Skylines

Yufei Tao

ITEE
University of Queensland

Today we will discuss problems closely related to the topic of multi-criteria optimization, where one aims to identify objects that strike a good balance—often optimal in some sense—among multiple dimensions.

Our focus will be placed on how to find such objects efficiently. In particular, we will consider points that are indexed by an R-tree, and study algorithms that can leverage the R-tree to reduce cost effectively.

Monotonically Increasing Functions

Let p be a d-dimensional point in $\mathbb{R}^d$. Let $f : \mathbb{R}^d \to \mathbb{R}$ a function that calculates a score $f(p)$ for p. We say that f is monotonically increasing if the score increases when any coordinate of p increases.

Top-1 Search

Let us first look at one of the most natural operations in multi-criteria optimization.

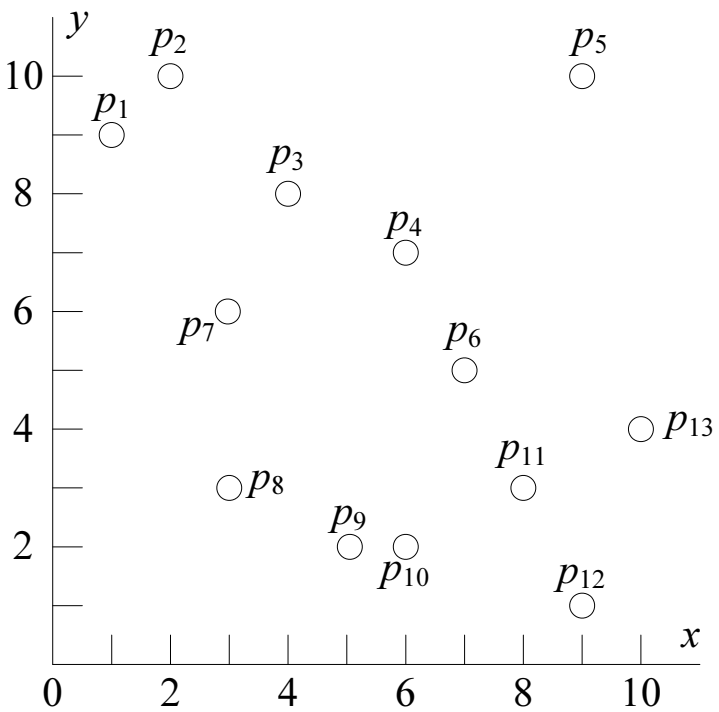
Let P be a set of d-dimensional points in $\mathbb{R}^d$. Given a monotonically increasing function f, a top-1 query finds the point in P that has the smallest score.

In the special case where multiple points have the smallest score, all of them should be returned.

- f is often referred to as the preference function of the query.
- Without loss of generality, let us assume that the coordinates of all the points are positive (think: why is this a fair assumption?).

Example

If $f(x, y) = x + y$, the top-1 point is p_8.



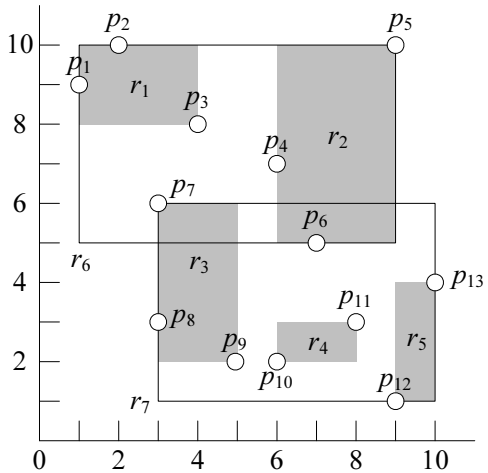
Applications of Top-1 Search

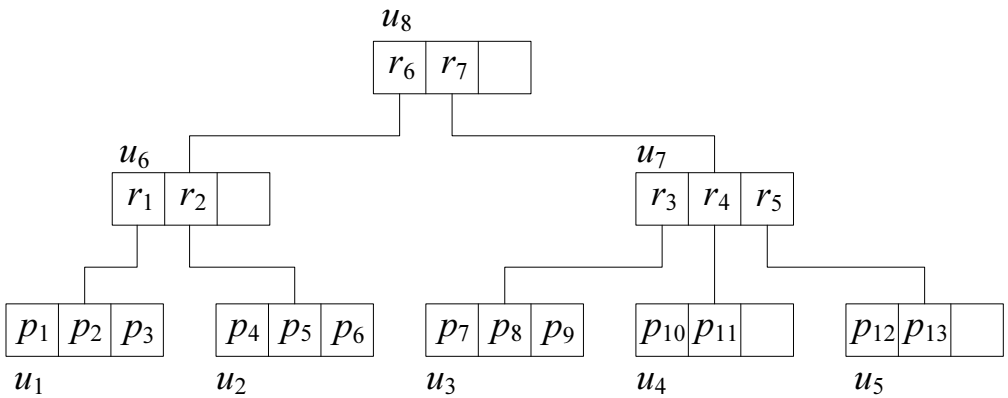
- "Find the hotel that minimizes *price* + *distance* (to the beach)."
- "Find the second-handed car that minimizes *price* + $0.1 \cdot$ *mileage*."
- "Find the interviewee that maximizes $0.4 \cdot$ *education* + $0.4 \cdot$ *experience* + $0.2 \cdot$ *personality*."
- ...

Top-1 Search is NN Search in Disguise!

Assuming that the dataset P is indexed by an R-tree, we can answer a top-1 query by directly applying a nearest neighbor algorithm discussed before. Specifically, the top-1 result is precisely the NN of the origin of the data space (i.e., the query point) according to the "distance function" f.

Think: What is the mindist of an MBR to the query point?

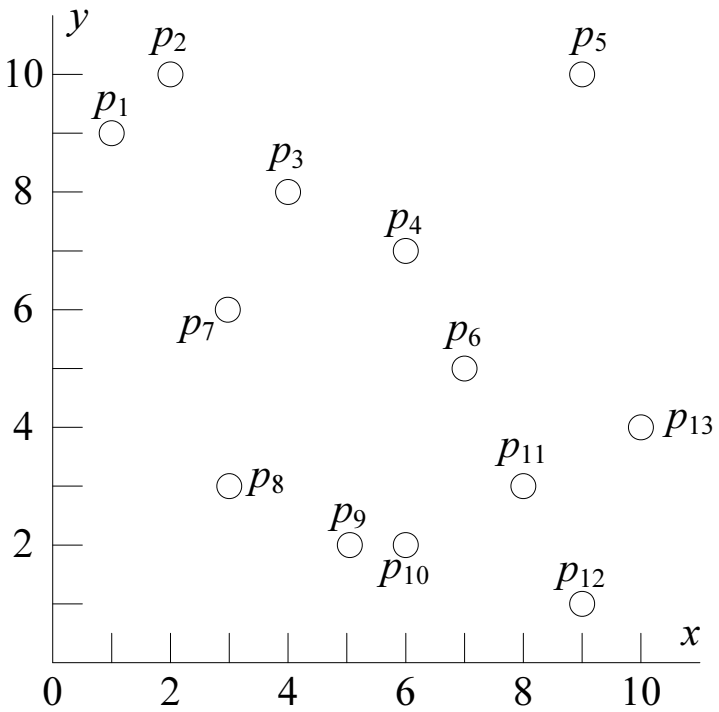




Drawback of Top-1 Search—The Preference Function

In general, it is difficult to decide which preference function f to use.

For example, assume that the x-dimension corresponds to the *price* of a hotel and the y-dimension to its *distance* to the beach. Why is $f(x, y) = x + y$ a good function to use? Why not $2x + y$, or something more complex like $\sqrt{x} + y^2$?

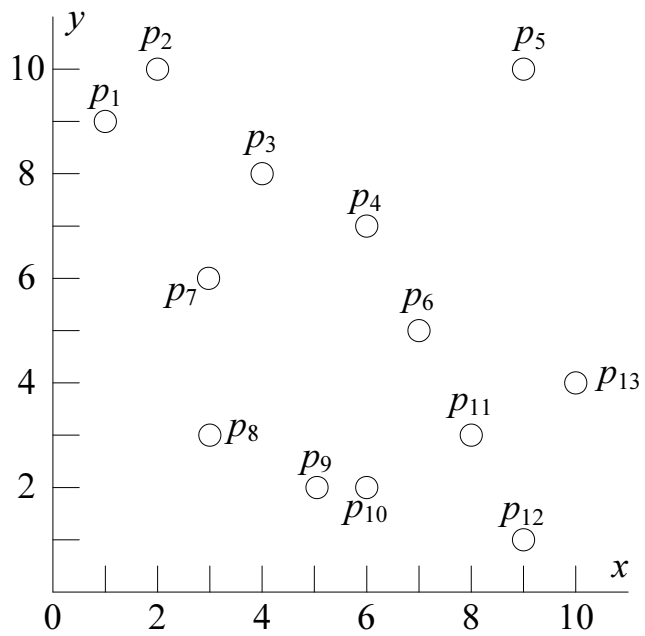


To remedy the drawback, we need to do away with preference functions. But then we fall into a dilemma—how do we return "top-1" points?

The skyline operator achieves the purpose with an interesting idea. Instead of returning only 1 object, how about returning a small set of objects that is guaranteed to cover the result of any top-1 query (i.e., regardless of the preference function).

Dominance

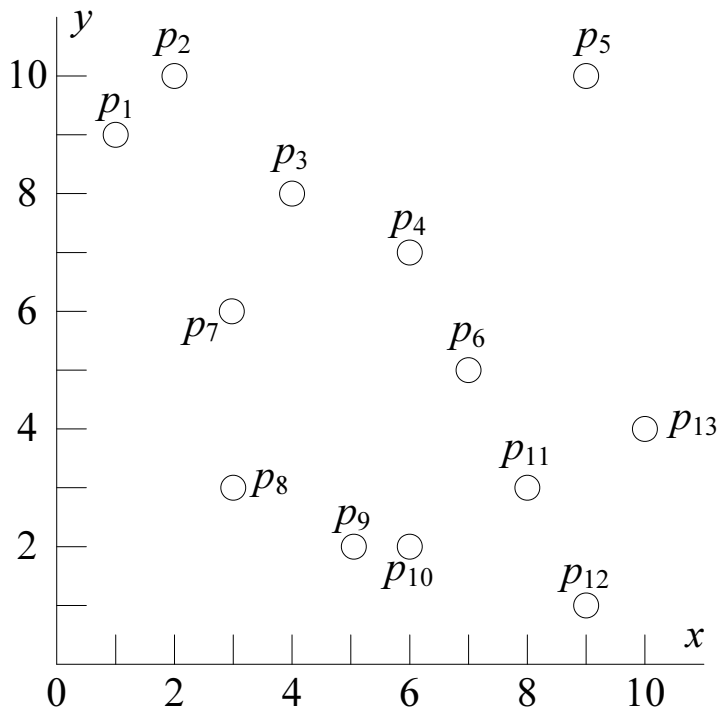
A point p_1 dominates p_2 if the coordinate of p_1 is smaller than or equal to p_2 in all dimensions, and strictly smaller in one dimension.



For example, p_8 dominates p_6, which dominates p_5. It is clear that dominance is transitive.

Skyline Search

Let P be a set of d-dimensional points in $\mathbb{R}^d$. The skyline of P is the set of points in P that are not dominated by others.



The skyline is $\{p_1, p_8, p_9, p_{12}\}$.

Now we can talk about how the skyline remedies the aforementioned drawback of top-1 search.

Theorem: For any monotonically increasing function, a top-1 point is definitely in the skyline.

Proof: Suppose that this is not true, and that there exists a function f for which a top-1 point p is not in the skyline. This means that p is dominated by at least another point p'. The monotonicity of f tells us that $f(p') < f(p)$, which contradicts the fact that p is a top-1 point. □

Recall that our goal is to return a small set of points that is guaranteed to cover the top-1 result of any preference function. The next theorem states that the skyline is indeed the smallest subset.

Theorem: Every point in the skyline is definitely a top-1 point of some monotonically increasing function.

The proof is not required, but the instructor will outline the idea behind.

Next we will proceed to discuss how to find the skyline efficiently using an R-tree. We will consider two scenarios:

- What if NN search is the only available operator on the R-tree (plus, of course the "standard" insertion/deletion operators)?
  - Think: Why could this happen?
  - We will discuss how to use NN search as a black box to find the skyline.
- What if we are allowed to design an algorithm of our own on the R-tree?
  - We will come up with another algorithm that solves the problem optimally.

First Algorithm: Using NN Search as a Black Box

Lemma: Let p be the (Euclidean) NN of the origin of the data space, among all the points in the dataset P. Then, p must be in the skyline of P.

Proof: Suppose that this is not true. Then, there exists another point p' that dominates p. This, however, means that p' must be closer to the origin than p, creating a contradiction. □

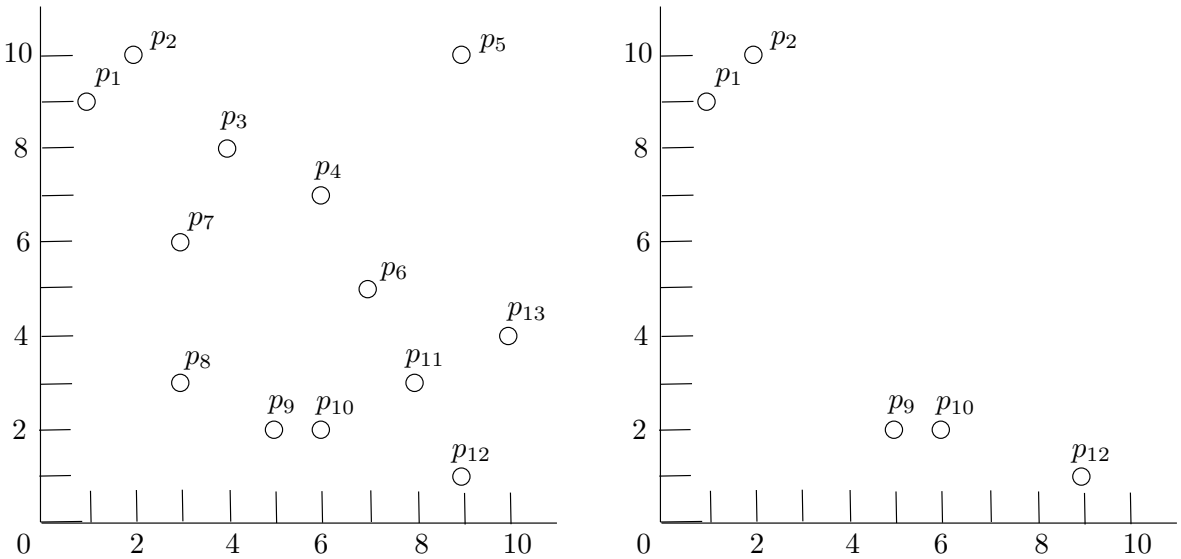
NN Search as a Black Box

Motivated by the previous observation, we can find the skyline by repeating the next two operations until the R-tree is empty:

- Find the NN p of the origin of the data space. Include p into the skyline.
- Delete p from the tree, as well as all the points that are dominated by p.

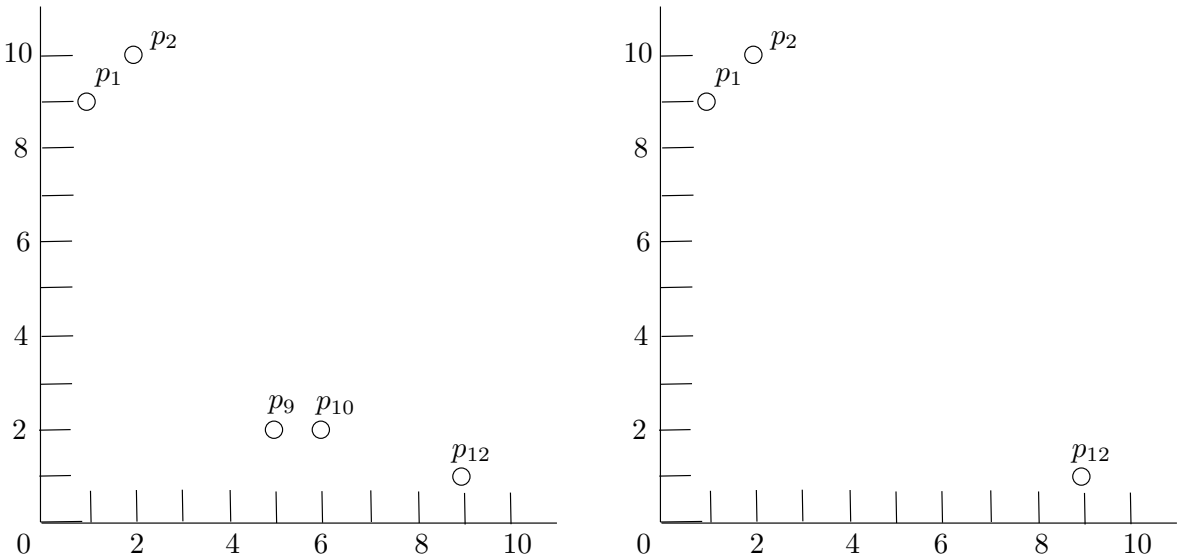
Think: Why is the algorithm correct?

Example



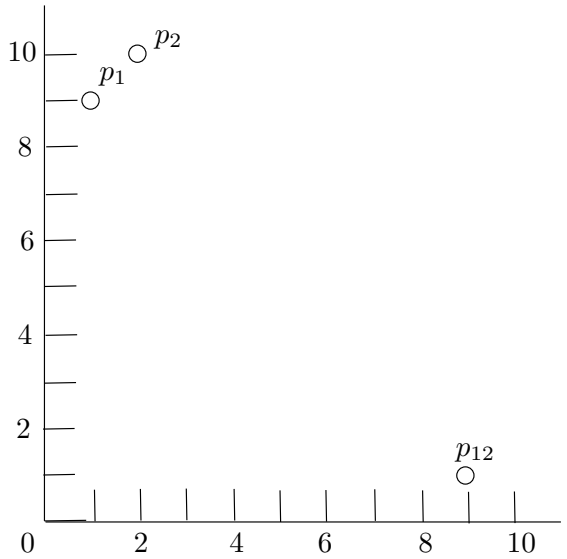
The first NN query finds p_8. The figure on the right shows the dataset after the corresponding deletions.

Example



The second NN query finds p_9. The figure on the right shows the dataset after the corresponding deletions.

Example



The third NN query finds p_1 and p_{12}. The dataset becomes empty after corresponding deletions.

Final skyline point $= \{p_8, p_9, p_1, p_{12}\}$.

This previous algorithm must access all the points in the dataset (why?). The next algorithm remedies the defect by accessing (typically) only a part of the dataset.

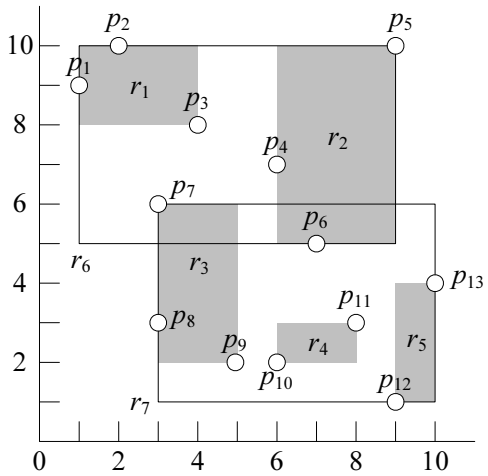
Second Algorithm: Branch and Bound Skyline (BBS)

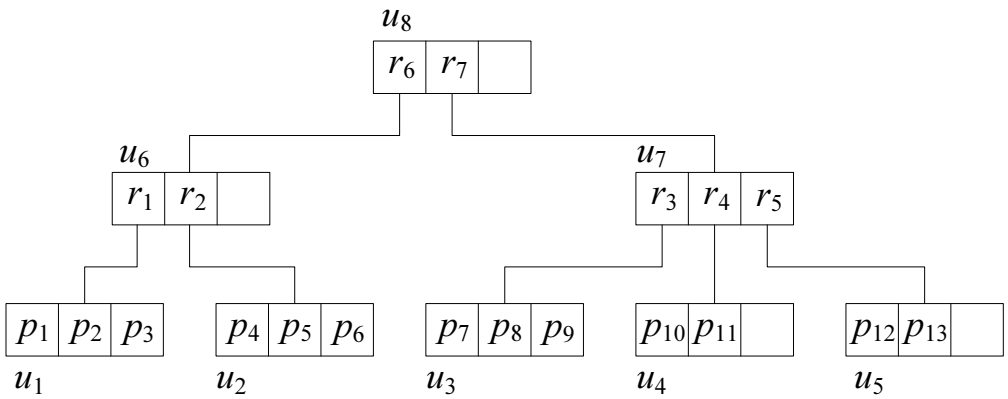
The BBS algorithm can be thought of a variation of the BF algorithm for NN search. It accesses the nodes of the R-tree in ascending order of their mindists from the origin. Different from NN search, however, once an MBR in its entirety is dominated by a skyline point already found, it can be pruned.

Next we will illustrate the algorithm through an example.

BBS

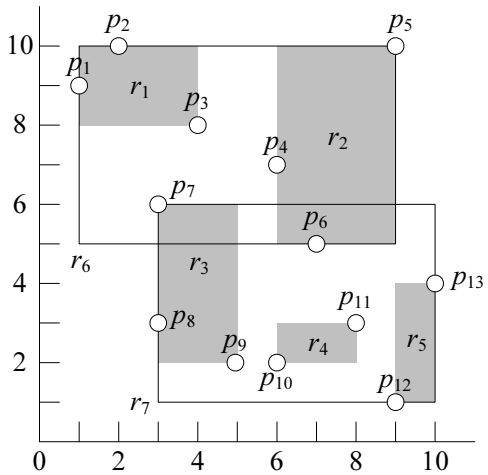
First, we access the root, and put the MBRs there in a sorted list H, namely, $H = \{(r_7, \sqrt{10}), (r_6, \sqrt{26})\}$. Note that the sorting key of its MBR is its mindist to the origin.

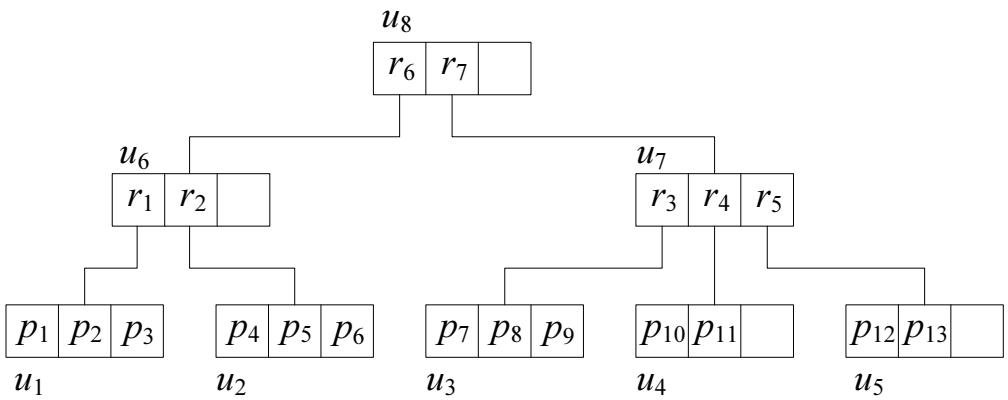




BBS

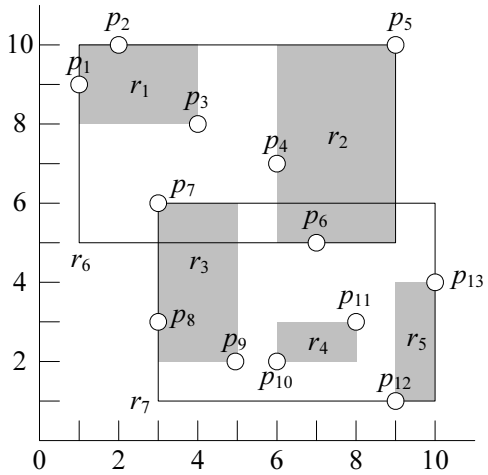
The algorithm then visits node u_7, after which $H = \{(r_3, \sqrt{13}), (r_6, \sqrt{26}), (r_4, \sqrt{40}), (r_5, \sqrt{82})\}$.

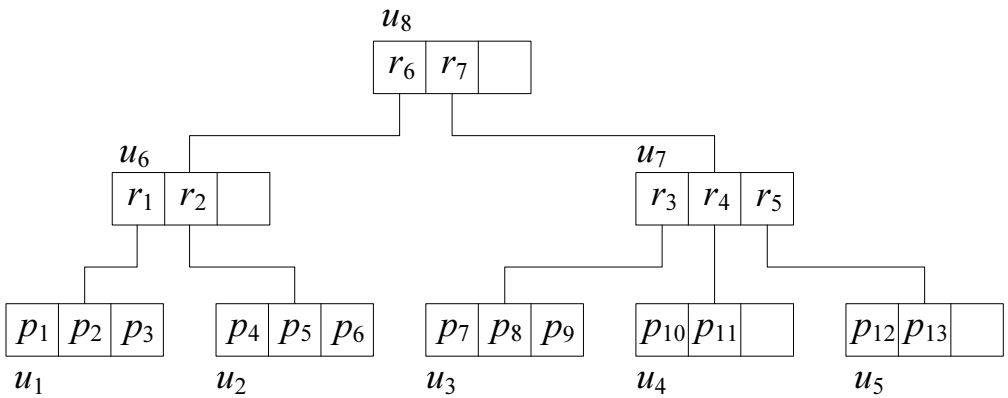




BBS

We now visit u_3 which is a leaf node. Insert all the points therein into H, which becomes:
$H = \{(p_8, \sqrt{18}), (r_6, \sqrt{26}), (p_9, \sqrt{29}), (r_4, \sqrt{40}), (p_7, \sqrt{45}), (r_5, \sqrt{82})\}$.

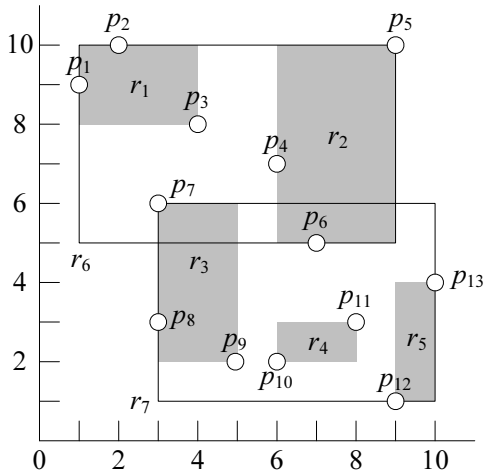


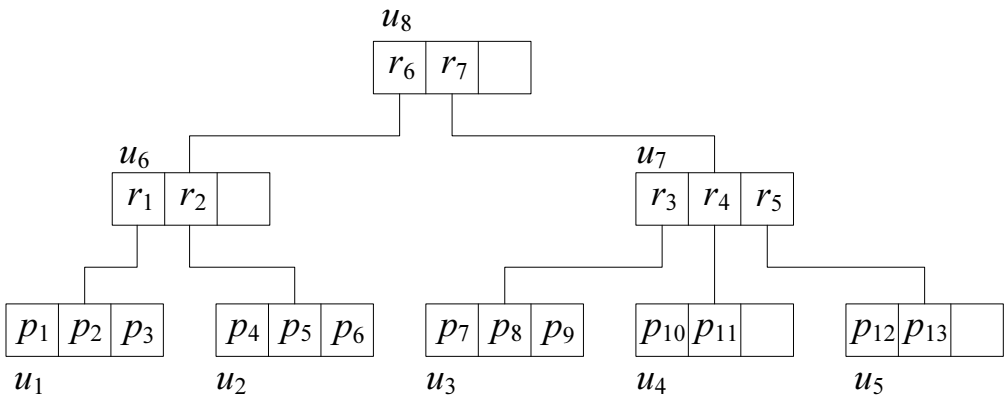


BBS

Now we remove p_8 from H. This is the first skyline point found.

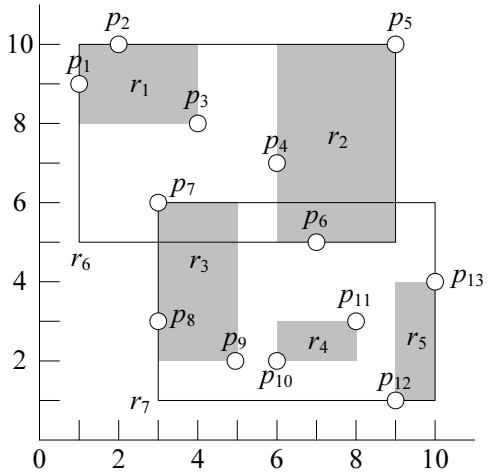
H is now $\{(r_6, \sqrt{26}), (p_9, \sqrt{29}), (r_4, \sqrt{40}), (p_7, \sqrt{45}), (r_5, \sqrt{82})\}$.

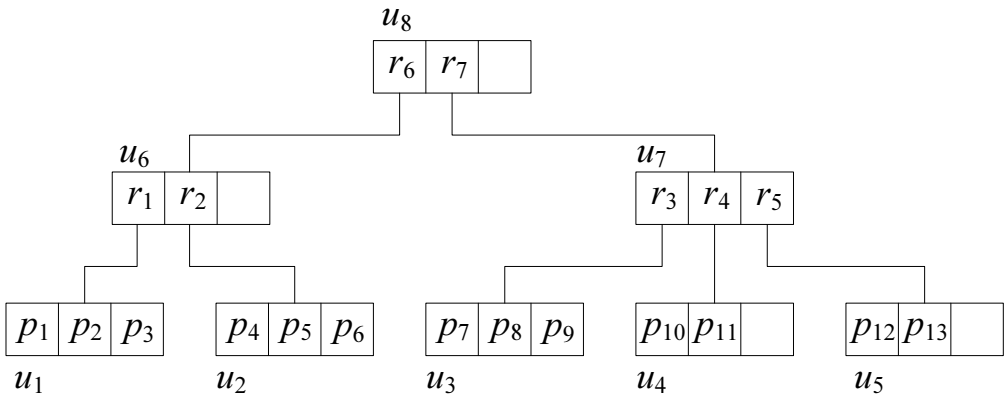




BBS

Next, access u_6, and update H to
$\{(p_9, \sqrt{29}), (r_4, \sqrt{40}), (p_7, \sqrt{45}), (r_2, \sqrt{61}), (r_1, \sqrt{65}), (r_5, \sqrt{82})\}$.

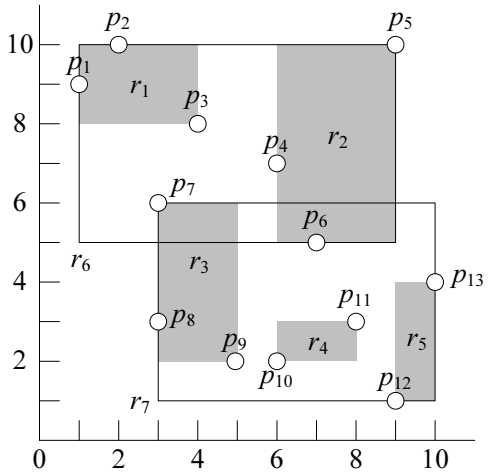


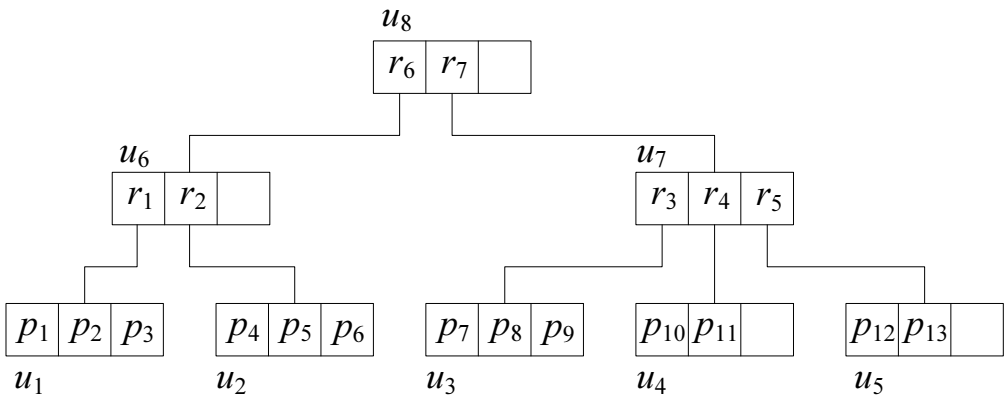


BBS

Remove from H the next point p_9, which is the second skyline point found.

H is now $\{(r_4, \sqrt{40}), (p_7, \sqrt{45}), (r_2, \sqrt{61}), (r_1, \sqrt{65}), (r_5, \sqrt{82})\}$.



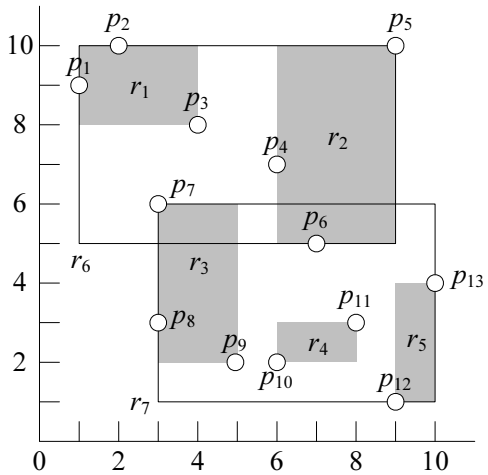


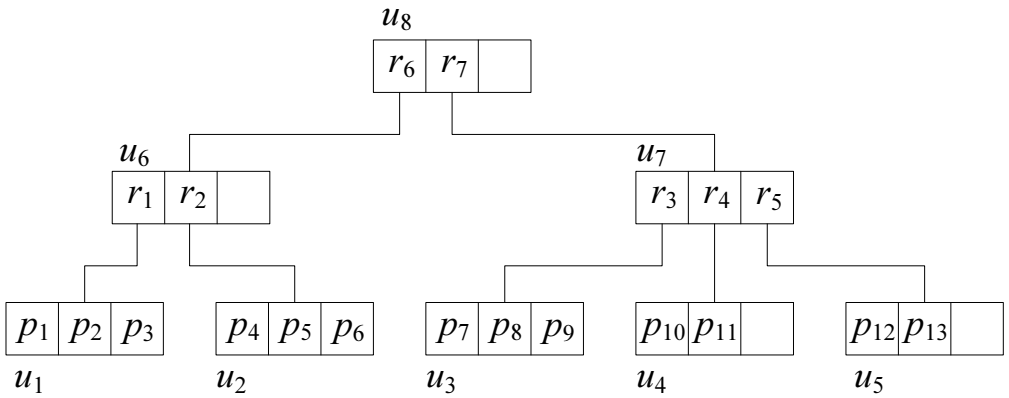
BBS

The next MBR r_4 in H can be directly discarded, because its lower-left corner is dominated by a skyline point p_8. This implies that no points in r_4 can possibly be in the skyline.

Following the same reasoning, both of the next point p_7 and the following MBR r_2 are also discarded.

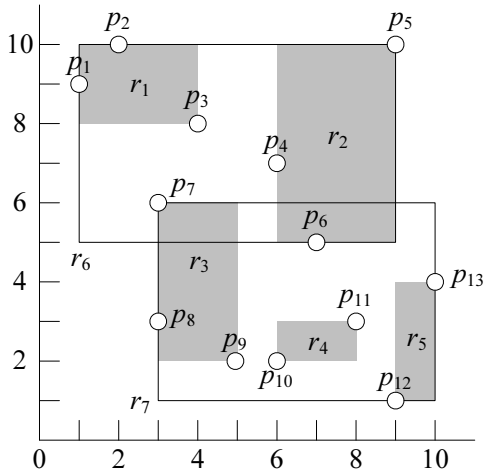
Currently $H = \{(r_1, \sqrt{65}), (r_5, \sqrt{82})\}$.
Skyline points found so far $= \{p_8, p_9\}$.

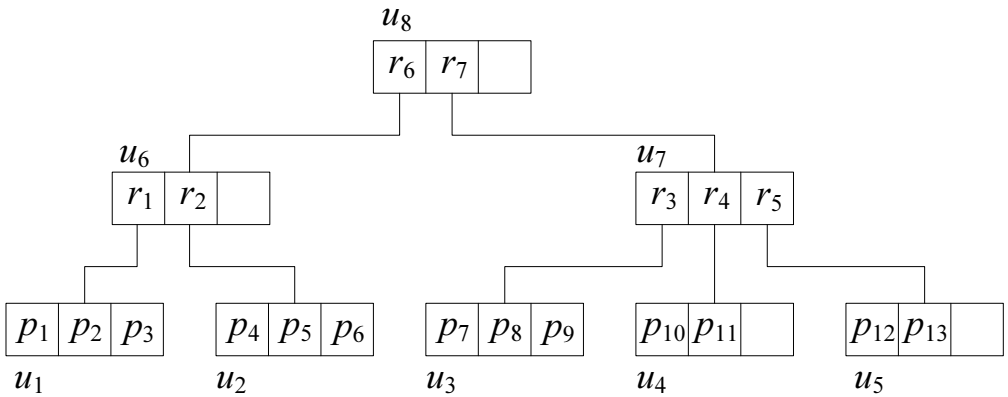




BBS

Next, access the leaf node u_1, after which H becomes $H = \{(p_3, \sqrt{80}), (p_1, \sqrt{82}), (r_5, \sqrt{82}), (p_2, \sqrt{104})\}$.

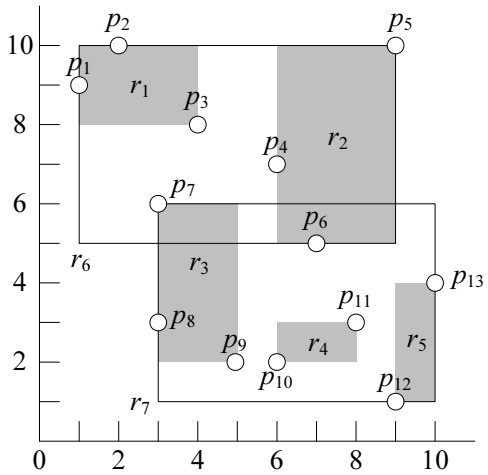


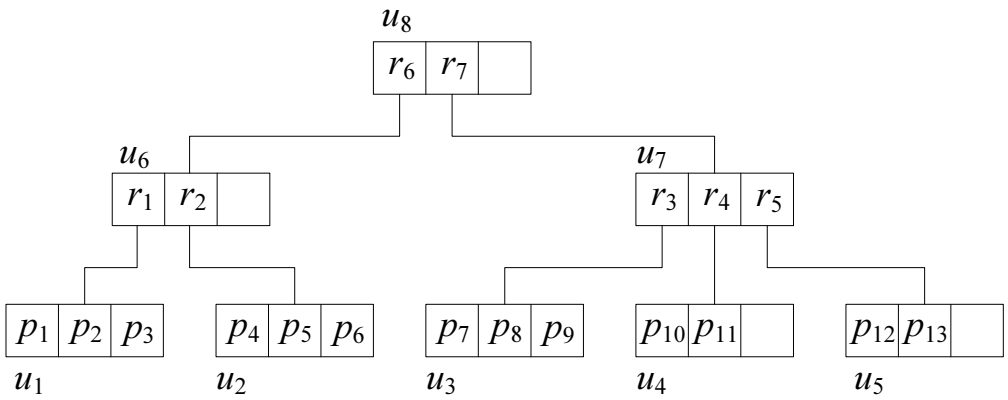


The next point p_3 is discarded. Then, we come to p_1, which is added to the skyline, which is now $\{p_1, p_8, p_9\}$.

BBS

Now $H = \{(r_5, \sqrt{82}), (p_2, \sqrt{104})\}$. The rest of the algorithm proceeds in the same manner. We discover the 4th skyline point p_{12} in r_5.





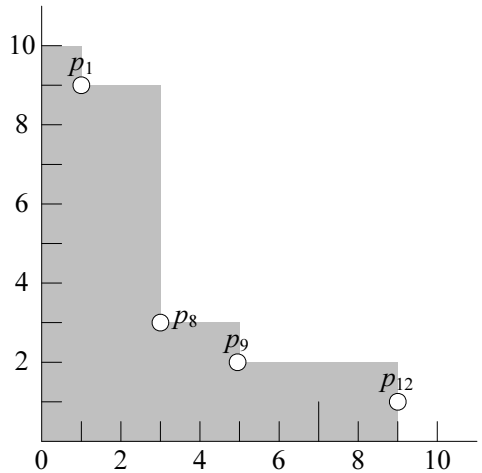
Pseudocode of BBS

algorithm BBS

1. insert the MBR of the root into a sorted list H
   /* entries in H are sorted by their mindists to the origin */
2. $SKY = \emptyset$ /* current result */
3. **while** H is not empty **do**
4. remove the the first entry e from H
5. **if** e is the MBR of a node u
6. **if** no point in SKY dominates the lower-left corner e **then**
7. visit u and insert all MBRs/points there into H
8. **else** /* e is a point */
9. **if** no point in SKY dominates e **then**
10. add e to SKY

Optimality of BBS

BBS is optimal, i.e., it accesses the least number of nodes among all algorithms that correctly finds the skyline using the same R-tree (without modifying the structure). To prove this, let us define the search region as the union of the points in $\mathbb{R}^d$ that are not dominated by any skyline point. For example, in our previous example, the search region is the shaded area below:



It is easy to see that any correct algorithm must access all the nodes whose MBRs intersect the search region.

Optimality of BBS

We can show that BBS accesses only the nodes whose MBRs intersect the search region. Assume, for contradiction, that the algorithm needs to visit a node u whose MBR r is disjoint with the region.

- It follows that a skyline point p dominates the lower-left corner of r.
- Thus, p has a smaller mindist (a.k.a. distance) to the origin than r.
- The sequence of points/nodes processed by BBS is in ascending order of mindist to the origin.
    - Prove this by yourself.
- Hence, p is processed before u.
- However, once p is processed, it enters the skyline, making it impossible for the algorithm to visit u.