Multidimensional Divide and Conquer 1 — Skylines

Yufei Tao

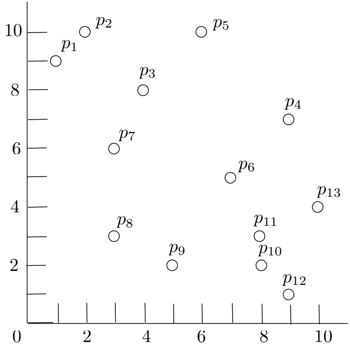
ITEE
University of Queensland

The next few lectures will be dedicated to an important technique: divide and conquer. You may have encountered the technique in an earlier algorithm course, but we will study it from a new perspective: computational geometry. We will see how it can be deployed to settle several fundamental problems on multidimensional data with attractive performance guarantees.

Our first application is the skyline problem, which we have already learned how to solve "optimally" with an R-tree. But the notion of optimality is confined to the assumption that all algorithms must use the same R-tree. We will depart from the assumption today, and seek better algorithms that do not use the R-tree.

Review: Skylines

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. 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 $\{p_1, p_8, p_9, p_{12}\}$.

One way of solving the problem is to first build an R-tree on the dataset, and then apply the BBS algorithm discussed before. While this approach would offer reasonable efficiency in practice, from a theoretical point of view, it is slow: in the worst case, its time complexity is $O(n^2)$, where n is the number points in P. This means when you are "unlucky" to receive a "hard" input set, your program will have to entail an excessive amount of cost.

We will see how to solve the problem in $O(n \log^{d-1} n)$ time (recall that d is the dimensionality).

Remark 1: Here, we assume that the input set is not already indexed by a data structure like an R-tree (think: when can this happen in practice?). As a result, any algorithm must incur $\Omega(n)$ time just to scan through the dataset once (think: why?).

Remark 2: This problem can actually be solved in $O(n \log^{d-2} n)$ time, as we will explore in the exercises.

Divide and Conquer: Overview

This methodology applies the ideas below:

1. Divide the input into two (or more) subsets.
2. Solve the problem separately on each subset.
3. Merge the solutions of the two subsets into the solution for the original input.

Think: Have you learned any divide-and-conquer algorithms before?

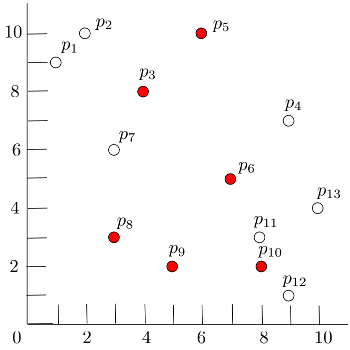
Dominance Screening

We will first study a different problem:

Let P and Q be sets of d-dimensional points in $\mathbb{R}^d$. The objective of the dominance screening problem is to report all the points $q \in Q$ such that q is not dominated by any points in P.

Dominance Screening

Suppose that P is the set of white points, and Q the set of red points.

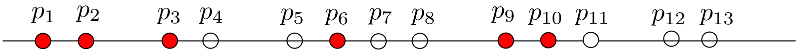


The answer is $\{p_8, p_9, p_6, p_{10}\}$. Note, in particular, that p_{10} is in the answer—it is not dominated by any white point (i.e., p_9 does not count).

Dominance Screening

Apparently, this is not the final "skyline problem" we aim to solve. However, as we will see, the former is at the core of using divide-and-conquer to attack the latter problem. This "change of target problem" is not uncommon in computer science, especially in the application of divide and conquer. We will come back to this issue at the end of the lecture.

1D Dominance Screening



Let us first consider the dominance screening problem in 1D space. The above is an example where P (Q) is the set of white (or red, resp.) points. The answer is $\{p_1, p_2, p_3\}$.

1D Dominance Screening

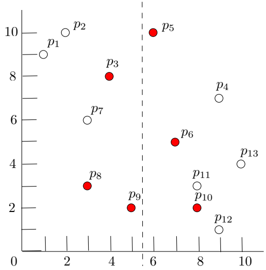
The previous slide implies the following simple algorithm for 1D dominance screening:

1. Identify the leftmost (i.e., smallest in value) point p_{min} in P.
2. Report every point $q \in Q$ such that $q \leq p_{min}$.

Clearly this can be done in $O(n)$ time where $n = |P| + |Q|$.

2D Dominance Screening

Now, let us move to 2D space. This time, we will apply divide and conquer. First, divide the input set into two halves by x-coordinate:

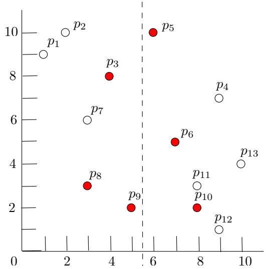


Let P_1 (Q_1) be the set of white (or red, resp.) points on the left half. That is, $P_1 = \{p_1, p_2, p_7\}$ and $Q_1 = \{p_3, p_8, p_9\}$.

Define P_2 and Q_2 analogously with respect to the right half.

2D Dominance Screening

Now we have two instances of the dominance screen problem. The first one consists of P_1 and Q_1, while the other one P_2, Q_2.



Solve (i.e., conquer) each instance individually:

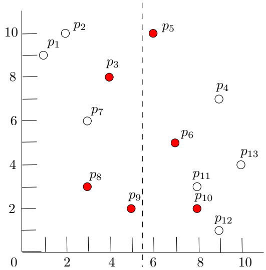
- Left instance: Report p_8, p_9
- Right instance: Report p_5, p_6, p_{10}

2D Dominance Screening

From the previous slide, we have:

- Left instance: Report p_8, p_9
- Right instance: Report p_5, p_6, p_{10}

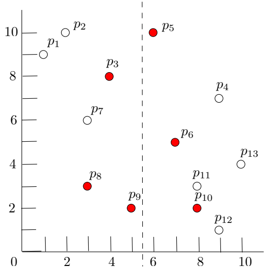
Next, we must merge the two solutions to obtain the final answer on the original dataset.



Observation: The answer from the left instance is definitely in the final answer. Think: why?

2D Dominance Screening

How about the answer of the right instance: p_5, p_6, p_{10}?

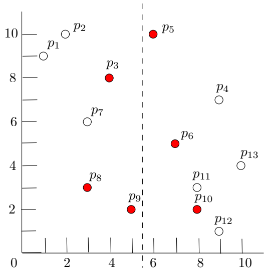


p_5 is not in the final answer because it is dominated by a white point p_7 on the left of the line.

Observation: Let q be a point in the answer of the right instance. It is in the final answer if and only if no white point from the left instance dominates q.

2D Dominance Screening

Motivated by the previous observation, we determine the final answer by reducing the problem to 1D dominance screening.

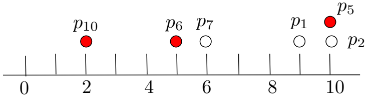


Let A_{right} be the answer from the right instance. Construct a 1D dominance screening problem with input sets P', Q' as follows:

- P' = the projections of P_1 onto the y-axis.
- Q' = the projections of A_{right} onto the y-axis.

2D Dominance Screening

The figure below shows the points in the 1D dominance screening problem:



$P' = \{p_7, p_1, p_2\}$, and $Q' = \{p_{10}, p_6, p_5\}$.

Solving this 1D problem—which we already know how to—gives $\{p_{10}, p_6\}$, which is precisely the set of points in A_{right} that should be added to the final answer.

2D Dominance Screening

The previous discussion points to the following divide-and-conquer algorithm (for solving 2D dominance screening with input sets P and Q):

1. Divide the points in P and Q into two equal halves by x-coordinate. In this way, we obtain two instances of the 2D dominance screening problem: (i) left instance P_1, Q_1, and (ii) right instance P_2, Q_2.
2. Solve the left and right instances, recursively. Let A_{left} and A_{right} be their answers, respectively.
3. Obtain a 1D dominance screening problem P', Q' where P' (Q') is the projection of P_1 (A_{right}, resp.) onto the y-axis. Solve this instance to obtain its answer A'.
4. Return the points in A_{left} and those corresponding to the ones in A'.

2D Dominance Screening

Let us now analyze the running time of our algorithm. Let $f(n)$ be the time on $n = |P| + |Q|$ points. We have:

$$f(n) \quad \leq \quad 2 \cdot f(n/2) + O(n)$$

Specifically, the two terms $f(n/2)$ capture the cost of solving the left and right instances, respectively. The term $O(n)$ captures the cost of Steps 1, 2, and 3—recall that the 1D problem can be solved in $O(n)$ time.

For $n \leq 2$, clearly $f(n) = O(1)$.

Solving the recurrence gives: $f(n) = O(n \log n)$.

Dominance Screening in d-dimensional Space

We now have proved that the problem can be solved in $O(n \log n)$ time in 2D space.

Our algorithm can be immediately generalized to any dimensionality d. Recall that we solved the 2D case by reducing it to 1D. In general, we solve a d-dimensional problem by reducing it to $d - 1$ dimensions. As we will see, divide-and-conquer allows us to do so very easily.

Dominance Screening in d-dimensional Space

A divide-and-conquer algorithm for solving d-dimensional dominance screening with input sets P and Q:

1. Divide the points in P and Q into two equal halves by the first coordinate. In this way, we obtain two instances of the d-dimensional dominance screening problem: (i) left instance P_1, Q_1, and (ii) right instance P_2, Q_2.
2. Solve the left and right instances, recursively. Let A_{left} and A_{right} be their answers, respectively.
3. Obtain a $(d-1)$-dimensional dominance screening problem P', Q' where P' (Q') is the projection of P_1 (A_{right}, resp.) onto dimensions $2, 3, ..., d$. Solve this instance to obtain its answer A'.
4. Return the points in A_{left} and those corresponding to the ones in A'.

Dominance Screening in d-dimensional Space

Let us now analyze the running time of our algorithm. Let $f(n)$ be the time on $n = |P| + |Q|$ points. We have:

$$f(n) \leq 2 \cdot f(n/2) + g(n)$$

Specifically, the two terms $f(n/2)$ capture the cost of solving the left and right instances, respectively. The term $g(n)$ is the cost of solving the problem in the $d - 1$ dimensional space.

For $n \leq 2$, clearly $f(n) = O(1)$.

Solving the recurrence gives:

- When $d = 3$: $f(n) = O(n \log^2 n)$.
- When $d = 4$: $f(n) = O(n \log^3 n)$.
- ...
- In general, $f(n) = O(n \log^{d-1} n)$.

Skyline

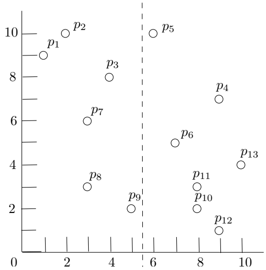
We now conclude that in d-dimensional space, dominance screening can be solved in $O(n \log^{d-1} n)$ time. Equipped with this result, we will now attack the skyline problem.

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

As we will see, this is another beautiful application of divide and conquer. The crux is to use dominance screening to "merge" the solutions to two subproblems.

Skyline

First, divide the input set into two halves by x-coordinate:

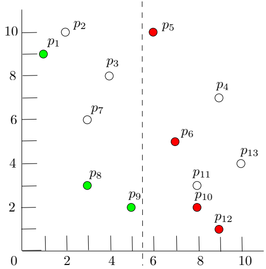


Let P_1 be the set of points on the left half. That is, $P_1 = \{p_1, p_2, p_3, p_7, p_8, p_9\}$.

Define P_2 analogously with respect to the right half.

Skyline

Find the skylines of P_1 and P_2 separately.

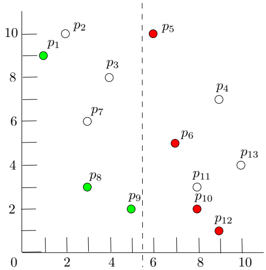


On the left side, the answer is $A_{left} = \{p_1, p_8, p_9\}$. On the right, it is $A_{right} = \{p_5, p_6, p_{10}, p_{12}\}$.

Observation: The points in A_{left} must be in the final skyline.

Skyline

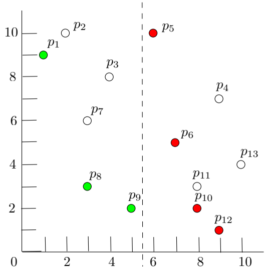
How about $A_{right} = \{p_5, p_6, p_{10}, p_{12}\}$.



Observation: Let q be a point in A_{right}. It is in the final answer if and only if no white point from A_{left} dominates q.

Skyline

Motivated by the observation, we determine the final answer by reducing the problem to 1D dominance screening.



Construct a 1D dominance screening problem with input sets P', Q' as follows:

- P' = the projections of A_{left} onto the y-axis.
- Q' = the projections of A_{right} onto the y-axis.

The y-coordinate of p_{10} needs to be increased infinitesimally. Think: why?

2D Skyline

The previous discussion points to the following divide-and-conquer algorithm (for computing the skyline of P):

1. Divide the points in P into two equal halves by x-coordinate. In this way, we obtain two instances of the skyline problem: (i) left instance P_1, and (ii) right instance P_2.
2. Solve the left and right instances, recursively. Let A_{left} and A_{right} be their answers, respectively.
3. Obtain a 1D dominance screening problem P', Q' where P' (Q') is the projection of A_{left} (A_{right}, resp.) onto the y-axis. Solve this instance to obtain its answer A'.
4. Return the points in A_{left} and those corresponding to the ones in A'.

2D Skyline

Let us now analyze the running time of our algorithm. Let $f(n)$ be the time on $n = |P| + |Q|$ points. We have:

$$f(n) \leq 2 \cdot f(n/2) + O(n)$$

Specifically, the two terms $f(n/2)$ capture the cost of solving the left and right instances, respectively. The term $O(n)$ captures the cost of Steps 1, 2, and 3—recall that the 1D dominance screening problem can be solved in $O(n)$ time.

For $n \leq 2$, clearly $f(n) = O(1)$.

Solving the recurrence gives: $f(n) = O(n \log n)$.

Skyline in d-dimensional Space

We now have proved that the skyline problem can be solved in $O(n \log n)$ time in 2D space.

Our algorithm can be immediately generalized to any dimensionality d. In general, we solve a d-dimensional problem by reducing it to a $(d-1)$-dimensional dominance screening problem. The running time of the algorithm is $O(n \log^{d-1} n)$.

The details have now become straightforward, and are left as an exercise for you.

Skyline in d-dimensional Space

We promised that the skyline problem can be solved in $O(n \log^{d-2} n)$ time. How?

The key lies in solving the 2D dominance screening problem in just $O(n)$ time, assuming that all the points have been sorted by x-dimension. The details are not required in the exam, but will be explained in an exercise.