THE SPATIAL SKYLINE QUERIES

Group 9: Vu Thi Diem Quynh 20197100, Nguyen Van Manh 20195088, Pham Thanh Nam 20191578

Hanoi University of Science and Technology

June 23, 2023

1. Spatial Skyline Query Problem

1.1 Problem Definitions

Concept of **Spatial Skyline Queries (SSQ)**:

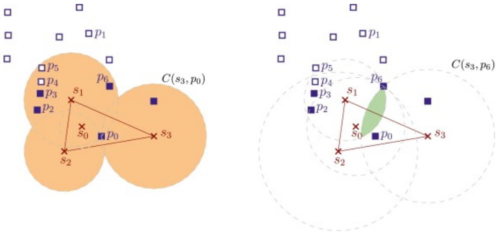
- Given a set of **data points P** and a set of **query points Q** , each data point has a number of *derived spatial attributes* (point's distance to a query point). An **SSQ** retrieves those points of P which are *not dominated* by any other point in P considering their derived spatial attributes.
- An interesting variation: an SSQ with the domination is determined with respect to both *spatial* and *non-spatial* attributes of P.

Differences between **Skyline Queries** and **Spatial Skyline Queries**:

- **Spatial skyline** points' distance attributes are *dynamically* calculated based on the user's query. The result depends on both *data* and the given *query*. Meanwhile **skyline** points, the result only depends on *database*.

→ The main difference with the regular skyline query is that this spatial domination depends on the location of the query points Q.

Problem illustration:



- Assume that the members of a team located in different (fixed) offices, determine a list of interesting (in terms of traveling distances) restaurants for their weekly meeting lunchs.
- Generating this list becomes more challenging when the team members change location over time.

source: Binay Bhattacharya, Arijit Bishnu, Otfried Cheong, Sandip Das, Arindam Karmakar, Jack Snoeyink, Computation of spatial skyline points, Volume 93, February 2021.

2. Formal Problem Definition

2.1 General Skyline Query

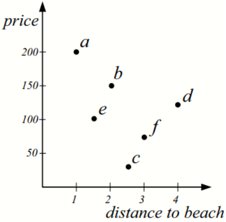
Assume that:

- **N**: number of objects in database (denoted by p).
- **p** with **d** real-valued attributes: d-dimensional point $(p_1, ..., p_d) \in \mathbb{R}^d$ where p_i is the i-th attribute of p.
- Given the two points $p = (p_1, ..., p_d)$ and $p' = (p'_1, ..., p'_d)$ in $\mathbb{R}^d$, **p dominates p'** iff we have $pi \leq p'i$ for $1 \leq i \leq d$ and $p_j < p'_j$ j for some $1 \leq j \leq d$.

For example:

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)

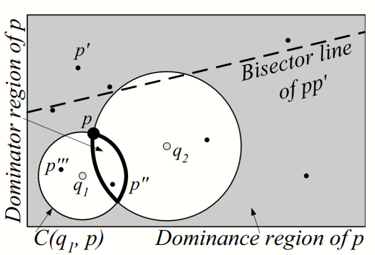
- Given a set of points P={a,b,c,d,e,f}.
- **The skyline** of P is the set of those points of P which are not dominated by any other point in P.
- The point f=(3,75) dominates the point d=(4,125) ($3 < 4$ and $75 < 125$).
- Similarly, we eventually get the skyline of the points is the set $S = \{a,c,e\}$.

2.2 Spatial Skyline Query

Assume that:

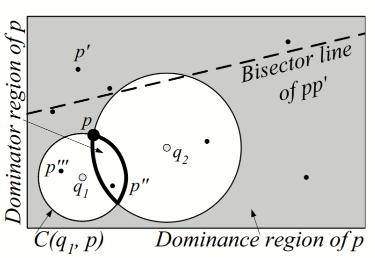
- **P**: set of points in the d-dimensional space. $\mathbb{R}^d$
- **D(.,.)**: distance metric defined in $\mathbb{R}^d$.
- **Q=**$\{q_1, ..., q_n\}$: set of d-dimensional query points.
- Two points **p** and **p'** in $\mathbb{R}_d$
- p **spatially dominates** p' with respect to **Q** iff we have $D(p, q_i) \leq D(p', q_i)$ for all $q_i \in Q$ and $D(p, q_j) < D(p', q_j)$ for some $q_j \in Q$.

For example:



- Given 9 2-d points and 2 query points q1 and q2.
- The point p **spatially dominates** the point p' (both q1 and q2 are closer to p than to p')
- Geometric interpretation of the spatial dominance relation between two points: draw the perpendicular bisector line of the line segment pp', then q1, q2, and p will be located on the same side of the bisector line.

For example:

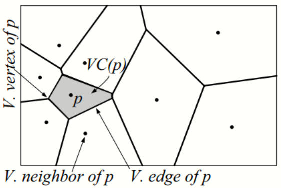


- $\mathbf{C}(q_i, p)$ centered at the query point q_i with radius $\mathbf{D}(q_i, p)$.
- p is in the spatial skyline iff we have:
  $\forall p' \in P, p' \neq p, \exists q_i \in Q$ s.t. $D(p, q_i) \leq D(p', q_i)$.
- p" which is inside the intersection of all $C(qi, p)$ for all $q_i \in Q$, spatially dominates p.

3. Theoretical Foundation

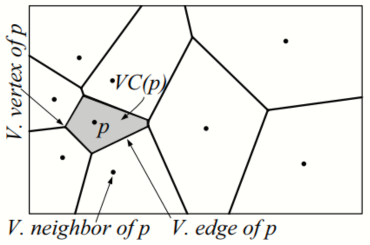
3.1 Concept of Voronoi Diagram, Delaunay Graph, and Convex Hull.

Voronoi Diagram Definition:



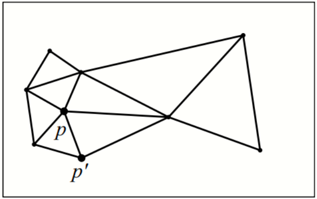
- **P** $= p_1, p_2, ..., p_n$: set of n distinct points in d-dimensional Euclidean space.
- **Voronoi diagram of P**: the subdivision of the Euclidean space into n cells, one for each point in P. The cell corresponding to the point $p \in P$ (denoted by **VC(p)**) contains all the points $x \in \mathbb{R}^d$ s.t $\forall p' \in P, p' \neq p, D(x, p) \leq D(x, p')$.

Note that:



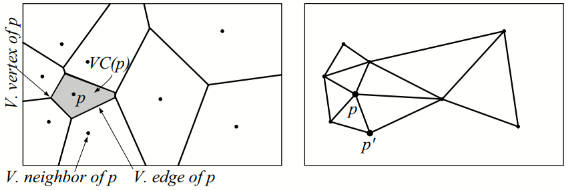
- For Euclidean distance in $\mathbb{R}^2$, VC(p) is a convex polygon.
- Each edge of this polygon (*Voronoi edge*) is a segment of the *perpendicular bisector line* of the line segment connecting p to another point of the set P.
- Each *Voronoi edge* of the point p refers to the corresponding point in the set P as a **Voronoi neighbor** of p.

Delaunay Triangulation and Delaunay Graph Definition:



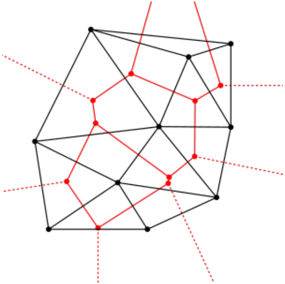
- **Delaunay Triangulation** is a triangulation of a set of points P such that *the circumcircle of each triangle* contains no other points in its interior.
- It *maximizes* the *minimum angle* of the triangles and *minimizes* the *maximum circumradius* of the triangles.
- The **Delaunay Graph** can be obtained from the Delaunay Triangulation.

Duality of Voronoi Diagram and Delaunay Graph:



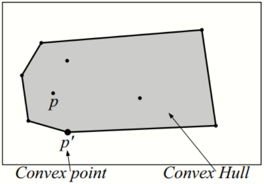
- The Delaunay Graph of the set of points P is the dual of the Voronoi Diagram of P.
- **G(V,E)**: an undirected graph with the set of vertices V = P.
- For each two points p and p' in V , there is an edge connecting p and p' in G iff p' is a Voronoi neighbor of p in the Voronoi diagram of P.

Duality of Voronoi Diagram and Delaunay Graph:



- Each *vertex* in the Voronoi diagram corresponds to a *region* in the Delaunay graph.
- Each *edge* in the Voronoi diagram corresponds to a pair of *adjacent regions* in the Delaunay graph, and each *edge* in the Delaunay graph corresponds to a pair of *adjacent vertices* in the Voronoi diagram.

Convex Hull Definition:

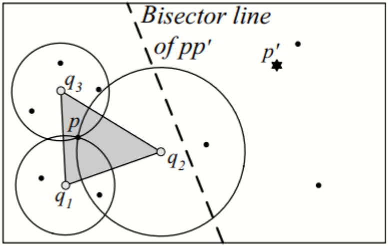


- The **Convex Hull (CH(P))** of points in $P \subset \mathbb{R}^d$, is the unique smallest *convex polytope* (polygon when d = 2) which contains all the points in P.
- $CH_v(P)$ is the set of CH(P)'s vertices, called *convex points*. All other points in P are *non-convex points*.
- The shape of the Convex Hull of a set P only depends on the *convex points* in P.

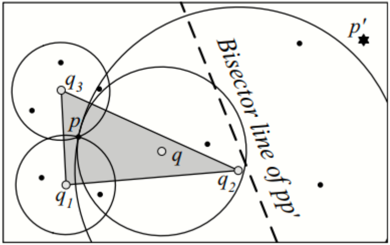
3.2 Theories.

We use lemma (1) and two theorems (1&3) to identify definite skyline points and theorem (2) to eliminate query points not contributing to the search.

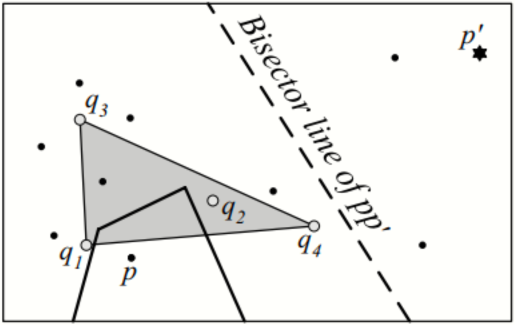
- **Lemma 1**: For each $q_i \in Q$, the *closest* point to q_i in P is a skyline point.
- **Theorem 1**: Any point $p \in P$ which is *inside* the convex hull of Q is a skyline point.



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



- **Theorem 3**: Any point $p \in P$ whose Voronoi cell V C(p) *intersects* with the boundaries of convex hull of Q is a skyline point.



4. Solutions

4.1 Static Query.

4.1.1 Branch-and-Bound Spatial Skyline Algorithm (B^2S^2)

Asume that:

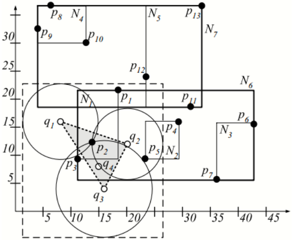
- The data points are indexed by a data-partitioning method such as R-tree.
- mindist(p, A): be the sum of distances between p and the points in the set A ($\Sigma_{q \in A} D(p, q)$).
- mindist(e, A): as the sum of minimum distances between the rectangle e and the points of A ($\Sigma_{q \in A} mindist(e, q)$)
- Data points P = p1,..., p13 and query points Q = q1,..., q4 .

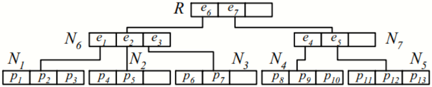
Branch-and-Bound Spatial Skyline Algorithm (B^2S^2)

- Step 1: Compute the *convex hull* of Q and determines the set of its vertices CHv(Q).
- Step 2: traverse the R-tree and maintaining a *minheap* H sorted based on the mindist values of the visited nodes.

$\rightarrow$ How the minheap H works?

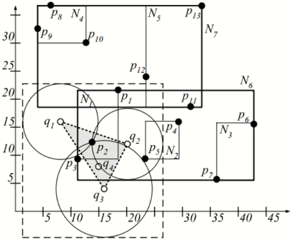
Remind: Minheap is a binary tree data structure. Value of each child node is greater than or equal to the value of its parent node. The smallest value is always stored at the root node.

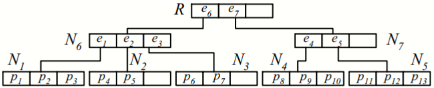




step	heap contents (entry e: $mindist(.,.)$)	$S(Q)$
1	$(e_6 : 8)$, $(e_7 : 52)$	$\varnothing$
2	$(e_1 : 18)$, $(e_2 : 49)$, $(e_7 : 52)$, $(e_3 : 115)$	$\varnothing$
3	$(\mathbf{p_2} : \mathbf{38})$, $(\mathbf{p_3} : \mathbf{42})$, $(e_2 : 49)$, $(e_7 : 52)$, $(p_1 : 70)$, $(e_3 : 115)$	$\varnothing$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

- CHv(Q) = q1, q2, q3 set of vertices the convex hull of Q.
- Insert (e6, mindist(e6, CHv(Q))) and (e7, mindist(e7, CHv(Q))).
- e6 is removed from H and its children e1, e2, and e3 are inserted.
- e1 is removed and the children of e1 are added to H.
- p2 is inside CH(Q) and is added to S(Q) (Theorem 1) as the first skyline point found. **From now, entry e must be checked for dominance before insertion into and after removal from H.**

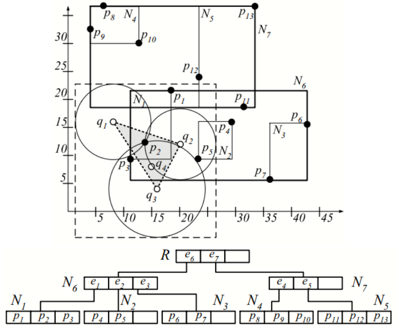




step	heap contents (entry e: $mindist(.,.)$)	$S(Q)$
1	$(e_6:8)$, $(e_7:52)$	$\varnothing$
2	$(e_1:18)$, $(e_2:49)$, $(e_7:52)$, $(e_3:115)$	$\varnothing$
3	$(\mathbf{p_2}:\mathbf{38})$, $(\mathbf{p_3}:\mathbf{42})$, $(e_2:49)$, $(e_7:52)$, $(p_1:70)$, $(e_3:115)$	$\varnothing$
4	$(e_7:52)$, $(p_5:53)$,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

- If e is dominated by a skyline point p, then B^2S^2 discards e.
- e is dominated by p if e does not intersect with any circle C(q, p) for $q \in CH_v(Q)$.
- Two easier tests to decrease cost:
  1. If e does not intersect with the MBR of the union of all circles C(q, p), then p dominates e. (e_3, e_4)
  2. If e is completely inside the convex hull CH(Q), e cannot be dominated (Theorem 1).

  If e does not pass both two tests, B^2S^2 requires to check e against the entire S(Q).



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	$(\mathbf{p_2} : \mathbf{38}), (\mathbf{p_3} : \mathbf{42}), (e_2 : 49), (e_7 : 52), (p_1 : 70), (e_3 : 115)$	$\varnothing$
4	$(e_7 : 52), (p_5 : 53), (p_1 : 70), (e_3 : 115)$	$\{p_2, p_3\}$
5	$(\mathbf{p_5} : \mathbf{53}), (p_1 : 70), (e_3 : 115)$	$\{p_2, p_3\}$
6	$(p_1 : 70), (e_3 : 115)$	$\{p_2, p_3, p_5\}$

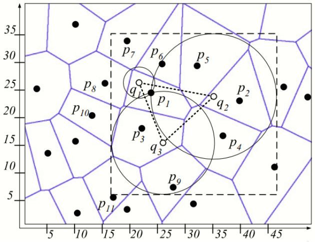
Algorithm $\mathbf{B^2S^2}$ (set Q)

```
compute the convex hull CH(Q);
set S(Q) = {};
box B = MBR(R);
minheap H = {(R, 0)};
while H is not empty
  remove first entry e from H;
  if e does not intersect with B, discard e;
  if e is inside CH(Q) or
      e is not dominated by any point in S(Q)
    if e is a data point p
      add p to S(Q);
      B = B ∩ MBR(SR(p, Q));
    else // e is an intermediate node
      for each child node e' of e
        if e' does not intersect with B, discard e';
        if e' is inside CH(Q) or
          e' is not dominated by any point in S(Q)
          add (e', mindist(e', CH_v(Q))) to H;
return S(Q);
```

4.1.2 Voronoi-based Spatial Skyline Algorithm (VS^2)

Ideal:

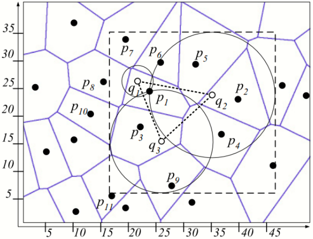
- VS2 algorithm uses the Voronoi diagram (the corresponding Delaunay graph) of the data points.
- R-tree on the data points does not exist.
- The points whose Voronoi cells are inside or intersect with the convex hull of the query points are skyline points.
- The adjacency list of the Delaunay graph is stored according to points' Hilbert values.



Steps:

1. Traverse the Delaunay graph starting from the closest point to a query point and determining traversal order based on a monotone function $mindist(p, CH_v(Q))$.
2. Maintain two lists of visited and extracted points and a rectangle B of candidate skyline points.
3. Add the closest point to the heap and iteratively examines the top entry.
4. Add skyline points if they are not dominated by current skyline and updates Extracted and B.
5. Add unvisited Voronoi neighbors to Visited and Heap if conditions are met.
6. Return the skyline points when the heap becomes empty.

For example:



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	$(\mathbf{p_5 : 34})$, $(p_4 : 38)$, $(p_8 : 44)$, $(p_7 : 46)$, $(p_9 : 49)$, $(p_{10} : 49)$, $(p_{11} : 63)$	$\{p_1, p_3, \mathbf{p_6}\}$
...	...	$\{p_1, p_3, p_6, \mathbf{p_5}, \mathbf{p_4}, \mathbf{p_2}\}$

Algorithm VS² (set Q)

compute the convex hull $CH(Q)$;
set $S(Q) = \{\}$;
Heap $H = \{(NN(q_1), mindist(NN(q_1), CH_v(Q)))\}$;
set $Visited = \{NN(q_1)\}$; set $Extracted = \{\}$;
box $B = MBR(SR(NN(q_1), Q))$;
while H is not empty
  (p, key) = first entry of H;
  if $p \in Extracted$
    remove (p, key) from H;
    if p is inside $CH(Q)$ or
      p is not dominated by $S(Q)$
      add p to $S(Q)$;
      $B = B \cap MBR(SR(p', Q))$;
  else
    add p to $Extracted$;
    if $S(Q) = \varnothing$ or a Voronoi neighbor of p is in $S(Q)$
      for each Voronoi neighbor of p such as p'
        if $p' \in Visited$, discard p';
        if p' is inside B or $VC(p')$ intersects with B
          add p' to $Visited$;
          add $(p', mindist(p', CH_v(Q)))$ to H;
return $S(Q)$;

4.2 Continuous Query. What happens when **query points** change their locations?

→ Recompute spatial skyline points.

→ **Problem**: R-tree-based algorithms (B2S2 and BBS) are expensive because the entire R-tree must be traversed per each update.

→ **Solution**: update the set of previously found skyline points by examining only those data points which may change the spatial skyline.

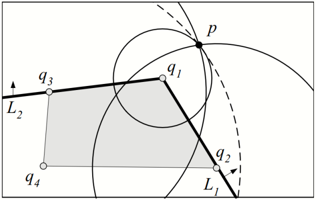
→ **Idea**:

- Find the query points which may change the dominance of a data point outside CH(Q). (Lemma 4)
- Choose the data points that must be examined when the location of q changes. (Lemma 5)

Assume that:

- $CH_v^+(Q)$ and $CH_v^-(Q)$: sets of the vertices on the *closer* and *farther* hulls of the convex hull CH(Q) to p, respectively.
- **Lemma 4**: Given a data point p and a query set Q, the dominance of p only depends on the query points in $CH_v^+(Q)$.
- **Lemma 5**: The locus of data points whose dominance depends on $q \in CH_v(Q)$ is the visible region of q.

For example:



$\rightarrow CH_v^+(Q) = \{q_1, q_2, q_3\}$ and $CH_v^-(Q) = \{q_4\}$.

4.2.1 Voronoi-based Continuous SSQ (VCS^2)

Assume that:

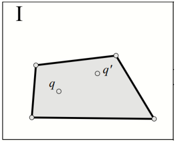
- q' :new location of q
- Q' $= Q \cup \{q'\} - \{q\}$ new set of query locations.

Intuition: VCS2 uses the visible regions of q and q' to partition the space into regions. $\rightarrow$ avoids excessive unnecessary dominance checks.

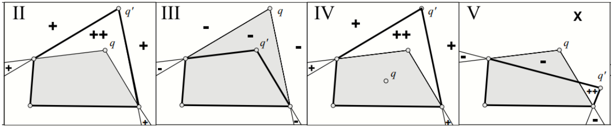
Steps:

1. Compute the convex hull of the latest query set CH(Q').
2. Compares CH(Q') with the old convex hull CH(Q).
3. Depending on how CH(Q') differs from CH(Q), VCS^2 decides to:
   - Case 1: traverse only specific portions of the graph and update the old skyline S(Q).
   - Case 2: rerun VS^2 and generate a new one.
4. Retrieve points in the candidate area and update Skyline S(Q) for new points.
5. Removes inverted points from Skyline S(Q).

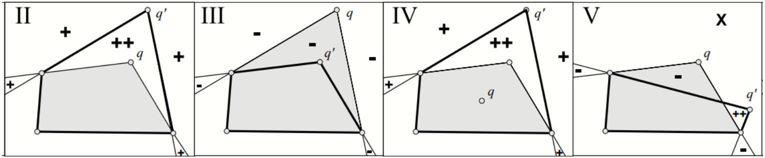
Case 1: Situations where VCS2 updates the skyline based on the change in CH(Q’).



Pattern I: CH(Q) and CH(q’) are identical, the skyline does not change and no graph traversal is required. (Theorem 2)

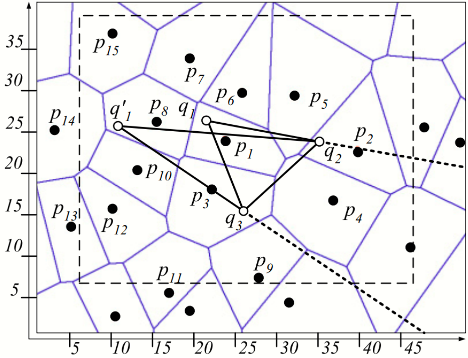


Patterns II-V: The intersection region of CH(Q) and CH(Q’) skyline points to both Q and Q’, no traversal needed.



- **The region "++"**: points inside CH(Q') and outside CH(Q). Any point in this region is a *skyline point* with respect to Q' → add to the skyline. (Theorem 1)
- **The region "+"**: points whose dominator regions have become smaller → might be skyline points and must *be examined*.
- **The region "-"**: points whose dominator regions have been expanded → *delete* from the old skyline.
- **The regions "x"**: must *be examined* as their dominator regions have changed.
- **Unlabeled white region**: points outside of the visible regions of both q and q' → remain the same. (Lemma 5)

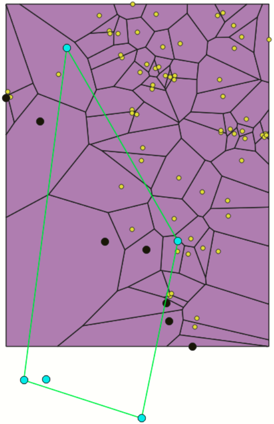
For example:



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	$(p_7 : 50)$, $(p_9 : 51)$, $(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$	$\{p_1, p_3, \cancel{\mathbf{p_6}}, p_5, p_4, p_2, p_8, p_{10}\}$
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	$(p_{12} : 52)$, $(p_{11} : 60)$, $(p_{14} : 64)$, $(p_{15} : 66)$, $(p_{13} : 66)$	$\{p_1, p_3, p_5, p_4, p_2, p_8, p_{10}\}$

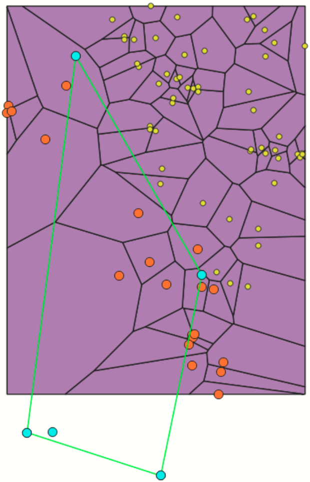
1.Branch-and-Bound Spatial Skyline Algorithm (B^2S^2)

```
user_id,latitude,longitude,timestamp
1,21.02162,105.845034,2023-06-21 21:10:59
2,21.007824,105.843126,2023-06-21 21:10:59
3,21.006242,105.848367,2023-06-21 21:10:59
4,21.013605,105.849983,2023-06-21 21:10:59
5,21.007844,105.844115,2023-06-21 21:10:59
```



2.Voronoi-based Spatial Skyline Algorithm (VS^2)

```
user_id,latitude,longitude,timestamp
1,21.02162,105.845034,2023-06-21 21:10:59
2,21.007824,105.843126,2023-06-21 21:10:59
3,21.006242,105.848367,2023-06-21 21:10:59
4,21.013605,105.849983,2023-06-21 21:10:59
5,21.007844,105.844115,2023-06-21 21:10:59
```



3.Voronoi-based Continuous SSQ (VCS^2)

```
user_id,latitude,longitude,timestamp
1,21.02162,105.845034,2023-06-21 21:10:59
1,21.021298,105.84443,2023-06-21 21:13:59
1,21.020935,105.843744,2023-06-21 21:16:59
1,21.021083,105.843175,2023-06-21 21:19:59
1,21.021877,105.842821,2023-06-21 21:22:59
1,21.021161,105.843527,2023-06-21 21:25:59
1,21.021968,105.843558,2023-06-21 21:28:59
1,21.021227,105.842993,2023-06-21 21:31:59
1,21.020602,105.842923,2023-06-21 21:34:59
1,21.019704,105.842424,2023-06-21 21:37:59
2,21.007824,105.843126,2023-06-21 21:10:59
2,21.007208,105.84374,2023-06-21 21:13:59
2,21.006696,105.84324,2023-06-21 21:16:59
2,21.007042,105.843824,2023-06-21 21:19:59
2,21.007453,105.844322,2023-06-21 21:22:59
2,21.007851,105.844522,2023-06-21 21:25:59
2,21.008116,105.844227,2023-06-21 21:28:59
2,21.008628,105.844438,2023-06-21 21:31:59
2,21.008323,105.844003,2023-06-21 21:34:59
2,21.009162,105.844563,2023-06-21 21:37:59
3,21.006242,105.848367,2023-06-21 21:10:59
3,21.006644,105.848513,2023-06-21 21:13:59
3,21.007319,105.848829,2023-06-21 21:16:59
3,21.007862,105.849165,2023-06-21 21:19:59
```

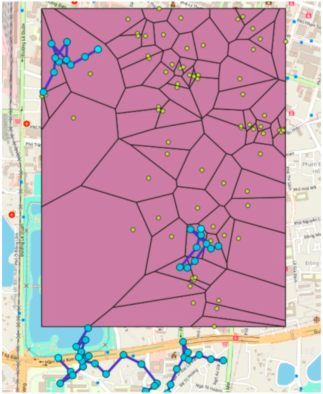


Figure: Movement visualization

here