The Spatial Skyline Queries*

Mehdi Sharifzadeh
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
sharifza@usc.edu

Cyrus Shahabi
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
shahabi@usc.edu

ABSTRACT

In this paper, for the first time, we introduce the concept of Spatial Skyline Queries (SSQ). Given a set of data points P and a set of query points Q, each data point has a number of *derived spatial* attributes each of which is the point's distance to a query point. An SSQ retrieves those points of P which are not dominated by any other point in P considering their derived spatial attributes. The main difference with the regular skyline query is that this *spatial domination* depends on the location of the query points Q. SSQ has application in several domains such as emergency response and online maps. The main intuition and novelty behind our approaches is that we exploit the geometric properties of the SSQ problem space to avoid the exhaustive examination of all the point pairs in P and Q. Consequently, we reduce the complexity of SSQ search from $O(|P|^2|Q|)$ to $O(|S|^2|C| + \sqrt{|P|})$, where $|S|$ and $|C|$ are the solution size and the number of vertices of the convex hull of Q, respectively.

We propose two algorithms, B^2S^2 and VS^2, for static query points and one algorithm, VCS^2, for streaming Q whose points change location over time (e.g., are mobile). VCS^2 exploits the pattern of change in Q to avoid unnecessary re-computation of the skyline and hence efficiently perform updates. Our extensive experiments using real-world datasets verify that both R-tree-based B^2S^2 and Voronoi-based VS^2 outperform the best competitor approach in terms of processing time by a wide margin (4-6 times better in most cases).

1. INTRODUCTION

Assume that the members of a multidisciplinary task force team located at different (fixed) offices want to put together a list of restaurants for their weekly lunch meetings. These meeting locations must be *interesting* in terms of traveling distances for all the team members; for each restaurant r in the list, no other restaurant is closer to *all* members than r. That is, there exists no *better* restaurant of choice than an interesting restaurant in terms of all of their comparable *attributes* (here, distances to team members). Generating this list becomes even more challenging when the team members are mobile and change location over time.

Suppose the information about all restaurants are stored in a database as objects with static attributes (e.g., rating of the restaurant). The database literature refers to the interesting objects with respect to their *static* attributes as *skyline* objects of the database; those that are not *dominated* by any other object. These static skyline objects depend only on the database itself (and the definition of the *interesting* object). Even though the interesting restaurants to our team members can also be considered as the skyline of the database of restaurants, there are two major distinctions: 1) The restaurants' distance attributes based on which the domination is defined are dynamically calculated based on the user's query (i.e., location of the team members). Consequently, the result depends on both data and the given query. 2) As these attributes are spatial, there is a unique corresponding geometric interpretation of spatial skyline in the space of the objects.

The query retrieving the set of interesting restaurants in our motivating example belongs to a broader novel type of spatial queries. In this paper, for the first time, we introduce the concept of these *Spatial Skyline Queries* (SSQ). Given a set of data points P and a set of query points Q in a d-dimensional space, an SSQ retrieves those points of P which are not dominated by any other point in P considering a set of spatial derived attributes. For each data point, these attributes are its distances to query points in Q. An interesting variation, which is also studied in this paper, is where the domination is determined with respect to both spatial and non-spatial attributes of P. Besides online map services and group navigation/planning, SSQ is critical for many applications. In the domain of *trip planning*, the spatial skyline of hotels with respect to the fixed locations of conference venue, beaches and museums includes all the interesting hotels for lodging during a business/pleasure trip. No other hotel is closer than these hotels to *all* three must-see locations. In *crisis management* domain, the residential buildings that must be evacuated first in the event of several explosions/fires are those which are in the spatial

*This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0238560 (PECASE), IIS-0324955 (ITR), and unrestricted cash gifts from Google and Microsoft. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VLDB '06, September 12-15, 2006, Seoul, Korea.

skyline with respect to the fire locations. The reason is that these places are either potentially trapped in the convex hull of fires or located at the edges of the expanding fire. In *defense and intelligence* applications, consider the locations of soldiers penetrating into enemy's camps as query locations and the enemy's guard stations as data points. The stations in the spatial skyline are those from which an attack might be initiated against the platoon of soldiers.

Since the introduction of the skyline operator by Börzsönyi et al. [1], several efficient algorithms have been proposed for the general skyline query. These algorithms utilize techniques such as divide-and-conquer [1], nearest neighbor search [6], sorting [2], and index structures [1, 10, 8] to answer the general skyline queries. Several studies have also focused on the skyline query processing in a variety of problem settings such as data streams [7] and data residing on mobile devices [5]. However, to the best of our knowledge, no study has addressed the *spatial skyline queries*. The most relevant work is the BBS algorithm proposed by Papadias et al. [8], which can be utilized to address SSQ by considering it as a special case of *dynamic* skyline query. However, since BBS is addressing a more general problem, it overlooks the geometric properties of SSQ and hence its performance is not optimal in spatial domain. We compare our techniques with BBS in Section 7.

Optimal processing of SSQs is more challenging than the general skyline queries as each dominance check here requires the computation of the dynamic distance attributes. Notice that the algorithms for Group or Aggregate Nearest Neighbor queries [9] are related but not applicable to SSQ as they only find the optimal (best) object based on a fixed preference function. Hence, they require no intermediate dominance checks that are vital in processing SSQs.

In this paper, we approach the problem of processing SSQs as a query in spatial domain from a geometric perspective. First, we analytically exploit the geometric properties of the problem and solution spaces. We theoretically prove that unlike general skyline there are data points in P that are definitely in the spatial skyline of P with respect to Q independent of the rest of P. Likewise, there are query points that have no impact on the inclusion/exclusion of any data point in/from the skyline. Utilizing this theoretical foundation, we propose two algorithms, B^2S^2 and VS^2, for static query points and one algorithm, VCS^2, for streaming Q whose points change location over time (e.g., are mobile).

The R-tree-based B^2S^2 is a customization of BBS [8] for SSQ that benefits from our theoretical foundation by exploiting the geometric properties of the problem space. B^2S^2 is more efficient than BBS as it not only avoids expensive dominance checks for definite skyline points but also prunes unnecessary query points to reduce the cost of each examination. VS^2, however, employs the Voronoi diagram and Delaunay graph of the data points as a roadmap to find the first skyline point whose local neighborhood contains all other points of the skyline. The algorithm traverses the nodes of the graph (i.e., data points of P) in the order specified by a monotone function over their distances to query points. Similar to B^2S^2, VS^2 also efficiently exploits the geometric properties of the problem space while traversing the Delaunay graph. VS^2 reduces the complexity of the naive SSQ search from $O(|P|^2|Q|)$ to $O(|S|^2|C|+\sqrt{|P|})$, where $|S|$ and $|C|$ are the solution size and the number of vertices of the convex hull of Q, respectively. The $\sqrt{|P|}$ factor can be reduced further to $O(\log|P|)$ if an index structure is used.

We comprehensively study the scenario where the query points are the locations of *multiple moving* objects with *unpredefined trajectories*. This occurs in our motivating applications when the team members are mobile agents or the soldiers are moving. Here, we frequently receive the latest query locations as spatial data streams. For each pattern of movement, we extract the locus of all points whose dominance change and hence trigger an update to the old spatial skyline. Consequently, to address continuous SSQ we propose VCS^2 that exploits the pattern of change in Q to avoid unnecessary re-computation of the skyline and hence efficiently perform updates.

Furthermore, we study a more general case of SSQ where one intends to find the skyline with respect to both static *non-spatial* attributes of the data points and their distances to query points. For instance, the best restaurant in LA might be dominated in terms of distance to our team members but it is still in the skyline because of its rating. We show that B^2S^2, VS^2, and VCS^2 all can support this variation of SSQ as well.

Finally, through extensive experiments with both real-world and synthetic datasets, we show that B^2S^2, VS^2, and VCS^2 can efficiently answer an SSQ query. Both R-tree-based B^2S^2 and Voronoi-based VS^2 outperform the best competitor approach BBS in terms of processing time by a wide margin (up to 6 times better). Our experimental results with synthetically moving objects verify that on average for 3-10 query points only less than 25% of movements require the entire skyline to be recomputed. For the other 75% of movements, VCS^2 outperforms VS^2 by a factor of 3 which makes it the superior algorithm for continuous SSQ.

The remainder of this paper is organized as follows. We first formally define the problem of SSQs and the terms we use throughout the paper in Section 2. In Section 3, we establish the theoretical foundation of our proposed algorithms. In Section 4, we discuss our alternative solutions for SSQ. We address continuous SSQ for moving objects and incorporating non-spatial attributes in SSQ in Sections 5 and 6, respectively. The performance of our proposed algorithms is evaluated in Section 7. The related work to SSQ and similar nearest neighbor queries are presented in Section 8. Finally, we conclude the paper and discuss our future work in Section 9.

2. FORMAL PROBLEM DEFINITION

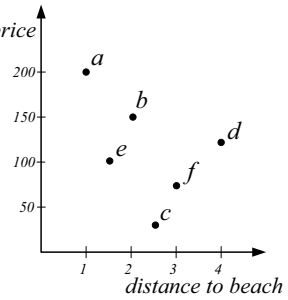
Assume that we have a database of N objects. Each database object p with d real-valued attributes can be conceptualized as a d-dimensional point $(p_1, \ldots, p_d) \in \mathbb{R}^d$ where p_i is the i-th attribute of p. We use P to refer to the set of all these points. Figure 1a illustrates a database of six objects $P = \{a, b, c, d, e, f\}$ each representing the description of a hotel with two attributes: *distance to beach* and *price*. Figure 1b shows the corresponding points in the 2-dimensional space where x and y axes correspond to the range of attributes *distance* and *price*, respectively. Throughout the paper, we use *point* and object interchangeably to refer to each database object. In the following sections, we first define the general skyline query using the above database conceptualization. Then, we introduce our spatial skyline query based on the definition of the general skyline query.

2.1 General Skyline Query

Given the two points $p=(p_1, \ldots, p_d)$ and $p'=(p'_1, \ldots, p'_d)$

object	distance (mile)	price ($)
a	0.5	200
b	2	150
c	2.5	25
d	4	125
e	1.5	100
f	3	75

(a)



(b)

Figure 1: A 2-dimensional database of six objects

in $\mathbb{R}^d$, p *dominates* p' iff we have $p_i \leq p'_i$ for $1 \leq i \leq d$ and $p_j < p'_j$ for some $1 \leq j \leq d$. To illustrate, in Figure 1b the point f=(3, 75) dominates the point d=(4, 125). Now, given a set of points P, the *skyline* of P is the set of those points of P which are not dominated by any other point in P. The skyline of the points shown in Figure 1b is the set $S = \{a, c, e\}$. The *Skyline Query* is to find the skyline set of the given database P considering attributes of the objects in P as dimensions of the space. Notice that every point of the skyline does not need to dominate a point of P. For instance in Figure 1b, while the points c and e each dominate two other points, the point a dominates no point.

2.2 Spatial Skyline Query

Let the set P contain points in the d-dimensional space $\mathbb{R}^d$, and $D(.,.)$ be a distance metric defined in $\mathbb{R}^d$ where $D(.,.)$ obeys the triangle inequality. Given a set of d-dimensional query points Q=$\{q_1, \ldots, q_n\}$ and the two points p and p' in $\mathbb{R}^d$, p *spatially* dominates p' with respect to Q iff we have $D(p, q_i) \leq D(p', q_i)$ for all $q_i \in Q$ and $D(p, q_j) < D(p', q_j)$ for some $q_j \in Q$. To be specific, p spatially dominates p' iff every q_i is closer to p than to p' or at the same distance from p and p'. Figure 2 shows a set of nine 2-d points and two query points q_1 and q_2. With Euclidean distance metric, the point p spatially dominates the point p' as both q_1 and q_2 are closer to p than to p'. Note that if we draw the perpendicular bisector line of the line segment pp', q_1, q_2, and p will be located on the same side of the bisector line (where p' is not; see Figure 2). We use this geometric interpretation of the spatial dominance relation between two points in Section 3.2 to justify the foundation of our proposed algorithms.

For each point p, consider circles $C(q_i, p)$ centered at the query point q_i with radius $D(q_i, p)$. Obviously, q_i is closer to any point inside $C(q_i, p)$ than to p. Therefore, by the above definition any point such as p'' which is inside the intersection of all $C(q_i, p)$ for all $q_i \in Q$ spatially dominates p. We call this intersection area which potentially includes all the points which spatially dominate p as the *dominator region* of p. Similarly, the locus of all points such as p' which are spatially dominated by p is the intersection of outside of all circles $C(q_i, p)$ (the grey region in Figure 2)[1]. For a point p, we refer to this region as the *dominance region* of p.

Given the two sets P of data points and Q of query points, the *spatial skyline* of P with respect to Q is the set of those points in P which are not spatially dominated by any other point of P; the points which are not inside the dominance region of any other point. The point $p \in P$ is in the spatial skyline of P with respect to Q iff for any point $p' \in P$ there is a query point $q_i \in Q$ for which we have $D(p, q_i) \leq D(p', q_i)$. That is, p is in the spatial skyline iff we have:

$$\forall p' \in P, p' \neq p, \ \exists q_i \in Q \ s.t. \ D(p, q_i) \leq D(p', q_i) \qquad (1)$$

We use *spatial skyline point* and *skyline point* interchangeably to denote any point in the spatial skyline. Considering the above definitions, the *Spatial Skyline Query (SSQ)* is to find the spatial skyline points of the given set P with respect to the query set Q.

The naive brute-force search algorithm for finding the spatial skyline of P given a query set Q requires to examine all points in P against each other. For each point p, $|Q|$ distances $D(p, q_i)$ are computed and compared against the corresponding distances of other points. If no point spatially dominates p, then p is added to the solution. The time complexity of this naive algorithm is $O(|P|^2|Q|)$ as it exhaustively examines all data points against each other.

However, an optimal SSQ algorithm must examine each point p against only those points which are inside the dominator region of p. In Section 3, we establish the theoretical foundation of our efficient SSQ algorithms which allows us to avoid a significant number of distance computations of the naive approach.

[1]Assume that the large rectangle shows the universe space of the data points.

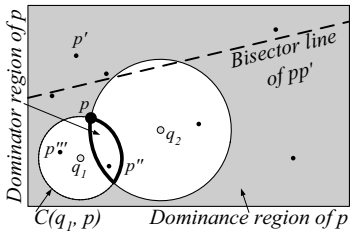


Figure 2: The spatial skyline of a set of nine points

Symbol	Meaning
P	set of database points
N	$\|P\|$, cardinality of P
$D(.,.)$	distance function in P
Q	set of query points $\{q_1, \ldots, q_n\}$
n	$\|Q\|$, cardinality of Q
$C(q_i, p)$	circle centered at q_i with radius $D(q_i, p)$
$VC(p)$	Voronoi cell of p
$CH(Q)$	convex hull of Q
$CH_v(Q)$	set of vertices of $CH(Q)$
$S(Q)$	spatial skyline points of P with respect to Q

Table 1: Summary of notations

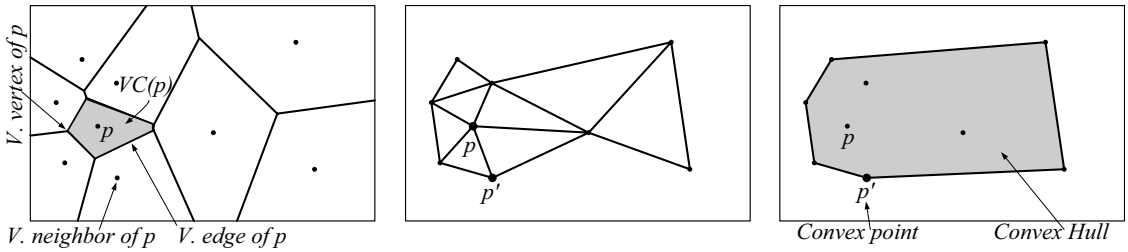
3. FOUNDATION

We first identify the geometric structure of the solution space corresponding to the SSQ problem. In particular, we intend to understand how the spatial skyline points of a given query are related to the data and query points. These relationships constitute the foundation of our proposed solutions to SSQ. To start, we first describe three geometric structures utilized by our solutions, namely *Voronoi diagram*, *Delaunay graph*, and *convex hull*. Notice that while throughout the paper we use Euclidean distance function in 2-*d* space, our results hold in general for any dimension *d*.

3.1 Preliminaries

3.1.1 Voronoi Diagram and Delaunay Graph

The Voronoi diagram of a given set P of points in $\mathbb{R}^d$ partitions the space into regions each including all points with a common closest point in P according to a distance



(a) (b) (c)

Figure 3: a) Voronoi diagram, b) Delaunay graph, and c) Convex hull

metric $D(.,.)$ [3].The region corresponding to the point $p \in P$ contains all the points $x \in \mathbb{R}^d$ for which we have

$$\forall p' \in P, p' \neq p,\ D(x,p) \leq D(x,p') \quad (2)$$

The equality holds for the points on the borders of p's and p''s regions. Figure 3a shows the Voronoi diagram of nine points in $\mathbb{R}^2$ where the distance metric is Euclidean. We refer to the region $VC(p)$ containing the point p as its Voronoi *cell*. For Euclidean distance in $\mathbb{R}^2$, $VC(p)$ is a convex polygon. Each edge of this polygon is a segment of the perpendicular bisector line of the line segment connecting p to another point of the set P. We call each of these edges a *Voronoi edge* and each of its end-points (vertices of the polygon) a *Voronoi vertex* of the point p. For each Voronoi edge of the point p, we refer to the corresponding point in the set P as a *Voronoi neighbor* of p.

Now consider an undirected graph $G(V,E)$ with the set of vertices $V = P$. For each two points p and p' in V, there is an edge connecting p and p' in G iff p' is a Voronoi neighbor of p in the Voronoi diagram of P. The graph G is called the Delaunay Graph of points in P. Figure 3b illustrates the Delaunay graph corresponding to points of Figure 3a. The Delaunay graph of any set of points is a connected planar graph. In Section 4.2, we traverse the Delaunay graph of the database points to find the set of skyline points.

3.1.2 Convex Hull

The convex hull of points in $P \subset \mathbb{R}^d$, is the unique smallest convex polytope (polygon when $d = 2$) which contains all the points in P [3]. Figure 3c shows the convex hull of the points of Figure 3a as a hexagon. The set of vertices of convex hull of points in P is a subset of P. We use *convex point* to denote any of these vertices and *non-convex* point to refer to all other points of P. In Figure 3c, p' and p are convex and non-convex points, respectively. We also use $CH(P)$ and $CH_v(P)$ to refer to the convex hull of P and the set of its vertices, respectively. It is clear that the shape of the convex hull of a set P only depends on the convex points in P. Consequently, the location of any non-convex point $p \in P$ does not affect the shape of $CH(P)$.

3.2 Theories

We assume that the set of data points P and query points Q are given. In this section, we exploit the geometric properties of spatial skyline points of P with respect to Q. To reduce our search space in finding spatial skyline points, we prove one lemma (1) and two theorems (1 & 3) that would help us to immediately identify definite skyline points and one theorem (2) to eliminate some of the query points not contributing to the search. The first property holds for both general and spatial skylines.

LEMMA 1. *For each $q_i \in Q$, the closest point to q_i in P is a skyline point.*

PROOF. If p is the closest point to q_i in P, we have $D(p,q_i) < D(p',q_i)$ for all $p' \in P$ ($p' \neq p$). By definition, no point in P spatially dominates p. Hence, p is a skyline point. □

A restated form of Lemma 1 is valid for the general skyline where all the points with the minimum value in one dimension are in the skyline. Lemma 1 shows that the inclusion of some data points in the spatial skyline of P does not depend on the location of any other point of P. For example in Figure 2, the point p''' is a skyline point as it is the closest point to the query point q_1. It is a skyline point only because of q_1's location regardless of where the other data points of P are located. In Section 4.2, we utilize Lemma 1 to start our search for skyline points from a definite skyline point such as p'''. The following theorem also shows that specific points in P are skyline points independent of the locations of other points of P.

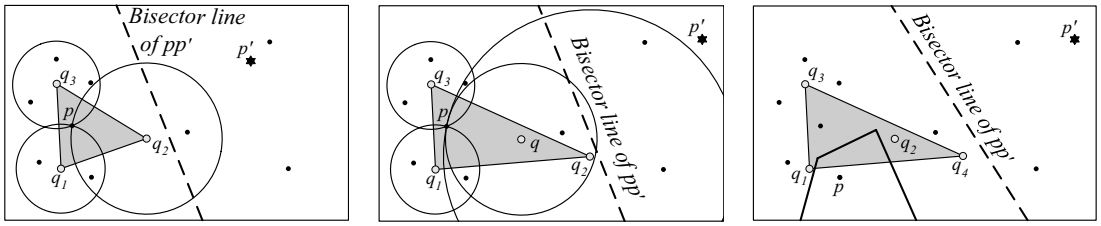
THEOREM 1. *Any point $p \in P$ which is inside the convex hull of Q is a skyline point.*

PROOF. The proof is by contradiction. Assume that p which is inside the convex hull $CH(Q)$ is not a skyline point (see Figure 4a). Then, there is a point $p' \in P$ which spatially dominates p. Therefore, if we draw the perpendicular bisector line of the line segment pp', all the query points q_i will be located on the same side of the line where p' is located. Hence, $CH(Q)$ is also on the same side as p'. That is, the perpendicular bisector line of pp' separates $CH(Q)$ from p. This contradicts our assumption that p is inside $CH(Q)$ and proves that p is a skyline point. □

Theorem 1 enables our SSQ algorithms to efficiently retrieve a large subset of skyline points only by examining them against the query points. This theorem is the basis of our proposed SSQ algorithms in Section 4. The next theorem improves the application of Theorem 1 by proving that even some of the query points have no impact on the final spatial skyline. Hence, our SSQ algorithms can easily ignore them. We first prove the following lemma:

LEMMA 2. *Given two query sets $Q' \subset Q$, if a point $p \in P$ is a skyline point with respect to Q', then p is also a skyline point with respect to Q.*

PROOF. As p is a skyline point with respect to Q', for any point $p' \in P$ there is a query point $q_i \in Q'$ for which we have $D(p,q_i) \leq D(p',q_i)$ (see Equation 1). As Q' is a subset of Q, we have $q_i \in Q$. Therefore, according to Equation 1 the point p is a skyline point with respect to Q. □



(a) (b) (c)

Figure 4: a) Theorem 1, b) Theorem 2, and c) Theorem 3

Lemma 2 holds for both general and spatial skylines. In the general case, if a point p is in the skyline considering only a subset of its coordinates (i.e., attributes), it remains in the final skyline when all coordinates of p are considered.

THEOREM 2. *The set of skyline points of P does not depend on any non-convex query point $q \in Q$.*

PROOF. Assume that the query point q is not a convex point (i.e., $q \notin CH_v(Q)$) (see Figure 4b). Consider the set $Q' = Q - \{q\}$. We prove that the spatial skylines of Q and Q' are equal (i.e., $S(Q') = S(Q)$). First, assume that p is a skyline point with respect to Q' (i.e., $p \in S(Q')$). According to Lemma 2, as we have $Q' \subset Q$, p is a skyline point with respect to Q.

Now, we assume that we have $p \in S(Q)$. We show that $p \in S(Q')$. The proof is by contradiction. Assume that p is not in $S(Q')$. Then, there is a point such as $p' \in P$ that spatially dominates p with respect to Q'. By definition, the query points in Q' and the point p are on different sides of the perpendicular bisector line of line segment pp'; Q' is on the same side as p'. Therefore, the convex hull of Q', $CH(Q')$, and p are separated by the bisector line. Moreover, as q is not a convex point of Q, we have $CH(Q) = CH(Q')$. Hence, $CH(Q)$ is also separated from p by the bisector line. That is, p' spatially dominates p with respect to both Q' and Q and so we get $p \notin S(Q)$ which contradicts our assumption. Therefore, we showed that $p \in S(Q')$.

Combining all the above, we proved that $S(Q') = S(Q)$ and hence the inclusion/exclusion of the single non-convex query point q in/from Q does not change the set of skyline points of P. □

To illustrate, consider the point q inside the convex hull of query points shown in Figure 4b. Theorem 2 implies that both dominance and dominator regions of any point p is independent from q. The intuition here is that the circle $C(q,p)$ is completely inside the union of the circles $C(q_i,p)$ for $q_i \in CH_v(Q)$. With the result of Theorem 2, we reduce the time complexity of our SSQ algorithms by disregarding the distance computation operations against the non-convex query points such as q (see Section 4). Finally, the last theorem specifies those skyline points which are identified by examining only the data points in a limited local proximity around them.

THEOREM 3. *Any point $p \in P$ whose Voronoi cell $VC(p)$ intersects with the boundaries of convex hull of Q is a skyline point.*

PROOF. The proof is by contradiction. Assume that the Voronoi cell of p intersects with $CH(Q)$ but p is not a skyline point (i.e., $p \notin S(Q)$) (see Figure 4c). Hence, a point such as $p' \in P$ spatially dominates p with respect to Q. Therefore, the perpendicular bisector line of line segment pp' separates both p' and Q from p. That is, p and the convex hull of Q are on different sides of the bisector line. As $VC(p)$ intersects with $CH(Q)$, the intersection of $VC(p)$ and $CH(Q)$ is also separated from p by the bisector line. It means that the points in this intersection region are closer to p' than to p. That is, there are some points such as x inside $VC(p)$ for which we have $D(x,p') < D(x,p)$. This inequality contradicts the definition of $VC(p)$ given in Equation 2 which states that any point in $VC(p)$ is closer to p than to any other point in the set P. Thus, p is a skyline point with respect to Q. □

Algorithm B²S² (set Q)
01. compute the convex hull $CH(Q)$;
02. set $S(Q) = \{\}$;
03. box $B = MBR(R)$;
04. minheap $H = \{(R, 0)\}$;
05. while H is not empty
06. remove first entry e from H;
07. if e does not intersect with B, discard e;
08. if e is inside $CH(Q)$ or
09. e is not dominated by any point in $S(Q)$
10. if e is a data point p
11. add p to $S(Q)$;
12. $B = B \cap MBR(SR(p,Q))$;
13. else // *e is an intermediate node*
14. for each child node e' of e
15. if e' does not intersect with B, discard e';
16. if e' is inside $CH(Q)$ or
17. e' is not dominated by any point in $S(Q)$
18. add $(e', mindist(e', CH_v(Q)))$ to H;
19. return $S(Q)$;

Figure 5: Pseudo-code of the B²S² algorithm

4. SOLUTIONS

In this section, we propose two algorithms to solve the SSQ problem. Both algorithms are empowered by the foundation established in Section 3; they utilize Lemma 1 and Theorems 1 and 3 to eliminate unnecessary dominance checks for the definite skyline points and Theorem 2 to prune some of query points.

4.1 B²S²: Branch-and-Bound Spatial Skyline Algorithm

Our B²S² algorithm is an improved customization of the original BBS algorithm for "spatial skyline" [8]. Similar to [8], we assume that the data points are indexed by a data-partitioning method such as R-tree. For each data point p, let $mindist(p, A)$ be the sum of distances between p and the points in the set A (i.e., $\sum_{q \in A} D(p,q)$). Likewise, we define $mindist(e, A)$ as the sum of minimum distances between the rectangle e and the points of A (i.e., $\sum_{q \in A} mindist(e,q)$).

Figure 5 shows the pseudo-code of B²S². We describe the algorithm using the set of data points $P = \{p_1, \ldots, p_{13}\}$ and query points $Q = \{q_1, \ldots, q_4\}$ shown in Figure 6. Each intermediate entry e_i in the corresponding R-tree represents

the minimum bounding box of the node N_i. B^{2}S^2 starts by computing the convex hull of Q and determines the set of its vertices $CH_v(Q)$ (e.g., $CH_v(Q) = \{q_1, q_2, q_3\}$). Subsequently, B^{2}S^2 begins to traverse the R-tree from its root R down to the leaves. It maintains a minheap H sorted based on the *mindist* values of the visited nodes. Table 2 shows the contents of H at each step. First, B^{2}S^2 inserts $(e_6, mindist(e_6, CH_v(Q)))$ and $(e_7, mindist(e_7, CH_v(Q)))$ corresponding to the entries of the root R into H. Then, e_6 with the minimum *mindist* is removed from H and its children e_1, e_2, and e_3 together with their *mindist* values are inserted into H. Similarly, e_1 is removed and the children of e_1 are added to H. In the next iteration, the first entry p_2 is inside $CH(Q)$ and hence is added to $S(Q)$ as the first skyline point found.

Once the first skyline point is found (i.e., $S(Q) \neq \varnothing$), any entry e must be checked for dominance before insertion into and after removal from H. If e is dominated by a skyline point p, then B^{2}S^2 discards e. This dominance check is done against all points p in $S(Q)$; e is dominated by p if e is completely inside the dominance region of p (see Section 2.2 for the definition). That is, if e does not intersect with any circle $C(q,p)$ for $q \in CH_v(Q)$. To decrease the use of this costly test, B^{2}S^2 first applies two easier tests: 1) If the entry e does not intersect with the MBR of the union of all circles $C(q,p)$ (termed as $SR(p,Q)$), then p dominates e. Similarly, if e does not intersect with the intersection of all such MBRs each corresponding to a current skyline points in $S(Q)$, then a point in $S(Q)$ dominates e. 2) If e is completely inside the convex hull $CH(Q)$, e cannot be dominated (see Theorem 1). If e does not pass any of the above tests, B^{2}S^2 requires to check e against the entire $S(Q)$.

B^{2}S^2 maintains the rectangle B corresponding to the intersection area described above and updates it when a new skyline point is found (see dotted box in Figure 6). Considering our example, B^{2}S^2 removes p_3 from H. As p_3 is inside B and is not dominated by p_2, it is added to the skyline points and the rectangle B is updated accordingly (lines 11-12 in Figure 5). The next step examines e_2 which is not dominated by current skyline points p_2 and p_3. Among e_2's children, only p_5 is inserted to H. p_4 is outside B (i.e., dominated by p_3) and hence is discarded. Then, B^{2}S^2 removes e_7 and extracts its children e_4 and e_5 as e_7 is not dominated. Entry e_4 does not intersect with B and e_5 is dominated by p_2. Hence, B^{2}S^2 discards both entries. At this step, p_5 is removed and added to the skyline points. The remaining steps discard both dominated entries p_1 and e_3. Finally, the points p_2, p_3 and p_5 consist the final spatial skyline.

4.1.1 B^{2}S^2 Correctness

The correctness of B^{2}S^2 follows that of BBS as both algorithms use the same approach to explore the solution space. B^{2}S^2 benefits from all the properties of BBS. For instance, B^{2}S^2 can also utilize any arbitrary monotone function instead of *mindist*() to sort the entries of its heap. Consequently, B^{2}S^2 is also able to employ any monotone preference function to support *ranked* skyline queries [8].

However, B^{2}S^2 is more efficient than BBS as it reduces the complexity of the costly dominance checks. 1) B^{2}S^2 uses only the query points on the convex hull of Q instead of the entire Q (Theorem 2) which decreases the number of distance computations during dominance checks. In most cases, B^{2}S^2 requires $|CH_v(Q)| < |Q|$ operations to compute distances of each point. 2) B^{2}S^2 utilizes the rectangle B to facilitate the pruning of dominated rectangular entries from $O(|Q|(d^2+|S(Q)|))$ to $O(d^2)$. Instead of computing the minimum distances between e and each $q \in Q$ and comparing them against those of each skyline point in $S(Q)$, B^{2}S^2 first checks whether e and B intersect. 3) B^{2}S^2 does not require any dominance check for points inside $CH(Q)$ (Theorem 1). Moreover, B^{2}S^2 utilizes Theorem 3 to precede its expensive dominance checks with an intersection check between the Voronoi cell $VC(p)$ and $CH(Q)$ when the later is cheaper (e.g., when $|S(Q)|$ and $|Q|$ are large comparing to $CH_v(Q)$ and the number of vertices of $VC(p)$).

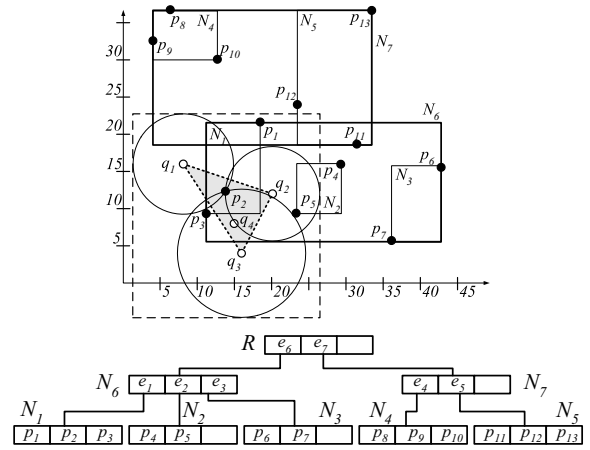


Figure 6: Points indexed by an R-tree

step	heap contents (entry e: $mindist(.,.)$)	$S(Q)$
1	$(e_6 : 8)$, $(e_7 : 52)$	$\varnothing$
2	$(e_1 : 18)$, $(e_2 : 49)$, $(e_7 : 52)$, $(e_3 : 115)$	$\varnothing$
3	$(\mathbf{p_2 : 38})$, $(\mathbf{p_3 : 42})$, $(e_2 : 49)$, $(e_7 : 52)$, $(p_1 : 70)$, $(e_3 : 115)$	$\varnothing$
4	$(e_7 : 52)$, $(p_5 : 53)$, $(p_1 : 70)$, $(e_3 : 115)$	$\{p_2, p_3\}$
5	$(\mathbf{p_5 : 53})$, $(p_1 : 70)$, $(e_3 : 115)$	$\{p_2, p_3\}$
6	$(p_1 : 70)$, $(e_3 : 115)$	$\{p_2, p_3, p_5\}$

Table 2: B^{2}S^2 for the example of Figure 6

4.2 VS^2: Voronoi-based Spatial Skyline Algorithm

According to Theorems 1 and 3, the points whose Voronoi cells are inside or intersect with the convex hull of the query points are skyline points. Therefore, our VS^2 algorithm utilizes the Voronoi diagram (i.e., the corresponding Delaunay graph) of the data points to answer an SSQ problem. We assume that the R-tree on the data points does not exist. Instead, the Voronoi neighbors of each data point is known. To be specific, the adjacency list of the Delaunay graph of the points in P is stored in a flat file. To preserve locality, points are organized in pages according to their Hilbert values.

VS^2 starts traversing the Delaunay graph from a data point (e.g., $NN(q_1)$, the closest point to q_1). The traversal order is determined by the monotone function $mindist(p, CH_v(Q))$. VS^2 maintains two different lists to track the traversal; *Visited* which contains all visited points and *Extracted* which contains those visited points whose Voronoi neighbors have also been visited (extracted). Similar to B^{2}S^2, VS^2 also maintains the rectangle B which includes all candidate skyline points.

Figure 7 shows the pseudo-code of VS^2. It first computes the convex hull of query points and initializes all the data structures. Then, the closest point to one of the query points

Algorithm VS^2 (set Q)
01. compute the convex hull $CH(Q)$;
02. set $S(Q) = \{\}$;
03. Heap $H = \{(NN(q_1), mindist(NN(q_1), CH_v(Q)))\}$;
04. set $Visited = \{NN(q_1)\}$; set $Extracted = \{\}$;
05. box $B = MBR(SR(NN(q_1), Q))$;
06. while H is not empty
07. (p, key) = first entry of H;
08. if $p \in Extracted$
09. remove (p, key) from H;
10. if p is inside $CH(Q)$ or
11. p is not dominated by $S(Q)$
12. add p to $S(Q)$;
13. $B = B \cap MBR(SR(p', Q))$;
14. else
15. add p to $Extracted$;
16. if $S(Q) = \varnothing$ or a Voronoi neighbor of p is in $S(Q)$
17. for each Voronoi neighbor of p such as p'
18. if $p' \in Visited$, discard p';
19. if p' is inside B or $VC(p')$ intersects with B
20. add p' to $Visited$;
21. add $(p', mindist(p', CH_v(Q)))$ to H;
22. return $S(Q)$;

Figure 7: Pseudo-code of the VS^2 algorithm

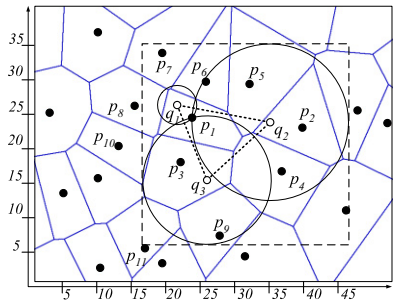


Figure 8: Example points for VS^2

(e.g., $p = NN(q_1)$) and its corresponding entry $(p, mindist(p, CH_v(Q)))$ are added to $Visited$ and minheap H, respectively. Then, VS^2 iteratively examines (p, key), the top entry of H. If all p's Voronoi neighbors have been already visited (i.e., $p \in Extracted$), then (p, key) is removed from H. Subsequently, if p is not dominated by current $S(Q)$, then p is added to $S(Q)$ as a skyline point. If $p \notin Extracted$, p is added to $Extracted$. Moreover, if at least one of the skyline points identified so far is a Voronoi neighbor of p, then VS^2 adds each unvisited Voronoi neighbor p' of p to $Visited$ and H iff a) p' is inside current rectangle B or b) p''s Voronoi cell $VC(p')$ intersects with B. Subsequently, B is updated accordingly and the entry $(p', mindist(p', CH_v(Q)))$ is added to the heap H. Finally, when H becomes empty, VS^2 returns $S(Q)$ as the result.

We describe VS^2 using the example shown in Figure 8. The three query points q_i and the data points are shown as white and black dots, respectively. Table 3 shows the contents of heap H. First, VS^2 adds $(p_1, midist(p_1, CH_v(Q)))$ to H and marks p_1 as visited. B is also initialized accordingly to the dotted box in Figure 8. The first iteration visits p_3, p_4, p_5, p_6, and p_8 as p_1's Voronoi neighbors and adds their corresponding entries to H. It also adds p_1 to the $Extracted$ list. The second iteration removes p_1 from H as p_1 and its neighbors have been already visited. It also adds p_1 to $S(Q)$ as p_1 is inside $CH(Q)$. The third iteration adds p_9, p_{10}, and p_{11} as the only unvisited neighbors of p_3 which are inside B to H. The next two iterations immediately remove p_3 and then p_6 from H and add them to $S(Q)$ as their neighbors have been already visited. The subsequent iterations add p_5, p_4, and p_2 to the skyline and eliminate the remaining entries of H as they are all dominated.

4.2.1 VS^2 Correctness

LEMMA 3. *Given a query set Q, VS^2 first adds the data points inside $CH(Q)$ to the skyline $S(Q)$. All other points are examined and added to $S(Q)$ in ascending order of their mindist() values.*

PROOF. We first show that p_0, the first data point added to $S(Q)$, is inside $CH(Q)$ if there is at least one point inside $CH(Q)$.

The traversal of Delaunay graph once started from point $NN(q_1)$ is towards the point with smaller *mindist*. The reason is that the *mindist* of the top entry of H is always decreasing before p_0 being added to $S(Q)$. It is clear that the *mindist* of any point inside $CH(Q)$ is smaller than those of the points outside $CH(Q)$. Hence, after a few iterations, the top entry of H becomes a point inside $CH(Q)$. Then during the next iterations, the point whose *mindist* is less than those of all of its neighbors is added as the first point to $S(Q)$. As *mindist* is monotone with respect to the distance to each query point q_i, this happens only inside $CH(Q)$. Therefore, p_0 is inside $CH(Q)$. If there is no point inside $CH(Q)$ then $VC(p_0)$ intersects with $CH(Q)$. Once p_0 is added, all other points inside $CH(Q)$ are added to $S(Q)$ as their *mindist* are smaller than those of outside points. Finally, as the pseudo-code of VS^2 shows, subsequent iterations always examine and add the top entry of H which has the minimum *mindist* to $S(Q)$. Hence, the non-dominated data points outside $CH(Q)$ are added to $S(Q)$ in the ascending order of their *mindist*. □

LEMMA 4. *Given a query set Q, VS^2 identifies all spatial skyline points with respect to Q.*

PROOF. We provide only the sketch of the proof. It is clear that VS^2 examines all the data points except those that either their Voronoi cells are outside the rectangle B or none of their already-visited Voronoi neighbors has a skyline neighbor. The first group of points are inside the dominator region of one of the visited points and hence need to be discarded. The second group of points are obviously outside $CH(Q)$ and the Voronoi cell of none of their Voronoi neighbors intersects with $CH(Q)$. These neighbors are all dominated. Voronoi neighbors of these neighbors are also dominated and hence do not intersect with $CH(Q)$. The two layer of dominated Voronoi neighbors around such a point p guarantees that p is also dominated by a point which dominates one of these neighbors. □

Similar to B^{2}S^2, VS^2 utilizes our foundation described in Section 3 to efficiently reduce the number of dominance checks. The number of data points visited by VS^2 is much less than those inside rectangle B. The reason is that VS^2 usually stops its traversal earlier than reaching the boundaries of B.

The time complexity of VS^2 is $O(|S(Q)|^2|CH_v(Q)| + \Phi(|P|))$ where $\Phi(|P|)$ is the complexity of finding the point from which VS^2 starts visiting inside $CH(Q)$ (e.g., $NN(q_1)$). If VS^2 utilize an index structure, then $\Phi(|P|)$ is $O(\log|P|)$. Otherwise, As the Delaunay graph of P is connected, VS^2 starts from a random point and keeps visiting the neighboring point closest to q_1 until it reaches q_1. For a uniformly distributed P in a square-shaped area, this takes $\Phi(|P|) = O(\sqrt{|P|}/2)$ steps.

step	heap contents (point p: $mindist(.,.)$)	$S(Q)$
1	$(p_1 : 24)$	$\varnothing$
2	$(\mathbf{p_1 : 24})$, $(p_3 : 28)$, $(p_6 : 32)$, $(p_5 : 34)$, $(p_4 : 38)$, $(p_8 : 44)$	$\varnothing$
4	$(\mathbf{p_3 : 28})$, $(p_6 : 32)$, $(p_5 : 34)$, $(p_4 : 38)$ $(p_8 : 44)$, $(p_9 : 49)$, $(p_{10} : 49)$, $(p_{11} : 63)$	$\{\mathbf{p_1}\}$
6	$(\mathbf{p_6 : 32})$, $(p_5 : 34)$, $(p_4 : 38)$, $(p_8 : 44)$, $(p_7 : 46)$, $(p_9 : 49)$, $(p_{10} : 49)$, $(p_{11} : 63)$	$\{p_1, \mathbf{p_3}\}$
8	$(\mathbf{p_5 : 34})$, $(p_4 : 38)$, $(p_8 : 44)$, $(p_7 : 46)$, $(p_9 : 49)$, $(p_{10} : 49)$, $(p_{11} : 63)$	$\{p_1, p_3, \mathbf{p_6}\}$
...	...	$\{p_1, p_3, p_6, \mathbf{p_5}, \mathbf{p_4}, \mathbf{p_2}\}$

Table 3: VS^2 for the example of Figure 8

5. CONTINUOUS QUERY

The algorithms proposed in the previous sections are appropriate for the applications where the query points in Q represent the locations of some stationary objects. Hence, the spatial skyline of the data points P with respect to Q once found is not changing. Now consider the scenario in which each query point q_i represents the location of a moving object in the space of $\mathbb{R}^2$. All moving objects frequently report their latest locations. Consequently, we gradually receive the new location of each object as a spatial data stream. Arrival of each new location causes an update to a single point of Q. We intend to maintain the spatial skyline of P with respect to the set of latest locations of all objects (i.e., the current query set Q).

Upon an update to Q, one can always rerun an SSQ algorithm to return the new set of skyline points. However, this approach is expensive specially with the R-tree-based algorithms such as B^{2}S^2 and BBS where the entire R-tree must be traversed per each update. These algorithms cannot tolerate updates for a fast rate data stream.

A smarter solution is to update the set of previously found skyline points when a query point q moves. With B^{2}S^2 and BBS, this still requires a complete traversal of R-tree and examining all candidate data points. However, a more efficient approach must examine only those data points which may change the spatial skyline according to q's new location. In the following, we first identify the subset of query points on which the dominance of a data point p depends. Then, we prove Lemma 6 which states that q affects the dominance of p iff p is in the line-of-sight of q. That is, the line segment pq does not intersect with the interior of $CH(Q)$. We term *visible region* of q to refer to the locus of all such points p. Finally, in Section 5.1 we identify the points which may change the skyline for different moves of q and propose our algorithm for continuous SSQ. The algorithm utilizes Lemma 6 to choose the data points that must be examined when the location of q changes.

First, we find the query points which may change the dominance of a data point outside $CH(Q)$. Consider the data point p and query points $Q = \{q_1, \ldots, q_4\}$ in Figure 9. The two tangent lines from p to $CH(Q)$, divide the vertices in $CH_v(Q)$ into two disjoint sets $CH_v^+(Q) = \{q_1, q_2, q_3\}$ and $CH_v^-(Q) = \{q_4\}$. The sets $CH_v^+(Q)$ and $CH_v^-(Q)$ include the vertices on the closer and farther hulls of the convex hull $CH(Q)$ to p, respectively. The following lemma proves that only the query points in the closer hull to p can change the dominance of p.

LEMMA 5. *Given a data point p and a query set Q, the dominance of p only depends on the query points in $CH_v^+(Q)$.*

PROOF. Suppose a point p' dominates p with respect to $CH_v^+(Q)$. It is clear that the bisector line of pp' which separates p and $CH_v^+(Q)$ intersects with the tangent lines from p to $CH(Q)$. Hence, it also separates the entire $CH(Q)$ from p. Therefore, p is dominated with respect to Q too. The proof of the reverse case is also similar. Therefore, p is (not) dominated with respect to Q iff p is (not) dominated with respect to $CH_v^+(Q)$. Hence, the dominance of p depends only on $CH_v^+(Q)$ (not $CH_v^-(Q)$). □

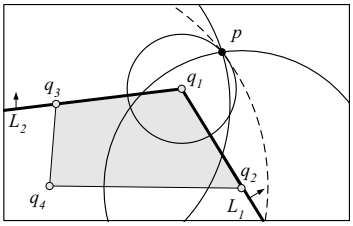


Figure 9: Visible region of $q_1 \in CH(Q)$

Lemma 5 states that the dominator region of p does not depend on $q \notin CH_v^+(Q)$. That is, this region is not changed by the circle $C(q, p)$ if $q \in CH_v^-(Q)$. Figure 9 illustrates this case where the dotted circle $C(q_4, p)$ does not contribute to the dominator region of p.

Utilizing Lemma 5, we easily find the locus of all points whose dominance depends on the location of a given query point. Consider q_1 in the example shown in Figure 9. The lines L_1 and L_2 pass through the line segments q_1q_2 and q_1q_3, respectively (q_2 and q_3 are immediate neighbors of q_1 on $CH(Q)$). Each line divides the space into two half planes. The figure illustrates the half planes using arrows. The union of the half planes which does not contain $CH(Q)$ (visible region of q_1) is the locus of the data points whose dominance does depend on q_1. The reason is that for any point such as p in this region, $CH_v^+(Q)$ does not contain q_1 and hence according to Lemma 5 p's dominance does not depend on q_1. The general statement of the above result follows.

LEMMA 6. *The locus of data points whose dominance depends on $q \in CH_v(Q)$ is the visible region of q.*

5.1 Voronoi-based Continuous SSQ (VCS^2)

Assume that the location of the query point q changes. We use q' and Q' to refer to the new location of q and the new set of query locations, respectively ($Q' = Q \cup \{q'\} - \{q\}$). In this section, we propose our Voronoi-based Continuous Spatial Skyline algorithm (VCS^2). We show how VCS^2 uses the visible regions of q and q' to partition the space into several regions for each a specific update must be applied to the old skyline. VCS^2 is more efficient than the naive approach as it avoids excessive unnecessary dominance checks.

The algorithm traverses the Delaunay graph of the data points similar to VS^2. It first computes $CH(Q')$, the convex hull of the latest query set. Then, it compares $CH(Q')$ with the old convex hull $CH(Q)$. Depending on how $CH(Q')$ differs from $CH(Q)$, VCS^2 decides to either traverse only specific portions of the graph and update the old skyline $S(Q)$ or rerun VS^2 and generate a new one. With the former case, it tries to examine only those points on the graph whose dominance changes because of q's movement.

We now describe the situations where VCS^2 updates the skyline based on the change in $CH(Q')$. Later, we show how the update is applied. We first enumerate different ways

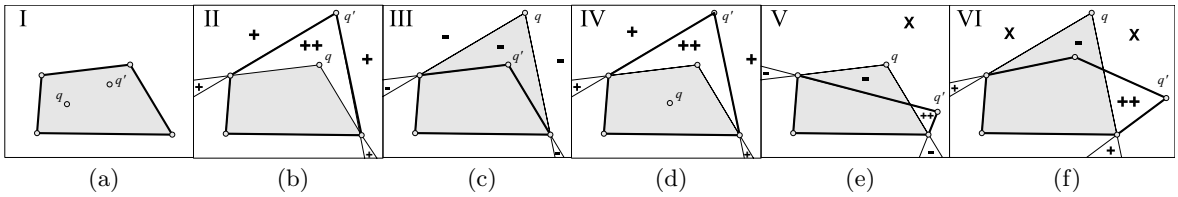


Figure 10: Change patterns of convex hull of Q when the location of q changes to q'

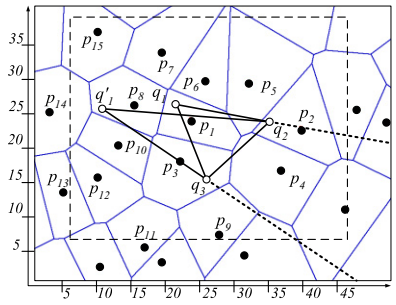


Figure 11: Example points for VCS^2

that the convex hull of query points might change when a query point q moves. Figures 10a-10f show only six change patterns. Each figure illustrates a case where q's location changes to q' (i.e., q moves to q'). The grey and the thick-edged polygons show $CH(Q)$ and $CH(Q')$, respectively. In general, identifying the appropriate pattern by comparing $CH(Q)$ and $CH(Q')$ is not a straightforward task. Therefore, VCS^2 only tries to recognize specific simple patterns **I-V** and subsequently updates the skyline accordingly as we explain later. For all other change patterns such as pattern f, it reruns VS^2. For each of patterns **I-V**, specific portions of the Delaunay graph must be traversed by VCS^2:

Pattern I) $CH(Q)$ and $CH(q')$ are equal as both q and q' are inside the convex hull. According to Theorem 2, the skyline does not change and no graph traversal is required.

Patterns II-V) The visible regions of $q \in CH_v(Q)$ and $q' \in CH_v(Q')$ together partition the space into six regions (seven regions for pattern **V**). The intersection region of $CH(Q)$ and $CH(Q')$ contains data points which are in skyline with respect to both Q and Q'. Therefore, VS^2 does not traverse this portion of the Delaunay graph. The region labeled by "++" includes the points inside $CH(Q')$ and outside $CH(Q)$. According to Theorem 1, any point in this region is a skyline point with respect to Q' and hence must be added to the skyline. The points in the regions labeled by "+" might be skyline points and must be examined. The intuition here is that their dominator regions have become smaller because of q's movement. The regions labeled by "−" contain the points whose dominator regions have been expanded and hence might be deleted from the old skyline. The points in the regions specified by "×" must be examined for inclusion in or exclusion from the skyline as their dominator regions have changed. Finally, neither q nor q' affects the dominance of the points in the unlabeled white region. The reason is that this region is outside of the visible regions of both q and q' (Lemma 6).

For each pattern **I-V**, only the data points in the union of the labeled regions might change the skyline. We use the term *candidate* region to refer to this region. Once VCS^2 identifies any of patterns **I-V**, it tries to update the skyline. The algorithm first assigns the old $S(Q)$ to $S(Q')$. For pattern **I**, VCS^2 returns the same old skyline as it is still valid. For patterns **II-V**, VCS^2 starts traversing the Delaunay graph from the closest data point to q'. Similar to VS^2, it initializes the rectangle B and traverses the points which are inside the intersection of the candidate region and B ordered by their *mindist* values. At each iteration, if the point with the minimum *mindist* has been dominated it is deleted from $S(Q')$ otherwise it is added to $S(Q')$.

At the end, VCS^2 evaluates the skyline $S(Q')$ and removes all the points which are dominated by another point in $S(Q')$. The reason behind this final check is to examine the old skyline points which are not in the traversal range of VCS^2.

We show the effectiveness of VCS^2 using the example of Figure 11 where shows q'_1 the new location of q_1 in Figure 11. First, VCS^2 computes the convex hull of $Q' = \{q'_1, q_2, q_3\}$ and compares it with $CH(Q)$. The change pattern matches pattern **V** in Figure 10. Therefore, the update to $S(Q)$ involves both insertion and deletion. Then, VCS^2 initializes $S(Q')$ to the old $S(Q)$ resulted from applying VS^2 in Section 4.2 and rectangle B accordingly. It also adds $(p_8, mindist(p_8, CH_v(Q')))$ in to H where p_8 is the closest point to q'_1, the new location of q_1. Table 4 shows the contents of the minheap H. The second iteration extracts p_8 and adds all of its Voronoi neighbors except p_1 in to H as p_1 is not in the candidate region of pattern **V**. Similarly, p_3 is extracted and all its neighbors except p_4 is added in to H. Next, as p_3 is not dominated it remains in $S(Q')$. The next two iterations add p_8 and p_{10} to $S(Q')$. The sixth iteration, visits p_6, the only unvisited Voronoi neighbor of p_7 in B which is subsequently removed from $S(Q')$ as it is dominated by $p_1 \in S(Q')$. The final four iterations of VCS^2 also eliminate remaining points in H as they are all dominated. No point in the final skyline set is dominated and hence VCS^2 returns $S(Q')$ as the result. The above example shows how VCS^2 avoids dominance checks for points such as p_4 outside the candidate region of the identified change pattern and p_1 inside the convex hull of query points. This proves VCS^2's superiority over the naive approach.

5.1.1 VCS^2 Correctness

The correctness of VCS^2 follows that of VS^2. The intuition is that VCS^2 also examines all the candidate skyline points. It also examines those old skyline points that are now dominated with respect to the new query set Q' and removes them from the skyline.

6. NON-SPATIAL ATTRIBUTES

One might be interested to find the skyline with respect to both static *non-spatial* attributes of the data points and their distances to points of Q. For instance, the best restaurant in LA might be dominated in terms of distance to our team members but it is still in the skyline considering its rating. We show how B^{2}S^2, VS^2, and VCS^2 can support this variation of SSQ. Let A be a subset of non-spatial at-

step	heap contents	$S(Q')$
1	$(p_8 : 39)$	$\{p_1, p_3, p_6, p_5, p_4, p_2\}$
2	$(p_3 : 32)$, $(p_8 : 39)$, $(p_{10} : 42)$, $(p_7 : 50)$, $(p_{14} : 64)$, $(p_{15} : 66)$	$\{p_1, p_3, p_6, p_5, p_4, p_2\}$
3	$(\mathbf{p_3 : 32})$, $(p_8 : 39)$, $(p_{10} : 42)$, $(p_7 : 50)$, $(p_9 : 51)$, $(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$	$\{p_1, p_3, p_6, p_5, p_4, p_2\}$
4	$(\mathbf{p_8 : 39})$, $(p_{10} : 42)$, $(p_7 : 50)$, $(p_9 : 51)$, $(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$	$\{p_1, p_3, p_6, p_5, p_4, p_2\}$
5	$(\mathbf{p_{10} : 42})$, $(p_7 : 50)$, $(p_9 : 51)$, $(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$	$\{p_1, p_3, p_6, p_5 p_4, p_2, \mathbf{p_8}\}$
6	$(p_7 : 50)$, $(p_9 : 51)$, $(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$	$\{p_1, p_3, p_6, p_5, p_4, p_2, p_8, \mathbf{p_{10}}\}$
7	$(\mathbf{p_6 : 41})$, $(p_7 : 50)$, $(p_9 : 51)$, $(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$	$\{p_1, p_3, p_6, p_5, p_4, p_2, p_8, p_{10}\}$
8	$(p_7 : 50)$, $(p_9 : 51)$, $(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$	$\{p_1, p_3, \mathbf{p_6}, p_5, p_4, p_2, p_8, p_{10}\}$
9	$(p_9 : 51)$, $(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$	$\{p_1, p_3, p_5, p_4, p_2, p_8, p_{10}\}$
10	$(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$	$\{p_1, p_3, p_5, p_4, p_2, p_8, p_{10}\}$
11	$(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$, $(p_{13} : 66)$	$\{p_1, p_3, p_5, p_4, p_2, p_8, p_{10}\}$

Table 4: VCS^2 for the example of Figure 11

Points	*Size*	*Points*	*Size*
1) Hospital	0.56%	5) Church	13%
2) Building	1.6%	6) School	15%
3) Summit	7%	7) Populated place	18%
4) Cemetery	12%	8) Institution	34%

Table 5: USGS data (http://geonames.usgs.gov/)

tributes of data points of P. Assume that $S(A)$ is the set of skyline points considering only the non-spatial attributes in A. Also, let $S(A, Q)$ be the skyline when both attributes in A and distances to query points in Q are considered. Consequently, we have $S(A) \subset S(A, Q)$ and $S(Q) \subset S(A, Q)$. Our algorithms can easily be changed to find $S(A, Q)$ as follows. 1) We first use a general skyline algorithm to find $S(A)$. Notice that this is a batch one-time computation independent from the query (i.e., Q). 2) We modify each of our algorithms such that the domination check for each point inside the search region and outside $CH(Q)$ considers both its non-spatial attributes and its distances to the points of $CH_v(Q)$. 3) The search region (i.e., rectangle B) is extended to examine all possible candidate points. The following lemma specifies the limits of the new search region.

LEMMA 7. *Any point p farther than all points in $S(A)$ from all $q_i \in Q$, cannot be a skyline point with respect to attributes in A and query points in Q. That is,*

$$\forall p' \in S(A), \forall q_i \in Q, D(p, q_i) \geq D(p', q_i) \Rightarrow p \notin S(A, Q)$$

Utilizing Lemma 7, we change the rectangle B to the MBR of the points violating the above equation. This is straightforward as the region is the union of circles $C(q_i, p)$ where $p \in S(A)$. Therefore, our B^{2}S^2, VS^2, and VCS^2 algorithms answer SSQs when mixed with non-spatial attributes.

7. PERFORMANCE EVALUATION

We conducted several experiments to evaluate the performance of our proposed approaches. In real-world applications with a small number of *original* attributes, the traditional BBS approach outperforms algorithms such as BNL [1]. Hence, we only compared our algorithms with BBS. First, we compared both B^{2}S^2 and VS^2 with the BBS approach, with respect to: 1) overall query response time, and 2) number of dominance checks. Moreover, we compared the disk I/O accesses incurred by the underlying R-tree index

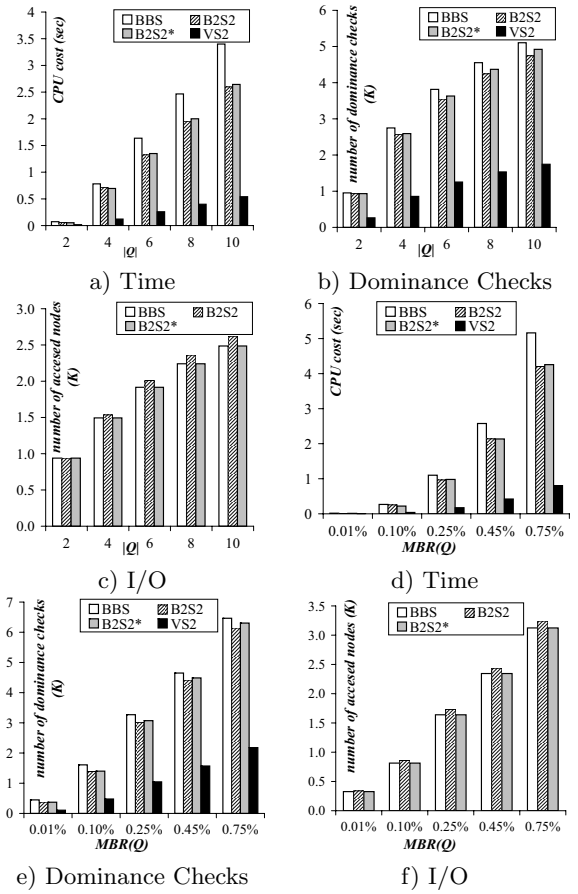


Figure 12: Query cost vs. number of query points and the area covered by Q

structures used by B^{2}S^2 with those of BBS. We evaluated all approaches by investigating the effect of the following parameters on their performances: 1) number of query points (i.e., $|Q|$), 2) the area covered by MBR of Q, 3) cardinality of the dataset, and 4) density of the data points. We also evaluated the performance of VCS^2 using a synthetic dataset of moving objects.

We used a real dataset for our experiments. The dataset is obtained from the U.S. Geological Survey (USGS) and consists of 950,000 locations of eight different businesses in the entire country. Table 5 shows the characteristics of this dataset which is indexed by an R*-tree index with the page size of 1K bytes and a maximum of 50 entries in each node (capacity of the node). The index structure is based on the original attributes of the data points (i.e., latitude and longitude) and is utilized by both BBS and B^{2}S^2 approaches. VS^2 and VCS^2 use a pre-built Delaunay graph of the entire dataset. We also used the Graham Scan algorithm [3] for convex hull computation in both VS^2 and VCS^2. The experiments were performed on a DELL Precision 470 with Xeon 3.2 GHz processor and 3GB of RAM. We ran 1000 SSQ queries using randomly selected query points and reported the average of the results.

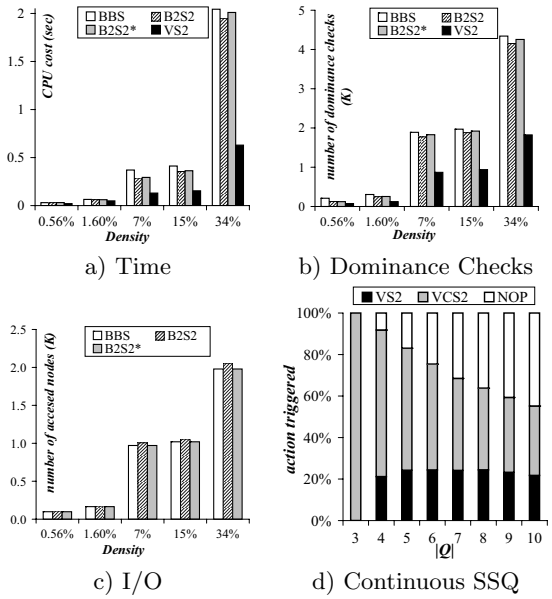
In the first set of experiments, we varied the number of query points and studied the performance of our proposed algorithms. We set the *maximum $MBR(Q)$* to 0.3% of the entire dataset. Figure 12a depicts query response time of B^{2}S^2, B^{2}S^2*, VS^2 and BBS where B^{2}S^2* is a variation of B^{2}S^2 in which we use $mindist(e, Q)$ instead of $mindist(e,$

$CH_v(Q)$) to sort the heap entries. Therefore, both B^2S^2* and BBS find the skyline points in the same order. Notice that B^2S^2 can employ any monotone function. As shown Figure 12a, as the superior algorithm VS^2 significantly outperforms BBS by a wide margin (3-6 times better in most cases). Given four query points, VS^2 finds the skyline in only 0.12 seconds while BBS requires 0.78 seconds for the same query. While the figure illustrates the results for SSQ with respect to up to 10 query points, the trend shows that VS^2 performs significantly better than BBS as the number of query points increases. The figure indicates that while both B^2S^2 and B^2S^2* outperform BBS, B^2S^2 is faster. The reason is that while calculating *mindist*, it computes only the distances to convex query points of Q instead of the entire Q (on average 6 points where $|Q| = 10$).

Figure 12b shows the average number of dominance checks performed by each algorithm. As expected, VS^2 constantly performs only 65%-75% of the dominance checks performed by BBS. Considering the fact that each VS^2's check is also faster than BBS's, this explains the superiority of VS^2 in terms of performance. The figure shows that both B^2S^2 and B^2S^2* require less checks than BBS which shows the efficiency of utilizing the rectangle B to avoid unnecessary checks (see Section 4.1). Figure 12c illustrates the number of R*-tree nodes accessed by B^2S^2, B^2S^2*, and BBS. As the figure shows, BBS and B^2S^2* access exactly the same nodes as they both use a common *mindist* function during the search. B^2S^2's use of a different function results into slightly more node accesses which has no impact on its overall performance as shown in Figure 12b. The reason here is that 1) the total number of dominance checks of B^2S^2 and B^2S^2* are less than that of BBS (see Figure 12b) and, 2) each check in B^2S^2 and B^2S^2* requires distance computation only for the convex query points. Even the extra convex hull computation in our algorithms does not downgrade their performance.

The next set of experiments investigates the impact of closeness of the query points on the performance of each algorithm. We varied the area covered by the MBR of Q from 0.01% to 0.75% of the entire USGS dataset (i.e., approximately 90 to 7K data points in the MBR). Figures 12d-f depicts the average query response time, number of dominance checks, and R*-tree node accesses for all the algorithms, respectively ($|Q| = 6$). As the query points get farther, more points need to be examined by an SSQ algorithm which downgrades its performance. The results show a trend similar to those of the first experiment; VS^2 is the superior algorithm. It performs only 33% of BBS's dominance checks which makes it by one order of magnitude faster than BBS even for highly scattered query points (i.e., $MBR(Q) = 0.75\%$). The reason is that as the MBR of Q grows, BBS requires more R-tree nodes to examine. However, VS^2 once enters $CH(Q)$ performs computation only in a local neighborhood. Likewise, B^2S^2 demonstrates superiority over BBS.

The third set of experiments focuses on the performance of the algorithms when the density of the data points changes. We used five different point types from Table 5 with hospitals and institutions as the most sparse and dense points, respectively. Figures 13a-c illustrates the results for $|Q| = 6$ and maximum $MBR(Q) = 0.5\%$. As expected, it always takes more time and I/O to find skyline for denser data points. The reason is that for denser data more skyline



a) Time b) Dominance Checks

c) I/O d) Continuous SSQ

Figure 13: Query cost and continuous SSQ

points correspond to a query set Q with fixed-size MBR and hence all proposed algorithms must examine more points. This experiment also shows that the cardinality of the dataset has the same impact as that of the density on the performance of all the algorithms.

Our next set of experiments studies the performance of VCS^2. We used GSTD [11] to generate trajectories of 3-10 moving query points where $MBR(Q) = 0.3\%$. The movements of query points obey a uniform distribution and follows a moderate speed. For different query sizes $|Q|$, we counted the number of moves (out of 1000 moves per query point) for which VCS^2 1) identifies the movement pattern **I** and hence does nothing (shown as **NOP**), 2) detects movement patterns **II**-**V** and updates the old skyline (shown as VCS^2), or 3) re-runs VS^2 (shown as VS^2). Figure 13d shows the percentage of the movements with the corresponding triggered actions. Surprisingly, the figure shows that on average for all query sizes only less than 25% of movements require the entire skyline to be recomputed (i.e., re-running VS^2). For three query points, this never happens as any movement follows one of the patterns **II**, **III** or **V**. The figure also verifies that as the number of query points increases, the chance that a movement changes the skyline decreases. We separately measured the average query response time of VCS^2 as only 35% of that of VS^2 for all query sizes. Therefore, this experiment justifies the superiority of VCS^2 for continuous SSQ over VS^2.

Finally, Table 6 shows the results of our last set of experiments with some extreme cases: spatial skyline of 1) all U.S. schools with respect to 3,000 summits in Colorado whose MBR covers about 2.5% of the entire U.S. (large Q), 2) all U.S. hospitals with respect to all U.S. summits ($|Q| \gg |P|$), and 3) all U.S. summits with respect to all buildings of USGS dataset (large Q). Notice that in the last two cases, both data and query sets cover the same area (i.e., entire U.S.). Our results show the superiority of both VS^2 and B^2S^2 over BBS. When the size of query set is significantly more than that of data, the overhead of convex hull computation becomes the dominant factor in the performance of VS^2 and B^2S^2. However, these algorithms still outperform BBS as

Data (# points)	Query (# points)	CPU (sec.)			# of Dominance Checks		
		BBS	B^2S^2	VS^2	BBS	B^2S^2	VS^2
Schools (139,523)	Colorado Summits (3,171)	31.4	6.7	6.5	3,745	3,460	1,353
Hospitals (5,314)	Summits (69,498)	22,171	1,379	1,314	10,629	5,726	5,167
Summits (69,498)	Buildings (15,126)	>30,000	2,212	1,427	-	143,222	65,749

Table 6: Experimental results for extreme scenarios

even with large query sets the number of convex points is significantly small (e.g., only 19 points for 69,498 points in the second case). This highly reduces the cost of dominance checks performed by our algorithms. In contrast, each dominance check in BBS involves a large number of distance computations ($2\times69,498$ computations in the second case).

8. RELATED WORK

The best known skyline algorithm that can answer SSQs is the Branch-and-Bound Skyline algorithm (BBS) [8]. BBS is a progressive optimal algorithm for the *general* skyline query. In the setting of BBS, a *dynamic* skyline query specifies a new n-dimensional space based on the original d-dimensional data space. First, each point p in the database is mapped to the point $\hat{p} = (f_1(p), \ldots, f_n(p))$ where each f_i is a function of the coordinates of p. Then, the query returns the general (i.e., static) skyline of the new space (the corresponding points in the original space). We can define the spatial skyline query as a special case of the dynamic query. Given the query set Q, we use $f_i = D(p, q_i)$ to map each point p to $\hat{p}$. Therefore, our spatial skyline can be defined as a special case of dynamic skyline presented in [8]. While BBS is a nice general algorithm for any function f, since it has no knowledge of the geometry of the problem space, it is not as efficient as our proposed algorithms for the spatial case when f is a distance function.

The Block Nested Loop (BNL) approach [1] for general skyline computation can also address SSQs. BNL outperforms BBS when the number of skyline points is small and the dimensionality is high [8]. Although with a large set of query points for SSQ the number of *derived* attributes is high, the number of original attributes which are used by BBS remains unchanged; the points usually have only two dimensions in real-world examples. This makes BBS to be the best competitor approach and hence we compared our techniques only with BBS in Section 7.

There are studies in the area of spatial databases related to SSQ. Papadias et al. [9] proposed efficient algorithms to find the point with minimum aggregate distance to query points in Q. The aggregate distance is a monotone function over distances to points of Q. The optimum point is *one* of the spatial skyline points with respect to Q. Their algorithm seeks only for the optimum point so no dominance check is required. Huang and Jensen [4] studied the interesting problem of finding locations of interest which are not dominated with respect to only two attributes: their network distance to a single query location q and the detour distance from q's predefined route through the road network. Their proposed algorithms rely on existing nearest neighbor and range query approaches to find a candidate set. They then apply naive in-memory skyline computation on the candidate set to extract the result. While their in-route skyline query is distance-based similar to SSQ, it focuses on a specific application. Our SSQ problem however targets a different and broader range of applications for which we propose efficient customized skyline computation algorithms in vector space.

9. CONCLUSIONS AND FUTURE WORK

We introduced the novel concept of spatial skyline queries. We studied the geometric properties of the solution to these queries. We analytically proved that a set of definite skyline points exists. These are the points whose Voronoi cells are inside or intersect with the convex hull of query points (see Theorems 1 and 3 in Section 3). We also proved that the locations of those query points inside the convex hull of all query points have no effect on the final spatial skyline (Theorem 2). Based on these theoretical findings, we proposed two efficient algorithms for SSQ considering static query points. Through extensive experiments, we showed the superiority of our algorithms over the traditional BBS approach. Our VS^2 algorithm exhibits up to 6 times faster performance improvement over BBS.

We also studied a variation of the SSQ problem for moving query points. Our VCS^2 algorithm effectively updates the spatial skyline upon any change to the locations of the query points. Our approach exhibits a significantly better performance than re-computing the skyline even using our efficient VS^2 algorithm. Finally, we showed how all three proposed algorithms can address the SSQ problem when mixed with non-spatial attributes.

We intend to study the challenges of SSQ in metric spaces such as road networks. Intuitively, similar geometric properties corresponding to those we proved in Section 3 for a vector space can be proved for metric spaces. We plan to utilize these properties to propose efficient SSQ algorithms for road network databases.

10. REFERENCES

[1] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proceedings of ICDE'01*, pages 421–430, 2001.

[2] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *Proceedings of ICDE'03*, pages 717–816. IEEE Computer Society, 2003.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer Verlag, 2nd edition, 2000.

[4] X. Huang and C. S. Jensen. In-Route Skyline Querying for Location-based Services. In *4th International Workshop on Web and Wireless Geographical Information Systems (W2GIS'04)*, volume 3428, pages 120–135. Springer, 2004.

[5] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline Queries Against Mobile Lightweight Devices in MANETs. In *Proceedings of ICDE'06*. IEEE Computer Society, 2006.

[6] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proceedings of VLDB'02*, pages 275–286, 2002.

[7] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the Sky: Efficient Skyline Computation over Sliding Windows. In *Proceedings of ICDE'05*, pages 502–513. IEEE Computer Society, 2005.

[8] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive Skyline Computation in Database Systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[9] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate Nearest Neighbor Queries in Spatial Databases. *ACM Trans. Database Syst.*, 30(2):529–576, 2005.

[10] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient Progressive Skyline Computation. In *Proceedings of VLDB'01*, pages 301–310, 2001.

[11] Y. Theodoridis and M. A. Nascimento. Generating Spatiotemporal Datasets on the WWW. *SIGMOD Record*, 29(3):39–43, 2000.