entire area space into several parts, denoted by dark grey and light grey color. A data point p cannot be in the final skyline when the query point q still resides in the estimated region and one of the following two situations is satisfied:*

1. *p lies in region D_1 and is dominated by another point $\mathcal{S}$ in C_1.*
2. *p lies in region D_2 and is dominated by another point $\mathcal{S}$ in C_2.*

Note that D_1, D_2, C_1, and C_2 are trapezoidal regions in Figure 6.

Proof. From the above analysis, for a data point p lying in region D_1 we only need to prove that sk still dominates p in distance. Similar to the proof of Lemma 3, when $\alpha \leq 60°$, we draw the perpendicular bisector of p and sk, the estimated region of q is on the sk side of the perpendicular bisector. Figure 6 has shown an extreme situation. Because H_1H_1', H_2H_2 are the perpendicular bisectors of L_1 and L_2, respectively, $\alpha + \beta = \beta + \gamma$, then $\alpha = \gamma$. Obviously, $\gamma + 2\theta = 180°$. When $\gamma = 60°$, $p_0sk_0 \parallel H_2H_2'$,

so the perpendicular bisector of p_0sk_0 is the line L_2, as the estimated region of q is on the sk_0 side of L_2, sk_0 still dominates p_0 in the future. When p belongs to region $D_2 = area(LqH_2)$ symmetrically to D_1, similar to situation 1, p is dominated as shown in Figure 6b. □

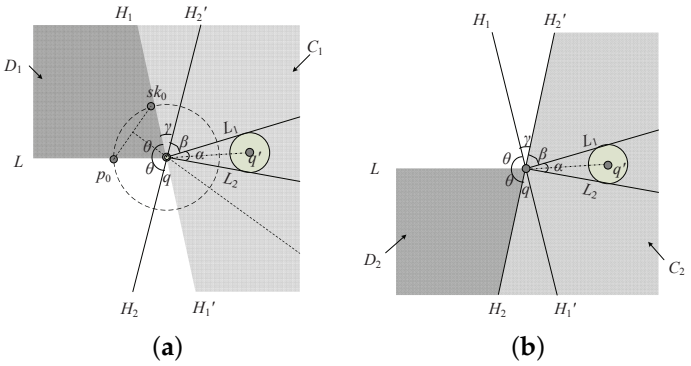

Figure 6. Sophisticated pruning. (**a**) D_1 is dominated; (**b**) D_2 is dominated.

Lemma 5. *As shown in Figure 7, let α be the angle of the incremental motion model, then we add extra angle α outside L_1, L_2, and draw two auxiliary lines L_1' and L_2'. After an elapsed time of s, assume that the estimated position of the query point is q'. If in initial any data point p in region P is dominated by a point sk in region S, p cannot be in the skyline in the elapsed time of s.*

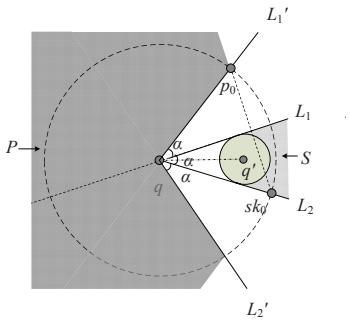

Figure 7. Pruning by prediction.

Proof. Similar to the proof of Lemma 3, we first draw the perpendicular bisector of p and sk and what we need to prove is that estimated region of q is on the sk side of the perpendicular bisector. Figure 7 shows the extreme case: draw the perpendicular bisector of sk_0 and p_0, sk_0 dominates p_0 in initial, in this situation, the perpendicular bisector goes through the start position of q. Additionally, $\angle L_1'qL_1 = \angle L_1qL_2 = \alpha$, so the perpendicular bisector is L_1. the estimated region of q is on the sk_0 side of L_1, sk_0 still dominates p_0 in the elapsed time of s. □

4.2. Change of Skyline under Moving Contexts

When the query point moves, the dominance of data points may change. As shown in Figure 8, at time t_1, $Pr(p_j <_{dist} p_i) = 1$, $p_j \prec_{dist} p_i$, and at time t_5, $Pr(p_i <_{dist} p_j) = 1$, $p_i \prec_{dist} p_j$. The distances to query point q of p_i and p_j overlap at time t_2, t_3, and t_4. Although the distance dominance relationship is uncertain at these moments, we can still compute the probabilities of dominance in distance.

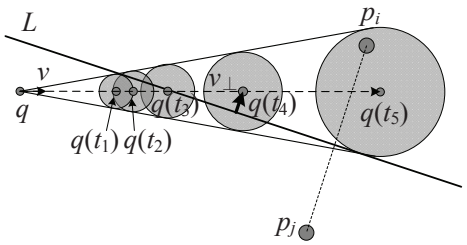

Figure 8. Change of distance dominance for two data points.

At moment t_3, the perpendicular bisector of p_ip_j goes through the center, $Pr(p_j <_{dist} p_i) = 0.5$, according to the threshold setting $0.5 \leq \tau \leq 1$, the dominating relation in distance is likely to change. At moment t_3, the relationship is determined by τ. Intuitively, we set $\tau = 0.5$, and t is the moment that distance dominance relationship between the two data points changes, called an intersection. A skyline point may leave the skyline after time t. On the other hand, a nonskyline point at time t may enter the skyline. In Figure 8, after time t_2 s_i must be dominated by a skyline point s_j. Those points that used to dominate the point before t_2 will stop dominating it. If $\tau > 0.5$, because of the change of the dominance relationship in distance, the time that p_j leaves the skyline will be later than t_2. Similarly, the time that p_i enters the skyline will be earlier than t_2. That is, both p_j and p_i will remain in the skyline as long as they cannot dominate each other in distance. If there is no intersection, the distance dominance relationship will remain unchanged. Whether an intersection will affect the skyline depends on which set p_i and p_j belong just before time t. Obviously, not every intersection causes the skyline to change. Table 2 shows possible dominance changes after an intersection.

We have the following theorems to describe these possibilities in detail.

Table 2. Possibilities for an intersection of two data points.

Possibility	Before Intersection	After Intersection
A	$p_i \prec_{dist} p_j$	$p_j \prec_{dist} p_i$
B	$p_i \prec_{dist} p_j$	*incomparable*
C	*incomparable*	$p_j \prec_{dist} p_i$

Theorem 1. *An intersection has no influence on the skyline if one of the following conditions holds before t:*

1. *In all conditions,* $p_i \in \mathcal{S}_{sta}$, $p_j \in \mathcal{S}$;
2. *In all conditions,* $p_i \notin \mathcal{S}$;
3. *In condition C,* $p_i \in \mathcal{S}$, $p_j \notin \mathcal{S}$;
4. *In condition B,* $p_i \in \mathcal{S}_{chg}$, $p_j \in \mathcal{S}$.

Note that condition A, B, C are shown in Table 2.

Proof.

1. If $p_i \in \mathcal{S}_{sta}$, it is obvious that p_i does not leave the skyline. As $p_j \in \mathcal{S}$, there are two situations. First, we assume that $p_j \in \mathcal{S}_{sta}$. Thus, p_j is still in the skyline and the skyline remains unchanged. Second, we assume that $p_j \in \mathcal{S}_{chg}$. Since $p_j \in \mathcal{S}$, there exists no point p that dominates p_j. Before time t, the points which have potential to dominate p_j do not intersect with p_j. So, p_j is still in $\mathcal{S}_{chg}$ and causes no change to the skyline.
2. Since $p_i \notin \mathcal{S}$, before time t there must be at least one point $sk \in \mathcal{S}$ in the skyline dominating it. The intersection has no influence on $sk \prec_{dist} p_i$ if $p_j \in \mathcal{S}_{sta}$ and p_j will not leave the skyline. Additionally, if $p_j \in \mathcal{S}_{chg}$, for the same reason as in item (1), p_j will stay in the skyline. If $p_j \notin \mathcal{S}$, as in item (1).

3. Since $p_i \in \mathcal{S}$, there are two situations: $p_i \in \mathcal{S}_{sta}$ and $p_i \in \mathcal{S}_{chg}$. We first assume $p_i \in \mathcal{S}_{sta}$. Then, p_i is still in the skyline after the intersection. Then, we assume $p_i \in \mathcal{S}_{chg}$. On one hand, because $p_j \notin \mathcal{S}$, p_j cannot dominate p_i in all static dimensions, and p_i is still a skyline point after the intersection; on the other hand, $p_j \notin \mathcal{S}$ and p_i and p_j cannot dominate each other in distance, so p_j is unable to dominate p_i, there must be a point $p \in \mathcal{S}$ which dominates p_j. Therefore, there is no intersection between p and p_j, $p \prec p_j$ and $p_j \notin \mathcal{S}$.
4. For the same reason as in item (1), p_j is still in the skyline after the intersection. $p_i \in \mathcal{S}_{chg}$, assume that p_j dominates p_i in all static dimensions. After the intersection, p_j and p_i cannot dominate each other in distance, so p_j cannot dominate p_i. Thus, $p_i \in \mathcal{S}_{chg}$.

□

Theorem 2. *An intersection may have influence on the skyline if one of the following conditions holds before t:*

1. *In condition A or B, $p_i \in \mathcal{S}$, $p_j \notin \mathcal{S}$;*
2. *In condition A or C, $p_i \in \mathcal{S}_{chg}$, $p_j \in \mathcal{S}$.*

Note that conditions A, B, and C are shown in Table 2

Proof.

1. First, assume that $p_i \in \mathcal{S}_{sta}$. So, p_i is still in the skyline after the intersection. Since $p_j \notin \mathcal{S}$, there must be at least one skyline point in $\mathcal{S}$ dominating it. If p_i dominates p_j before t, then after t there may be one of the following situations: $p_j \prec_{dist} p_i$ or p_i and p_j cannot dominate each other in distance. Consequently, p_j will enter the skyline after t in both situations.
2. Obviously, p_j will not leave the skyline after t. $p_i \in \mathcal{S}_{chg}$, $p_j \in \mathcal{S}$, if p_j dominates p_i in all static dimension, after t, p_i will leave the skyline since $p_j \prec_{dist} p_i$.

□

According to Theorem 2, we only need to consider the two following cases in which the skyline may change:

1. In initial, $s_i \in \mathcal{S}$, $s_n \notin \mathcal{S}$, and $s_i \prec s_n$. After an intersection, s_i can no longer dominate s_n, then s_n can enter the skyline and depends on whether s_i is the unique skyline point that dominates it.
2. Initially, $s_i \in \mathcal{S}_{chg}$, s_j dominates s_i in all static dimensions. After an intersection, s_i can be dominated by s_j and leave the skyline.

5. Skyline Computation

5.1. Continuous Skyline Query Processing

We now explore the continuous skyline query processing techniques according to the above analysis. A naive way is to call the existing algorithms to process skyline queries in each time step to acquire continuous query results. Since we know the estimated position of the query point and parameters of the associated incremental motion model, the results are credible. However, processing continuous skyline queries in this way needs to traverse the entire dataset repeatedly, which will inevitably increase the running time and I/Os. In practice, the speed of a query point is not very fast; therefore, the skyline will not change frequently. For example, as mentioned in Section 1, a user is looking for restaurants. Her/his moving speed is considerable not too fast. So, the skyline results in the last moment can be utilized to process skyline queries of the next moment. Otherwise, if the speed is too fast or the time interval is large enough that the skyline results in the last time step are all changed at the next moment, we would rather utilize snapshot skyline queries for this situation. For the motivation examples, these will not occur. So, in this paper, we assume the query points are not moving too fast and skyline results in last moment can be used for next time step query

processing. For the problem, we can use the strategies mentioned in previous sections to compute the intersections that may influence the skyline and maintain the skyline results incrementally.

First, we compute the initial skyline. After that, we decide which moment may cause the skyline change and record the intersections the skyline results may change. Then, when an intersection comes, we deal with it and determine further intersections for the updates of the skyline.

Corollary 1. *Assume that the distances of skyline points s_i, s_j, and s_k to the estimated query point is an increasing sequence, then the intersection between s_i and s_k will not occur before an intersection between any two adjacent points of s_i, s_j, and s_k.*

Proof. Assume that s_i and s_j intersect at time t_x, s_j, s_k, and s_i, s_k intersect at time t_y, t_z, respectively. Before an intersection, we know that $s_i <_{dist} s_j <_{dist} s_k$, and we only need to prove that t_z must be later than t_x and t_y. Now suppose that t_z is earlier than t_x or t_y. So, after time t_z, $s_k <_{dist} s_i$. Additionally, $s_i <_{dist} s_j$ is still valid since no intersection happens between any two adjacent points, which contradicts $s_k <_{dist} s_i$. Therefore, t_z must be later than t_x and t_y. □

We stored the skyline in a sequence according to their distances to the query point. We also only need to compute intersections between two adjacent skyline points.

5.2. Data Structure and Conditions

We use a bidirectional linked list (other similar data structures, such as heap and array are also fine for we process these structures not "on-line"; instead, they could be processed during the interval of two time steps) named LL to store current skyline points, which are sorted in ascending order of their distances to the query point. The form of each skyline point s_i in LL is denoted as $(flag, dist, t_{valid}, t_{skip})$. $flag$ indicates whether s_i belongs to $\mathcal{S}_{sta}$, $dist$ is the distance between s_i and the query point q, t_{valid} is the validity time of s_i, which is only available to each changing skyline point and recording the time when s_i is dominated, and t_{skip} is the time s_i will exchange its position with its successor in LL (see the algorithms below for details).

By Theorem 2, there are two situations that may cause the skyline to change. Assume the time of an intersection is t_{insec} ($\tau = 0.5$):

1. Before time t_{insec}, s_i is a changing skyline point. s_j is farther to query point q than s_i, and s_j dominates s_i in all static dimensions. Then, after t_{insec}, s_i will be dominated by s_j and leave the skyline;
2. Before time t_{insec}, s_i is a skyline point, and s_n is a nonskyline point. Then, after t_{insec}, s_n can enter the skyline depending on whether s_i is the unique skyline point which dominates it.

To summarize the above analysis, we only need to consider the cases which may cause the skyline to change. For simplicity, this paper has made two assumptions on the threshold τ:

1. The perpendicular bisector of p_ip_j goes through the centre of the ball in the incremental motion model, so $\theta = 90°$, we can derive from Lemma 1 that:

$$Pr(p_i <_{dist} p_j) = \frac{2\theta - \sin(2\theta)}{2\pi} = 0.5$$

   For $\tau = 0.5$, before time t_2, $p_j \prec_{dist} p_i$, and after that $p_i \prec_{dist} p_j$ (as shown in Figure 8).
2. Assume that the perpendicular bisector of p_ip_j goes through the point C where is $1/4$ of the diameter (see Figure 4); i.e., $|qC| = 1/2|qA|$. Therefore, $\cos\theta = 0.5$, and $\theta = 60°$; we can obtain from Lemma 1 that:

$$Pr(p_i <_{dist} p_j) = \frac{2\theta - \sin(2\theta)}{2\pi} = \frac{1}{3} - \frac{\sqrt{3}}{4\pi}$$

For convenience of calculation, we set $Pr(p_i <_{dist} p_j) = \tau$.

Theorem 3. *As shown in Figure 8, assume that the position coordinates of p_1, p_2 are (x_1, y_1) and (x_2, y_2). The query point is starting from (x_q, y_q) with velocity $\mathbf{v}(v_x, v_y)$, then the time of the intersection can be presented as below:*

1. $\tau = 0.5$, $t = \frac{y_q - kx_q - C}{kv_x - v_y}$
2. $\tau = \frac{1}{3} - \frac{\sqrt{3}}{4\pi}$, *time of the intersection are t_2 and t_4, t_2 and t_4 can be given as follows:* $t_2 = \frac{kx_q - y_q + C}{\delta|v|\sqrt{k^2+1} - kv_x + v_y}$, $t_4 = -\frac{kxq - yq + C}{\delta|v|\sqrt{k^2+1} + kv_x - v_y}$ *where* $k = \frac{x_1 - x_2}{y_2 - y_1}$, $C = \frac{y_1 + y_2}{2} - k\frac{x_1 + x_2}{2}$.

Proof. The coordinates of p_1, p_2 are (x_1, y_1) and (x_2, y_2), the perpendicular bisector of p_1p_2, denoted by L, can be written as: $y - \frac{y_1+y_2}{2} = \frac{x_1-x_2}{y_2-y_1}(x - \frac{x_1+x_2}{2})$; that is, $\frac{x_1-x_2}{y_2-y_1}x - y + (\frac{y_1+y_2}{2} - \frac{x_1-x_2}{y_2-y_1}\frac{x_1+x_2}{2}) = 0$. Let $k = \frac{x_1-x_2}{y_2-y_1}$, $C = \frac{y_1+y_2}{2} - k\frac{x_1+x_2}{2}$:

1. If $\tau = 0.5$, we only need to compute when the query point will meet the perpendicular bisector of p_1p_2; that is, the moment p_1 and p_2 is equal to the distance to the query point: $(x_q + tv_x - x_1)^2 + (y_q + tv_y - y_1)^2 = (x_q + tv_x - x_2)^2 + (y_q + tv_y - y_2)^2$, then $t = \frac{y_q - kx_q - C}{kv_x - v_y}$
2. If $\tau = \frac{1}{3} - \frac{\sqrt{3}}{4\pi}$, at time t_2 and t_4, the query point satisfies the condition of 1/4 of the diameter (see Figure 4) , time t_1 and t_5 are the tangential moments of the perpendicular bisector and the bound of the ball in the incremental motion model. We derive time t_4 via t_3 and t_5; t_2 is similar. The vertical speed of the perpendicular bisector is denoted by $v_\perp$. At time t_5, distance from the query point to L is equal to the current radius of the ball in the motion model, so we have $\frac{|k(x_q + v_x t_5) - (y_q + v_y t_5) + C|}{\sqrt{k^2+1}} = \delta|v|t_5$. According to (1), $t_3 = \frac{y_q - kx_q - C}{kv_x - v_y}$, then $v_\perp$ can be obtained by $v_\perp = \frac{\delta|v|t_5}{t_5 - t_3}$. Therefore, from t_3 to t_4, the distance varies along the direction of $v_\perp$ is equal to half of the radius. So, $v_\perp(t_4 - t_3) = \frac{1}{2}\delta|v|t_4$.

Based on the above equations, as a conclusion, $t_4 = -\frac{kx_q - y_q + C}{\delta|v|\sqrt{k^2+1} + kv_x - v_y}$. Similarly, $t_2 = \frac{kx_q - y_q + C}{\delta|v|\sqrt{k^2+1} - kv_x + v_y}$. □

As the query point moves, the distances between all data points and the query point are varying, which may cause the skyline to change. According to the type of the change, three events are formulated as follows:

- Event *exit*. This occurs when any skyline point leaves the skyline, which will only happen to a volatile skyline. Assume that $s_i \in \mathcal{S}_{chg}$, and there is another skyline point s_j with potential to dominate s_i, then if s_i intersects with s_j in distance and $Pr(s_j <_{dist} s_i) > \tau$, s_i will leave the skyline; that is, an *exit* event happens.
- Event *in*. This occurs when any nonskyline point enters the skyline. For a nonskyline point s_n and all those skyline points currently dominating it, if s_n gets closer to query point q than skyline point s_i, s_i can no longer dominate it; that is, an *in* event happens. However, whether it will enter the skyline depends on whether s_i is the only one to dominate it. This will be checked when an event of this kind is being processed.
- Event chg_{ord}. This occurs when a couple of skyline points in LL make a sequential change. For a skyline point s_i, if it intersects with its successor s_j and s_j cannot dominate it, s_i and s_j exchange positions in LL; that is, a chg_{ord} event happens. Notice that s_j does not have the potential to dominate s_i; otherwise, an *exit* event will happen instead.

As shown in Figure 9, the list includes $\{sk_1, sk_2, sk_3, sk_4\}$ data points, and the points are sorted in ascending order of their distances to the query point. At time t, $Pr(sk_3 <_{dist} sk_2) > \tau$, sk_3 dominates sk_2 in all static dimensions. Then, an *exit* event will happen because sk_2 will be dominated and

leaves the skyline (see Figure 9b). If s_n is the skyline point dominated by sk_3 uniquely, at time t, if $dist(q, sk_3) > dist(q, s_n)$, then an *in* event will happen because s_n will enter the skyline (see Figure 9c). Additionally, at time t, if sk_3 gets closer to query point q than sk_2 and sk_3 has no potential to dominate sk_2, then an chg_{ord} event will happen because sk_2 and sk_3 will exchange their positions in the list (see Figure 9d).

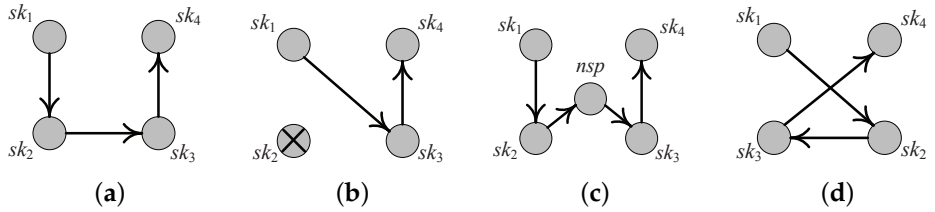

Figure 9. Processing for three events. (**a**) Original event sequence; (**b**) *Exit* event; (**c**) *In* event; (**d**) *Change* event.

A global queue is used to maintain all events to represent future skyline changes. Each event is in the form of $(time, type, self, rel)$ when the event happens at time $time$, and $type$ is used to record the kind of this event. $self$ and rel respectively represent the skyline point and the relevant data point involved in the event. In an *exit* or chg_{ord} event, $self$ represents the skyline point s_i, while *peer* is its successor s_j. In an *in* event, $self$ represents the skyline point while *peer* stands for the relevant nonskyline point s_n.

Initially, LL contains all the current skyline points while Q contains recent events that will happen in the nearest future. As time elapses, events in the queue are dequeued and handled according to their types. While handing events and updating the skyline, the process also incurs future incoming events. Therefore, Q evolves with existing events being dequeued and new events enqueued. After all due events are processed, LL contains all the correct skyline points with respect to the query point q's current position.

5.3. Event-Driven Mechanisms for Continuous Query Processing

In our method for the static data set, we use a simple 2D *grid file* index dividing the data space into $h \times v$ cells. We set the data points within each cell are stored in one disk page. At the beginning of the algorithm, the static skyline will be computed in advance. According to Lemma 2, the farthest distance is recorded in variable d_{farest} as a search bound, and the cells beyond d_{farest} are pruned for reducing I/Os (see Figure 10).

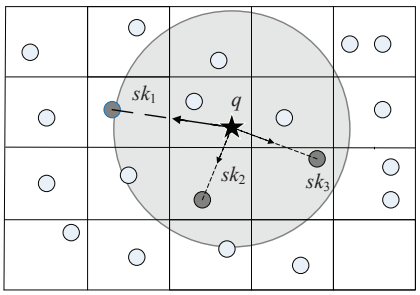

Figure 10. Pruning with the grid file index.

As shown in Algorithm 1, in initial, all permanent skyline points in $\mathcal{S}_{sta}$ are inserted into LL based on their distance to the starting position of query point q. First, we prune the dataset by utilizing geometric properties. Then, starting from the cell where q's initial position lies, all grid cells are searched in a spiral manner so that those on an inner surrounding circle are searched before those

on an outer one, as shown in Figure 10. Then, we organize the data set according to Lemma 3, 4, 5 with heap structure. A heap H is used to store cells or data points that are possible to enter the skyline, which its top is the cell or the data point which is closest to q's estimated position. Points or cells in the heap are sequentially compared to the current skyline points in LL, which is adjusted with deletion or insertion if necessary. After that, events will be created for continuous skyline query—all events for all skyline points, except the last one in LL. Next, the farthest skyline point is applied to compute possible *in* events for those points farther than it.

Algorithm 1: Initialization

Input: the position of a query point q
Output: the initial skyline IS for q, the event queue Q with initial events

```
1  According to S_sta, insert an entry (1, dist, ∞, ∞) into LL;
2  Search bound determined by d_farest = (LL.last, q);
3  Prune the dataset by S_sta;
4  Insert the cell where q's position lies into H
5  Scan the grid file from where q lies;
6  while !Empty H do
7      pop(H), pop and process the top entry e;
8      if e is a data point then
9          compare with all the current skyline points in LL;
10         if Not dominated then
11             Insert e into LL;
12     if s is a cell then
13         if Not dominated then
14             Insert child entries of e into H sequentially;
15     for each cell cell_i on next outer surrounding circle do
16         if dist(cell_i, q) < d_farest || cell_i is not dominated then
17             insert cell_i into H
18         else
19             break
20 for each s_i from LL.last.prev to LL.first do
21     CreateEvents(s_i, q);
22 Compute possible in events with LL.last;
```

Algorithm 2 has shown the process of *CreateEvents* in detail. For a given skyline point s_i, the algorithm first computes the time t when s_i and the next skyline point s_j in LL will exchange their positions in the list that s_j will dominate s_i in distance. If t is later than s_j's exchange time or s_i's validity time, it is ignored. Otherwise, it means an *exit* event depending on s_j's validity time if $s_i \in \mathcal{S}_{chg}$, or it is a simple chg_{ord} event. Then compute *in* events for each nonskyline point s_n that distance to q's estimated position comparing with s_i and s_j's distances.

When the nearest event in Q happens, it is dequeued and processed with the relevant points involved according to its type. Then create new events after the new skyline is obtained (as shown in Algorithm 3). At any time when Q is empty, all the points in LL are the correct skyline of the current time point.

According to Algorithm 3, the actions to process each kind of event are described as follows: for an exit event, s_i is removed from the skyline list LL and creates new events for its predecessor since the successor has been changed; for an *in* event, the nonskyline point s_n will be checked to see whether it is unique and dominated by s_i. If yes, s_n will be inserted into the skyline list LL and new events are computed for relevant points. Otherwise, a possible new *in* event is computed and enqueued. For a chg_{ord} event, the skyline list is correctly adjusted by exchanging the positions of s_i and s_j. Similarly, relevant events are created and enqueued for them and their predecessors if exists.

Algorithm 2: CreateEvents

Input: a skyline point s_i in LL
Output: events for s

```
1  rel = NULL;
2  s_j = s_i.next; // s_j is the successor point of s_i in LL_i;
3  t = time s_j and s_i will intersect in distance to q;
4  if t < s_j.t_skip and t < s_j.t_valid then
5      if s_i ∉ S_sta then
6          if (t < s_j.t_skip) and (t < s_j.t_valid) then
7              rel = s_j;
8              s_i.t_valid = t;
9              EnQueue an exit event to Q;
10     else
11         s_i.t_skip = t;
12         EnQueue a chg_ord event to Q;
13 for each s_p from s_i to LL.first do
14     t is time nonskyline point p get closer to q than s_p;
15     if (t > s_p.t_valid) or (t > s_p.t_skip) then
16         continue;
17     if ∀s in LL, s ⊁ p then
18         Enqueue an in event to Q;
19         break;
```

Algorithm 3: Process due event

Input: the current time *t_cur*
Output: the updated skyline $\mathcal{S}$, updated list LL

```
1  while !empty Q & Q.front.time == t_cur do
2      Dequeue Q.front process due event based on type;
3      if type == exit then
4          Delete s_i in LL;
5          CreateEvent for s_i.prior;
6      if type == chg_ord then
7          exchange positions of s_i and s_j in LL;
8          CreateEvent for relevant points;
9      if type == in then
10         Insert s_n into LL and before s_i;
11         CreateEvent for relevant points;
12     else
13         EnQueue a new in event to Q;
14     DeQueue;
```

6. Pruning Strategies for Specific Motion Patterns

The geometric pruning strategies can be easily integrated into the known moving trajectories to reduce the number of data points. In this section, we provide the geometric pruning strategies under the incremental motion model for three different classes of trajectories: enclosed, bounded, and unbounded. In the case that the movement or part of the moving range boundary of a query point is known, if the position of the point is now not qualified to establish an incremental motion model, we assume that there is a "virtual query point" somewhere in the past, and it was able to be applied to the incremental model. Moreover, the movement of the true query point—though not consistent with the incremental motion model—stays in the estimated region of the incremental motion model set up by the virtual query point.

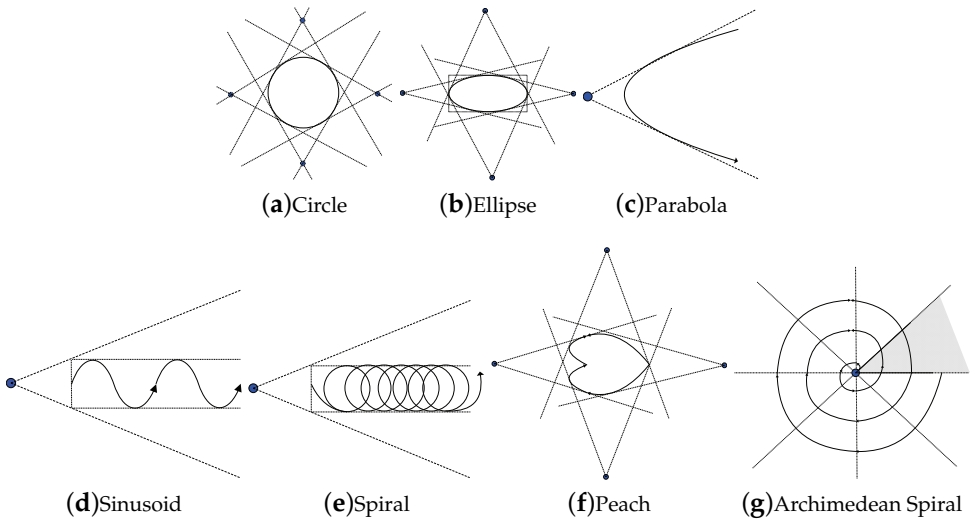

Figure 11. Motion Patterns: (**a**–**c**) Enclosed; (**d**,**e**) Bounded; (**f**,**g**) Unbounded.

Enclosed motion patterns If the movement of a query point q is an enclosed curve, the query point stays in the enclosed region surrounded by the curve. Figure 11a–c are three typical enclosed motion patterns: *circle*, *ellipse*, and *parabola*. In this case, in spite of q's moving speed, the point moves along the curve and makes an influence to the skyline repeatedly. In consideration of the characteristics of enclosed curves, the exact upper and lower bounds in all directions of a query point can be obtained. Based on all of the above curves, we can establish incremental motion models in different directions to acquire a series of virtual query points; after that, the geometric pruning algorithm will be executed iteratively, and filtering out most of the redundant points which make no influence to the skyline.

Motion patterns with bounds If the movement of a query point q is a curve with bounds, it means that we can acquire its upper and lower bounds in one or several directions while the rest have no boundaries or could not be predicted (*Sinusoid* and *Spiral* are two of this kind of motion pattern, as shown in Figure 11d,e). According to the property of these curves, we can still utilize the starting position of the query point and the directions that bounds can be obtained to establish an incremental motion model, get a virtual point, and apply to the geometric pruning strategies. Basically, though the effect of filtering is not as efficient as the situation of enclosed patterns, it is still worth executing compared to snapshot skyline queries.

Motion patterns without bounds If the movement of a query point q is a curve without bounds, then no bounds can be gained in each direction and it is infeasible to use the geometric pruning strategies based on incremental motion model (examples of this kind of motion pattern are *peach* and *Archimedean spiral* in Figure 11f,g).

According to the characteristics of different types of trajectories, the curves can be classified as follows:

1. There exists an upper or lower bound in one or several directions (prerequisite). For this kind of curve we can try to establish an incremental motion model by making a pair of tangent lines (e.g., *parabola*, *logarithmic* curves). The qualifications of tangent lines are:
   (a) There exists an upper or lower bound in some direction(s).
   (b) The pair of tangent lines will intersect and generate a virtual query point.
   (c) The angle between the pair of tangent lines satisfies the demands of the geometric pruning framework.
2. There exists no bound in all directions or it is unable to acquire a pair of qualified tangent lines in (1). In this case, we cannot adopt the geometric pruning strategies directly. Then, we need

to take the characteristics of the curves into account and adapt it to the framework by adding additional restrictions.

7. Experiments

A näive approach to monitoring moving skylines is to call existing algorithms such as I/O optimal BBS [3,4] to recompute the skyline whenever the results need to be updated. In this section, we compare the näive algorithms to the proposed methods against various factors which may potentially affect the performance of the algorithms. All the algorithms are implemented in standard C++ with STL library support and compiled with GNU GCC 4.9.3. Experiments were run on a PC with Intel Core i3-3240 3.40GHz dual CPU and 4G memory running Ubuntu Linux 14.04 LTS. The disk page size was fixed to 4096 bytes.

To generate datasets for the experiments, we first fetched real-life California's interesting points from the website [28] (see Figure 12), then we combined the real locations with nonspatial dimensions following different distributions: Independent, Correlated, Anti-correlated, and Zipf. The data size of California's real locations is about 100 K, and we generate nonspatial dimensions like that in reference [1]. The attribute value of a data point varied from 1 to 1000. CPU time and I/O counts were used to measure the efficiency of the algorithms under 100 runs of skyline queries. The concerned parameters used in the experiments are listed in Table 3. In particular, the following algorithms were evaluated:

- Continuous skyline query (CSQ): Incremental model-based continuous skyline method which performs the skyline change tracing algorithm in reference [5] directly without using the geometric pruning strategies.
- GP-CSQ: Method performing the skyline change tracing algorithm combined with the geometric pruning strategies using the Lemma 3 (Lemma 4, its extension) only.
- GP2-CSQ: An instance of the method performing the skyline change tracing algorithm combined with the geometric pruning strategies using both Lemma 3 (Lemma 4) and 5.

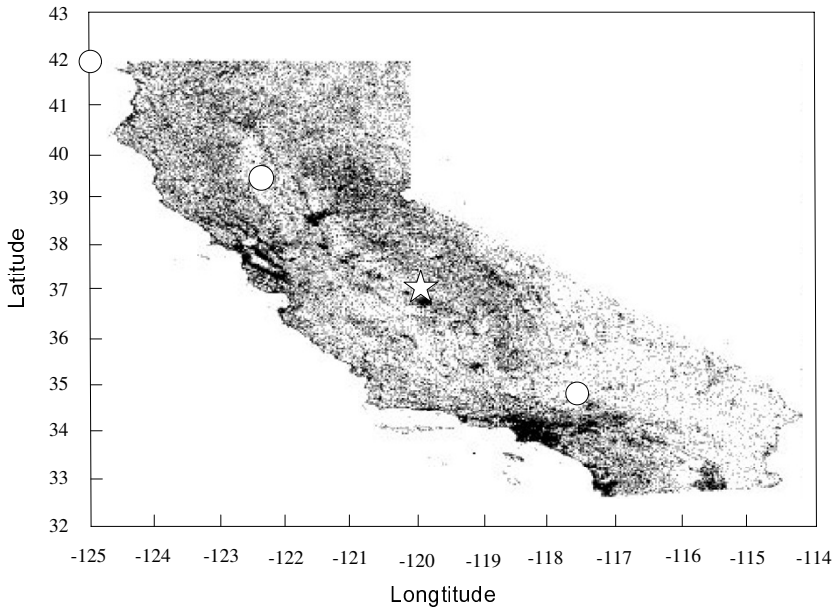

Figure 12. California's points of interest (circle dots and the star denote the starting positions).

Since the BBS algorithm is the most efficient method for computing skyline in static settings (both data points and query point are static), we adopted it for comparison in the experiments. When the location of the query point changed, we only modified the "*mindist*" to adapt the basic

BBS algorithm (see [4]). Besides, we also used the method which is called "ex-BBS" in [5] for contrast to our proposed methods.

Table 3. Parameter and ranges (Default value *).

Parameter	Range
Cardinality	10 K, 20 K, 30 K, 50 K, 100 K*
Static Attributes	2*, 3, 4, 5
Starting Position	(42° N, –125° W), (39.5° N, –122.5° W) (37 °N, –120° W)*, (34.5° N, –117.5° W)
Moving Speed	(1,-1)*, (2,-2), (4,-4), (8,-8)
Error Bound	0.2, 0.35, 0.5*, 0.65

Effect of Cardinality. To generate different-sized datasets, we randomly selected part of the real locations and then combined with two synthetic nonspatial dimensions. Thus, we converted the size of the datasets from 10 K to 100 K. Then, we executed 100 (10 for anti-correlated) continuous skyline queries on the datasets. For each query, we set the starting position of the query point as (37° N,–120° W). The default speed of the query point was $(1, -1)$, while the moving direction was the same as vector $(1, -1)$. The threshold was fixed to be 0.5.

In Figure 13, the CPU cost of the original CSQ algorithm is higher than BBS algorithms in some cases, because CSQ not only processes non-skyline objects but also computes the initial events to maintain the skyline in the future; GP-CSQ and GP2-CSQ were faster in general because the geometric pruning policies can filter out a large number of unqualified data points and cut down the CPU cost of event computing. Note that in Figure 13d, CSQ takes much time and the size of the skyline results is large because anti-correlated datasets incur more events. Figure 14 shows that as cardinality increases, the I/Os of CSQ, GP-CSQ, and GP2-CSQ are nearly 10% less than that of BBS, while GP-CSQ and GP2-CSQ are a little better than CSQ algorithm.

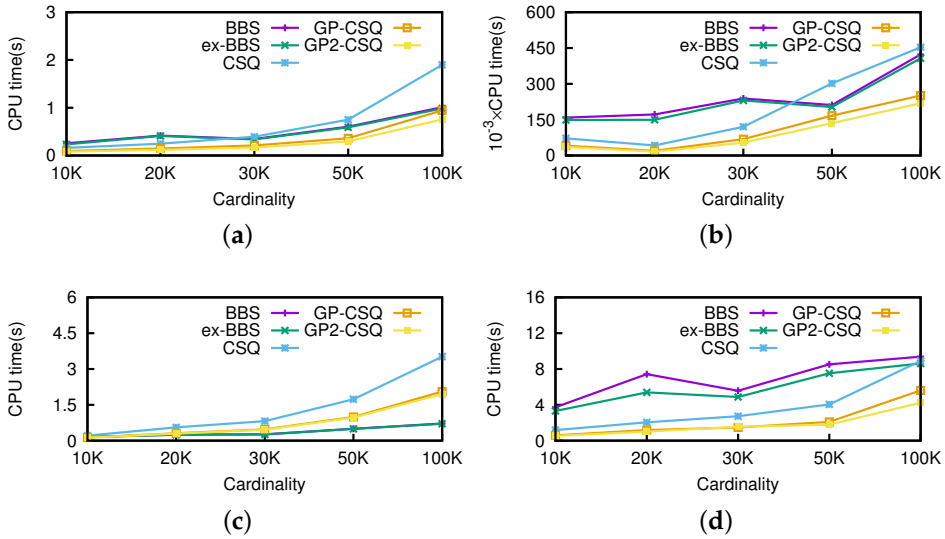

Figure 13. Effect of cardinality (CPU time). BBS: branch-and-bound skyline; CSQ: continuous skyline query; GP-CSQ: CSQ with geometric pruning; GP2-CSQ: CSQ using geometric pruning from Lemmas 3, 4, 5. (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

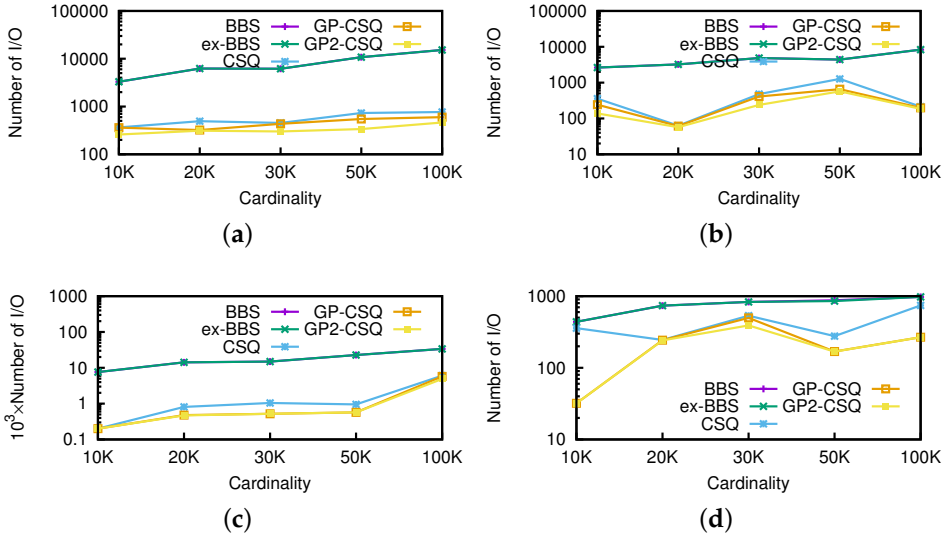

Figure 14. Effect of cardinality (I/O). (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

Effect of nonspatial dimensionality. We used a real 100 K road network dataset combined with nonspatial dimensionality ranging from two to five to evaluate the effect of nonspatial dimensionality on our methods. Values on these nonspatial dimensionalities varied from 1 to 1000. The set of other parameters are the same as shown in Table 3.

As shown in Figure 15, the pruning strategies can save running time to a certain extent while supporting higher nonspatial dimensionality. In Figure 16, CSQ algorithms have a clear advantage in I/Os since they focus on dynamic attributes while nonspatial attributes are considered only in dominance checking. Note that the efficiency of the geometric pruning strategies were affected slightly because a data point is harder to be dominated in higher nonspatial dimensionality and makes it almost impossible to prune more data points.

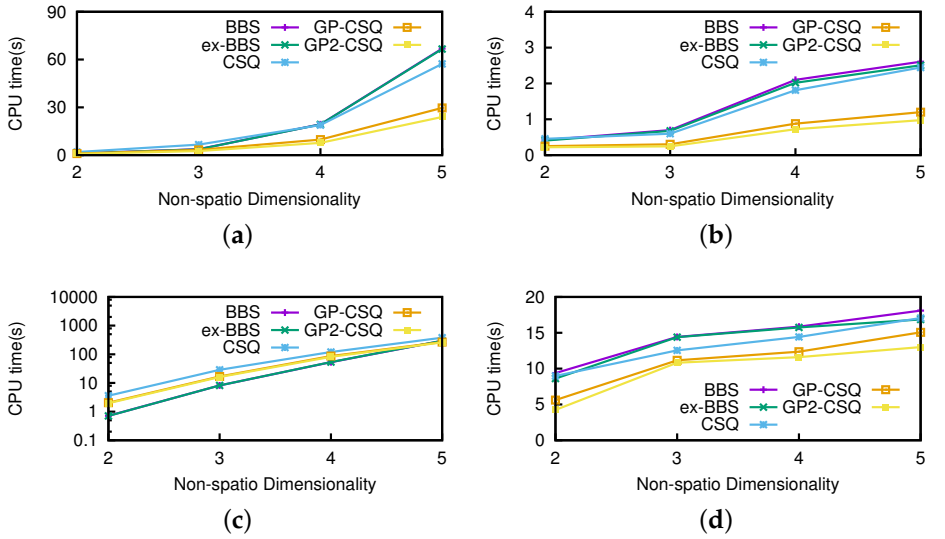

Figure 15. Effect of nonspatial dimensionality (CPU time). (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

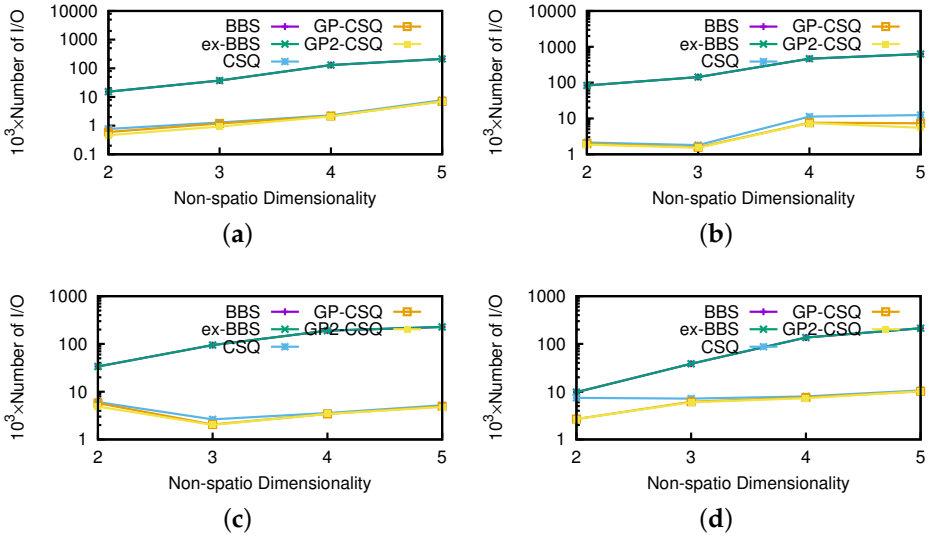

Figure 16. Effect of nonspatial dimensionality (I/O). (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

Effect of Starting Positions. Obviously, the effectiveness of geometric pruning strategies is related to the location of query points. In this section, we performed experiments simulating an object moving from the northwest to the southeast in California to verify the influence of different starting positions. Representatively, the starting positions will be chosen in (42° N,–125° W), (39.5° N,–122.5° W), (37° N,–120° W), (34.5°N,–117.5° W); other query parameters were picked up in the same way as in previous experiments. We mainly explored the effect of position related to the efficiency of the pruning strategies by choosing the above four representative positions for evaluation. Moreover, we intended to obtain the general performance of the geometric pruning strategies.

As shown in Figures 17 and 18, the costs are quite distinct since the efficiency of pruning operations are obviously affected by the positions of the query point. The pruning operation of ex-BBS was almost disabled since there exists a permanent skyline far from the query position. If the query point approaches the center of the spatial area, the pruning operations of ex-BBS are more likely to malfunction. The geometric pruning strategies tend to be available except in the extreme situation in which the query point is starting from the edge of the spatial location. Moreover, the proposed GP-CSQ and GP2-CSQ methods can filter out most of the unqualified data points in some specific cases in which the region to be pruned is extensive (e.g., in the position (34.5°N, –117.5°W), saving about 60%–80% of CPU time and I/Os). Note that in Figure 17d, the CPU cost of the CSQ algorithm is much higher since there are a mount of events needing to be computed and GP-CSQ and GP2-CSQ are approaching that of BBS due to the optimization of the geometric pruning strategies.

Effect of Moving Speed. In this section, we run the experiments where the speed of the query point varies from (1,−1) to (8,−8). We still use the real-world dataset of 100 K combined with two correlated, independent, anti-correlated, and zipf-distributed nonspatial attributes.

In Figures 19 and 20, it is obvious that the cost of CSQ increases with the query speed because the distance of data points intersects more frequently, which means larger numbers of events incur and need to be disposed of, thus consuming more I/Os and CPU time. Optimized by the proposed geometric pruning strategies, the I/O cost of GP-CSQ and GP2-CSQ is not too sensitive. However, the CPU time for creating and handling events is still too high when the query point moves very fast. We can see that CSQ, GP-CSQ, and GP2-CSQ algorithms are more suitable in the context of lower moving speeds. If the query point moves at a high speed, we prefer to compute skyline from scratch; i.e., call BBS algorithm for each time step.

Effect of Error Bound. In this section, we change the setting of thresholds from 0.2 to 0.65 running on the California road network dataset. A higher threshold means that more data points can get in the skyline earlier or leave it later. In Figures 21 and 22, as the thresholds get wider, the costs of CSQ increases slightly since more points remain in the skyline and create more events. Generally, the change of threshold would not cause a prominent effect on the costs of the GP-CSQ and GP2-CSQ methods. In particular, in Figure 22c, the second-time pruning operations using Lemma 5 were evidently weakened by wider threshold, since the available region of data points to be pruned became too small and the distribution made it fail to filter out unqualified data points. The CPU cost was lower when the threshold was equal to 0.5 because the process of computing the probability of dominance checking is simplified to speed up the processing.

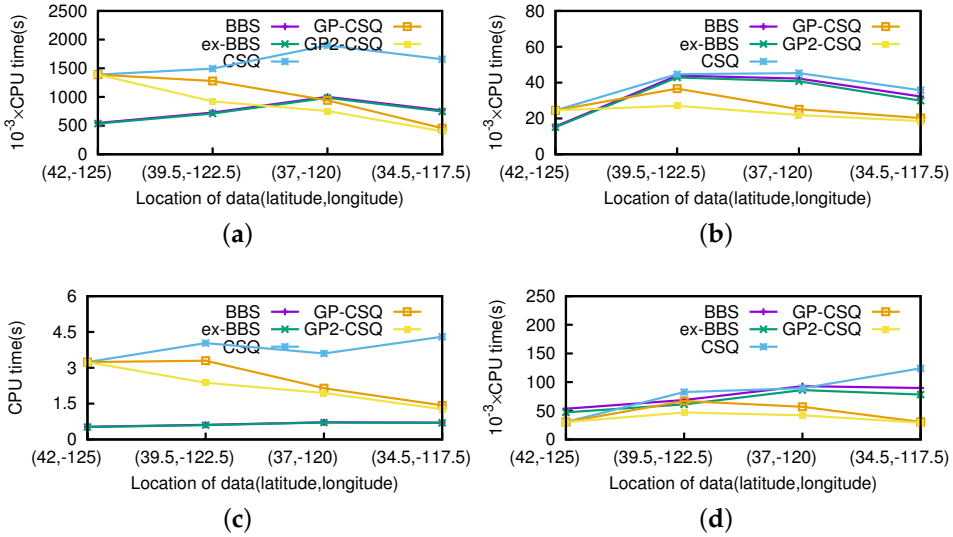

Figure 17. Effect of starting position (CPU time). (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

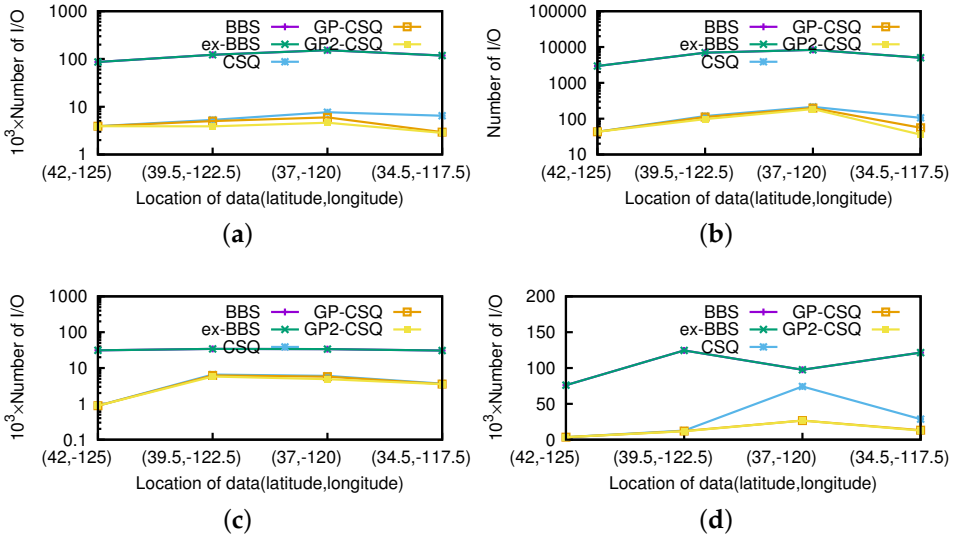

Figure 18. Effect of starting position (I/O). (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

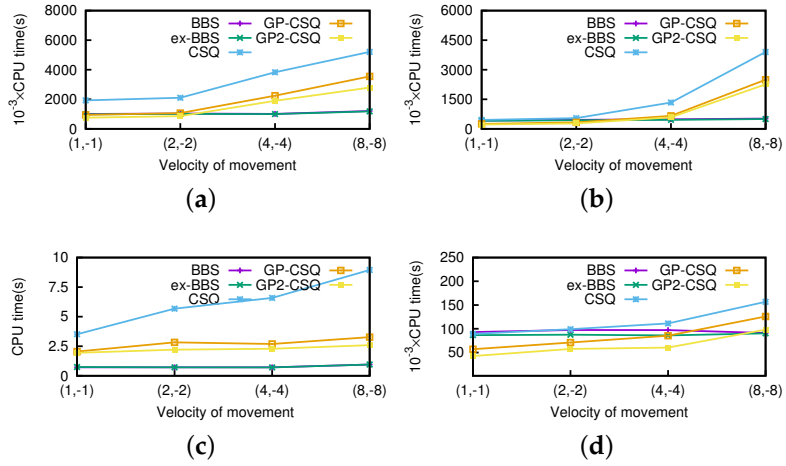

Figure 19. Effect of moving speed (CPU time). (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

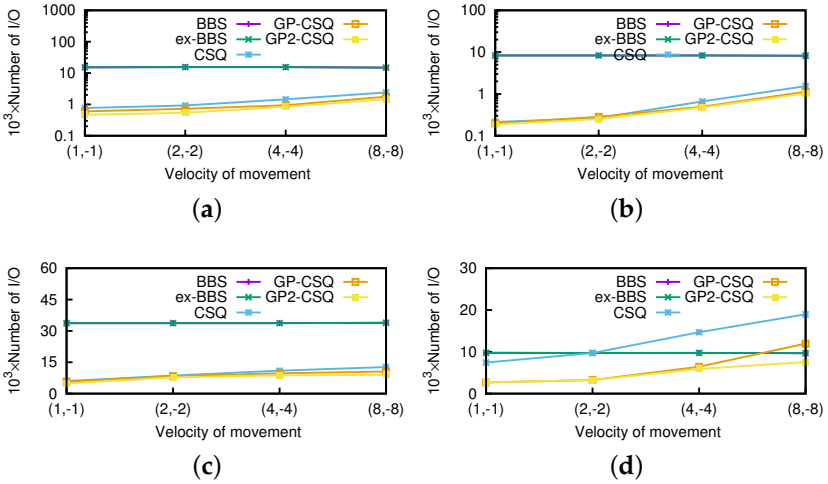

Figure 20. Effect of moving speed (I/O). (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

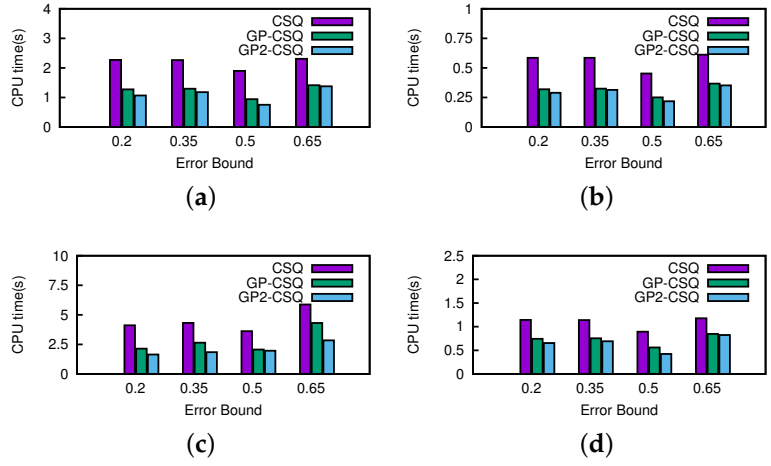

Figure 21. Effect of error bound (CPU time). (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

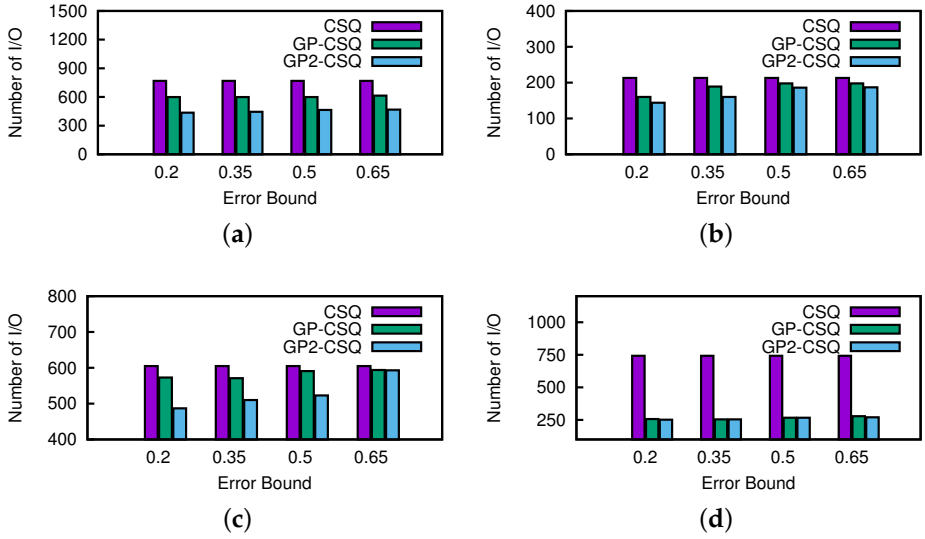

Figure 22. Effect of error bound (I/O). (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

Geometric Pruning Framework Efficiency Evaluation. In this section, we compare the effect of the geometric pruning framework under different kinds of motion patterns (e.g., Enclosed, with Bounds, and without Bounds) to explore whether it is efficient and versatile enough. For simplicity, we take *ellipse*, *sinusoid*, and *parabola* as representative of the three motion patterns mentioned above.

Figure 23 shows that the geometric pruning framework works efficiently as the cardinality of each dataset increases. In particular, for the *ellipse*, the pruning effect is very strong since we can eliminate most of the data points by invoking the geometric pruning algorithm several times for an enclosed motion pattern. More specifically, for the rest motion patterns, about 60% of the initial candidates were pruned out, which can still dramatically reduce the query execution time.

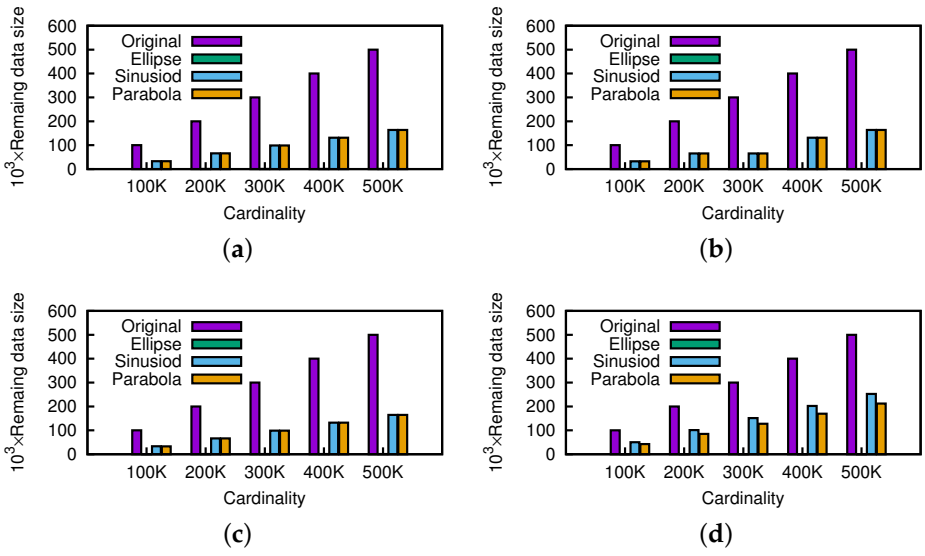

Figure 23. Effect of cardinality. (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

Figure 24 shows the influence of nonspatial dimensionality. The proposed pruning strategy is slightly affected by higher nonspatial dimensionality, since every data point will not be easily dominated in higher dimensionality, but the result indicates that it is still worth executing the pruning

operations. Note that the remaining data sizes of the anti-correlated datasets are large due to their greater skyline results.

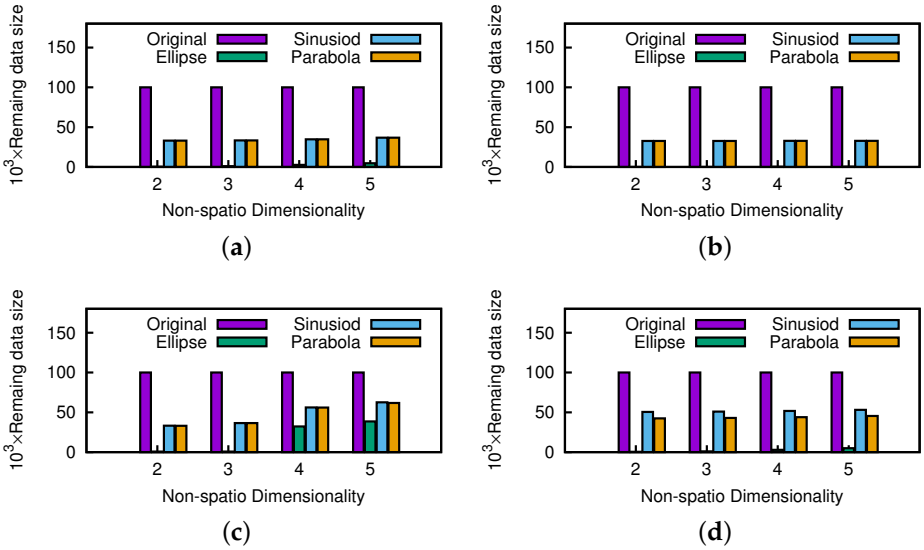

Figure 24. Effect of nonspatial dimensionality. (**a**) Independent; (**b**) Correlated; (**c**) Anti-correlated; (**d**) Zipf.

8. Conclusions

In this paper, we address continuous skyline queries on moving query points under the incremental motion model. Geometric properties are fully exploited to prune the data points which will not belong to the final skyline results, thus improving the efficiency of skyline query processing. Further, event-based mechanisms and a grid file index-based pruning policy are proposed to maintain continuous skyline results instead of computing skyline results from scratch. Two efficient algorithms (GP-CSQ and GP2-CSQ) are proposed based on geometric properties, and our extensive experiments have shown that the two geometric property-based algorithms are more effective and efficient than existing methods.

There are many promising future directions. Firstly, suitable motion patterns for specific applications can be found, and we can study how to alter the pruning strategies based on corresponding geometric properties to adapt the new motion pattern under our framework. Secondly, since we assume only query points and point attributes are dynamic, if the databases change (i.e., the data points are varying—insert, update, or delete), we can study how to develop efficient algorithms to answer user continuous skyline queries. Thirdly, future work can be devoted to investigating the possibility of using the proposed geometric pruning strategies to support other variants of skyline queries, such as reserve skyline queries [24], skyline cubes [29], spatial skyline queries [16], probabilistic skylines [27], and so on. While results obtained from comprehensive experiments indicated the superiority of our approaches devised based on the geometric pruning strategies over existing works, we believe that these explored geometric features and the proposed framework are useful for other skyline query variants not examined in this paper. Another interesting problem is to extend the geometric pruning strategies to other real applications (e.g., recommendation systems [30], mobile sensor networks [31,32], and surveillance systems [33]) such that whenever a query point is located, after a snapshot query is performed, by using the proposed geometric pruning strategies or other extentions, we can get the small part of candidates which may impact the query result in the near future without verifying all data points in the system so that the query results can be maintained efficiently according to the small-scale candidate datasets.

Acknowledgments: This work is partially supported by the National Basic Research 973 Program of China under Grant No. 2014CB744900, the Natural Science Foundation of Jiangsu Province of China under grant No. BK20140826, the Fundamental Research Funds for the Central Universities under grant No. NS2015095, Funding of Graduate Innovation Center in NUAA under grant No. KFJJ20151606. The authors also would like to thank the anonymous reviewers for their helpful suggestions.

Author Contributions: Jiping Zheng, Jialiang Chen and Haixiang Wang conceived and designed the experiments; Jialiang Chen and Haixiang Wang performed the experiments; Jiping Zheng analyzed the data; Jiping Zheng and Jialiang Chen wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

BNL	Block nested loop
SFC	Sort filter skyline
NN	Nearest neighbor
MP	Motion processor
IM	Incremental motion
PWS	Possible world aemantics
LBS	Location-based service
KDS	Kinetic data structure
BBS	Branch-and-bound skyline
CSQ	Continuous skyline query
GP	Geometric pruning

References

1. Börzsöny, S.; Kossmann, D.; Stocker, K. The Skyline Operator. Available online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.2504 (accessed on 21 March 2017)
2. Chomicki, J.; Ciaccia, P.; Meneghetti, N. Skyline Queries, Front and Back. *SIGMOD Rec.* **2013**, *42*, 6–18.
3. Papadias, D.; Tao, Y.; Fu, G.; Seeger, B. An Optimal and Progressive Algorithm for Skyline Queries. In Proceedings of the SIGMOD 2003, San Diego, CA, USA, 9–12 June 2003,.
4. Papadias, D.; Tao, Y.; Fu, G. Progressive skyline computation in database systems. *TODS* **2005**, *30*, 41–82.
5. Huang, Z.; Lu, H.; Ooi, B.C.; Tung, A.K.H. Continuous Skyline Queries for Moving Objects. *TKDE* **2006**, *18*, 1645–1658.
6. Lin, X.; Xu, J.; Hu, H. Range-Based Skyline Queries in Mobile Environments. *ICDE* **2013**, *25*, 835–849.
7. Guo, X.; Zheng, B.; Ishikawa, Y.; Gao, Y. Direction-based Surrounder Queries for Mobile Recommendations. *VLDB J.* **2011**, *20*, 743–766.
8. Qiao, Z.; Gu, J.; Lin, X.; Chen, J. Privacy-Preserving Skyline Queries in LBS. In Proceedings of the 2010 International Conference on Machine Vision and Human-machine Interface, Kaifeng, China, 24–25 April 2010.
9. Basch, J.; Guibas, L.J.; Hershberger, J. Data Structures for Mobile Data. In Proceedings of the SODA '97, New Orleans, LA, USA, 5–7 January 1997.
10. Basch, J.; Guibas, L.J.; Hershberger, J. Data Structures for Mobile Data. *J. Algorithms* **1999**, *31*, 1 – 28.
11. Tan, K.L.; Eng, P.K.; Beng, C.O. Efficient Progressive Skyline Computation. Available online: http://www.vldb.org/conf/2001/P301.pdf (accessed on 21 March 2017).
12. Chomicki, J.; Godfrey, P.; Gryz, J. Skyline with Presorting. Available online: http://ieeexplore.ieee.org/document/1260846/ (accessed on 21 March 2017).
13. Godfrey, P.; Shipley, R.; Gryz, J. Maximal Vector Computation in Large Data Sets. In Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, 30 August–2 September 2005.
14. Kossmann, D.; Ramsak, F.; Rost, S. Shooting Stars in the Sky: An online algorithm for skyline queries. In Proceedings of the VLDB '02 , Hong Kong, China, 20–23 August 2002.
15. Hose, K.; Vlachou, A. A Survey of Skyline Processing in Highly Distributed Environments. *VLDB J.* **2012**, *21*, 359–384.

16. Sharifzadeh, M.; Shahabi, C. The Spatial Skyline Queries. In Proceedings of the VLDB '06, Seoul, Korea, 12–15 September 2006.
17. Deng, K.; Zhou, X.; Shen, H.T. Multi-source Skyline Query Processing in Road Networks. In Proceedings of the ICDE '07, Istanbul, Turkey, 15–20 April 2007.
18. Mount, D.M.; Netanyahu, N.S.; Piatko, C.D.; Silverman, R.; Wu, A.Y. A Computational Framework for Incremental Motion. In Proceedings of the SCG '04, Brooklyn, New York, USA, 8–11 June 2004.
19. Cho, M.; Mount, D.; Park, E. Maintaining Nets and Net Trees under Incremental Motion. In *Algorithms and Computation*; Springer: Berlin, Germany, 2009; pp. 1134–1143.
20. Lee, M.W.; Hwang, S.W. Continuous Skylining on Volatile Moving Data. In Proceedings of the ICDE '09, Shanghai, China, 29 March–2 April 2009.
21. Hsueh, Y.; Hascoet, T. Caching Support for Skyline Query Processing with Partially Ordered Domains. *TKDE* **2014**, *26*, 2649–2661.
22. Cheema, M.A.; Lin, X.; Zhang, W.; Zhang, Y. A Safe Zone Based Approach for Monitoring Moving Skyline Queries. In Proceedings of the EDBT 2013, Brussels, Belgium, 23–27 March 2013.
23. Vu, K.; Zheng, R. Efficient Algorithms for Spatial Skyline Query with Uncertainty. In Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Orlando, FL, USA, 5–8 November 2013.
24. Dellis, E.; Seeger, B. Efficient Computation of Reverse Skyline Queries. In Proceedings of the 33rd International Conference on Very Large Data Bases, Vienna, Austria, 23–28 September 2007.
25. Sacharidis, D.; Bouros, P.; Sellis, T. Caching Dynamic Skyline Queries. In Proceedings of the 20th International Conference on Scientific and Statistical Database Management, Hong Kong, China, 9–11 July 2008.
26. Mortensen, M.L.; Chester, S.; Assent, I.; Magnani, M. Efficient caching for constrained skyline queries. In Proceedings of the EDBT '15, Brussels, Belgium, 23–27 March 2015.
27. Pei, J.; Jiang, B.; Lin, X.; Yuan, Y. Probabilistic Skylines on Uncertain Data. In Proceedings of the 33rd International Conference on Very Large Data Bases, Vienna, Austria, 23–28 September 2007.
28. Feifei, L. Spatial Datasets. Available online: https://www.cs.utah.edu/~lifeifei/SpatialDataset.html (accessed on 21 March 2017).
29. Yuan, Y.; Lin, X.; Liu, Q.; Wang, W.; Yu, J.X.; Zhang, Q. Efficient Computation of the Skyline Cube. In Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, 30 August–2 September 2005.
30. Pazzani, M.J.; Billsus, D. Content-Based Recommendation Systems. In *The Adaptive Web: Methods and Strategies of Web Personalization*; Brusilovsky, P.; Kobsa, A.; Nejdl, W., Eds.; Springer: Berlin, Germany, 2007; pp. 325–341.
31. Ciuonzo, D.; Buonanno, A.; D'Urso, M. Distributed classification of multiple moving targets with binary wireless sensor networks. In Proceedings of the FUSION '11, Chicago, IL, USA, 5–8 July 2011.
32. Buonanno, A.; D'Urso, M.; Prisco, G. Mobile sensor networks based on autonomous platforms for homeland security. In Proceedings of the Advances in Radar and Remote Sensing (TyWRRS), Naples, Italy, 12–14 September 2012.
33. Tsiligkaridis, T.; Sadler, B.M.; Hero, A.O. On decentralized estimation with active queries. *IEEE Trans. Signal Process.* **2015**, *63*, 2610–2622.