On Finding Skyline Points for Range Queries in Plane

Anil Kishore Kalavagattu * Ananda Swarup Das † Kishore Kothapalli ‡ Kannan Srinathan §

Abstract

We consider the dominating point set reporting problem in two-dimension. We propose a data structure for finding the set of dominating points inside a given orthogonal query rectangle. Given a set of n points in the plane, it supports 4-sided queries in $O(\log n + k)$, where k is size of the output, using $O(n \log n)$ space. This work can be of application when range queries are generated using mobile devices with limited display capacity.

1 Introduction

Range searching is one of the widely studied topics in computational geometry. As stated in [1], the fascination for range searching is due to the fact that it has wide applications in geographic information systems, CAD tools, database retrieval, etc. Informally range aggregate is defined as follows: We are given a set S of objects which can be points or line segments and the like. We need to preprocess the data set into a data structure such that given a query object q, we can efficiently return the objects in $S \cap q$, the objects that belong to both S and q. A careful note of the above definition reflects that in a traditional range aggregate query, an algorithm designer is more focused in finding the result set efficiently and is not much bothered about the size of the result set. But this cannot be true any further. The advancement of technology has introduced the era of information revolution which led to information explosion. Imagine a user who is accessing a database in a server through his mobile device whose computing power as well as display model mechanism is sufficiently limited. Computation can be outsourced to server but returning the entire result set may not be a clever decision. Rahul et al. [7] have proposed an algorithm to return the top k answers of the result set. But for this purpose, our points in the data set have to be weighted by means of a preference function. But in practice, finding a good preference function is not an easy task. To address the issue, we borrow the concept of skyline query from the database community.

*International Institute of Information Technology , Hyderabad, India, anilkishore@research.iiit.ac.in

†International Institute of Information Technology , Hyderabad, India, anandaswarup@gmail.com

‡International Institute of Information Technology , Hyderabad, India, kkishore@iiit.ac.in

§International Institute of Information Technology , Hyderabad, India, srinathan@iiit.ac.in

2 Definitions

We are given a set of n points $P = \{p_1, \ldots, p_n\}$ in $\mathbb{R}^2$. Assuming all the points have distinct coordinates in each dimension, a point p_i is said to dominate a point p_j if $p_i(x) > p_j(x)$ and $p_i(y) > p_j(y)$. The set P' ($\subseteq P$) of all the points, each of which is not dominated by any other point in P is called the dominating set of P. They are also called maximal elements, or *maxima*, of the set P [2]. In the database terminology, these maximal points are also known as skyline points. A skyline query is then, given a set of points P, report the skyline points of P. The skyline point set P' forms a sample (representation) for the set P. In practice, often $|P'| < |P|$. However in the worst case scenario, $|P'| = |P|$. More information about skyline queries can be found in [6].

3 The Problem

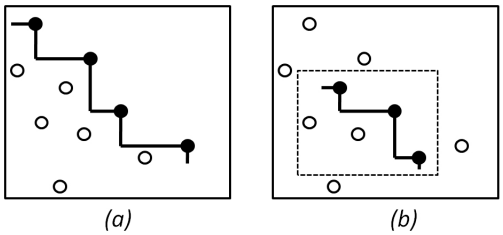

Figure 1: The nodes filled black are not dominated by any other point. *a.*) skyline of all the points. *b.*) skyline of points in the query rectangle (dotted border).

In this work, we study the following problem.

Problem 1 *We are given a set S of n points in $\mathbb{R}^2$. We wish to pre-process S into a data structure such that given an orthogonal query rectangle q, we can efficiently report the skyline points of $S \cap q$, where $S \cap q$ is the set of points in both S and q.*

Consider Fig 1.*a* There are ten points and among them only four points are not dominated by any other point. The same set of points are queried with an orthogonal rectangle, as shown in Fig 1.*b* and its easy to see that

the result can contain points which may not be in the skyline point set obtained by considering all the points. In rest of the paper, whenever we say size of result, we actually mean number of skyline points inside the query rectangle.

4 The General Algorithm

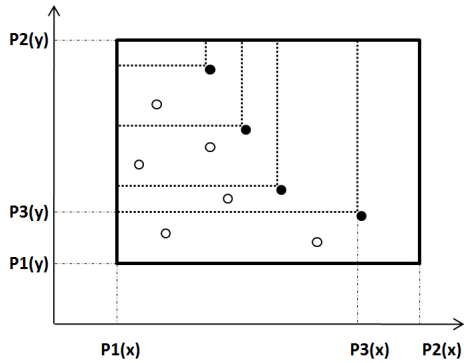

Figure 2: The query rectangle is $q = [P_1(x), P_2(x)] \times [P_1(y), P_2(y)]$ and the point $P_3 = (P_3(x), P_3(y))$ is the point with the largest x coordinate inside q.

A sketch of solution for the problem is as follows:

1. Let the set of points in the plane be S and the query rectangle be $q = [P_1(x), P_2(x)] \times [P_1(y), P_2(y)]$.
2. Find the point with the largest x coordinate in $S \cap q$. Let the point be $P_3 = (P_3(x), P_3(y))$. Add P_3 to the skyline point set.
3. Create a new rectangle $q' = [P_1(x), P_3(x)] \times [P_3(y), P_2(y)]$ and update $q \leftarrow q'$. See Fig. 2.
4. Repeat the steps 2, 3 until we get a rectangle q' such that $S \cap q' = \emptyset$. All the intermediate query rectangles encountered are also shown in Fig. 2.

As can be seen, our algorithm depends on a solution of range successor problem in plane. Range successor problem is widely studied in literature and Yu et al. in [9] proposed a data structure of size $O(n)$ using which range successor queries can be efficiently answered in $O(\frac{\log n}{\log \log n})$ time. Let k be the size of output, then a solution which repeatedly uses range successor query in step 2 above, will have a query time of $O(k\frac{\log n}{\log \log n})$. We propose a data structure of size $O(n \log n)$ using which our algorithm will have a query time of $O(\log n + k)$. Clearly using the proposed data structure, our algorithm will perform better whenever $k \geq O(\log n)$. Here we considered the static version of the problem, where the input points are fixed. A dynamic version of this problem with $O(\log^2 n + k)$ query time using $O(n \log n)$ space and $O(\log^2 n)$ time per update is studied in [4].

5 The Data Structure

5.1 Preprocessing

The data structure is a standard layered range tree [3] in which the main tree stores the points sorted by x-coordinates. Each secondary structure is an array storing y-coordinates in sorted order (decreasing), along with the corresponding point. This can be constructed by carrying out merge sort using y-coordinates as keys. In order to speed up query time, we use fractional cascading [3] at the cost of storing additional pointers at each node. Let w and v be the two children of μ. While merging the secondary arrays A_w and A_v to construct A_μ, we create and store pointers as follows. Each index i of A_μ stores a pointer to the smallest value in A_w which is greater than or equal to $A_\mu[i]$ and a pointer to the largest value in A_w which is smaller than or equal to $A_\mu[i]$. Similarly, two more such pointers are stored pointing to elements of A_v. Also, each index of A_w and A_v has a pointer to its corresponding position in the parent array A_μ. Each of these arrays A is preprocessed for range maxima queries [10] such that given two indices i, j of A_μ, we can find the point with maximum x coordinate among the points whose y coordinates are stored between $A_\mu[i]$ and $A_\mu[j]$ in O(1) time.

Lemma 1 *The storage space needed for the data structure is $O(n \log n)$. The preprocessing time needed to construct the data structure is $O(n \log n)$.*

This data structure is similar to the data structure used in [5] [8].

5.2 The Query Algorithm

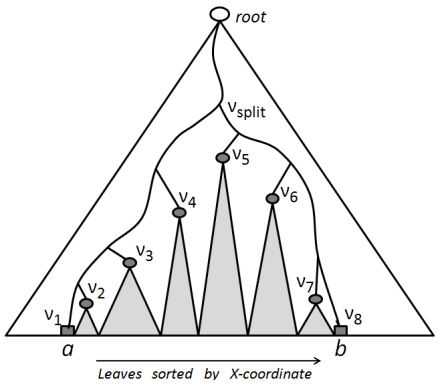

Figure 3: The dark nodes are the ones to which the segment $[a, b]$ of the query $q = [a, b] \times [c, d]$ is allocated.

After we construct the data structure, the query algorithm for a query rectangle $q = [a, b] \times [c, d]$ is as follows.

1. The range of x-coordinates in $[a, b]$ can be expressed as the disjoint union of $l = O(\log n)$ canonical subsets. Let the canonical subsets of nodes be $\nu_1, \nu_2, \ldots, \nu_l$ from left to right in that order, as shown in Fig. 3.

2. Find the node ν_{split}, which is the least common ancestor of ν_1 and ν_l. Find the largest sub-range of y-coordinates $\in [c, d]$ in $A_{\nu_{split}}$ using binary search and store the indices of the two ends.

3. Process the canonical nodes in reverse order, starting from ν_l back to ν_1, as follows. Initialize $i \leftarrow l$, $y_{low} \leftarrow c$ and $y_{high} \leftarrow d$.

4. Consider the node ν_i. Find the smallest index lt and the largest index rt such that $y_{high} \geq A_{\nu_i}[lt] \geq A_{\nu_i}[rt] \geq y_{low}$. Note that the y-coordinates are sorted in decreasing order in A_{ν_i}. This can be done by following the pointers from $A_{\nu_{i+1}}$ along the path to the node ν_i. For the starting node ν_l, follow the pointers from $A_{\nu_{split}}$ found in step 2.

5. Find the point with the largest x coordinate in $A_{\nu_i}[lt \ldots rt]$ using a range maxima query. Let this point be $p\prime$ and its y-coordinate be $p\prime(y)$. Report the point $p\prime$ and move rt to the position of the array just before $p\prime$ and update $y_{low} \leftarrow p\prime(y)$.

6. While there are points still left in $A_{\nu_i}[lt \ldots rt]$, i.e., $lt \leq rt$, repeat the above step.

7. At this point, we processed the nodes $\nu_l, \nu_{l-1}, \ldots, \nu_i$ and also have two pointers to the current limits on the y-coordinate $[y_{low}, y_{high}]$ at ν_i.

8. Set $i \leftarrow i - 1$. If $i \geq 1$, move to the node ν_i along with the pointers and repeat from step 4 , else exit.

Lemma 2 *The time needed for the query algorithm above is $O(\log n + k)$.*

Proof : At each of the $O(\log n)$ levels of the tree, at most two nodes are visited [3]. Among them, each canonical node with all its x-coordinates $\in [a, b]$ is visited at most once and each of the other nodes along the path traversed is visited at most three times, first time from its parent and at most one more time from each of its two children □

Using the above lemma, we can now conclude the section stating the following theorem.

Theorem 3 *A set S of n points in $\mathbb{R}^2$ can be preprocessed into a data structure of size $O(n \log n)$ such that given an orthogonal query rectangle q, we can efficiently report the set of points in $S \cap q$ not dominated by any other point in $S \cap q$ in $O(\log n + k)$ time where k is the size of the output.*

Our solution can be easily modified to report the top-m sky line points having largest(or smallest) y(or x) coordinates in $O(\log n + m)$ time.

6 Conclusion

In this work we studied the problem of finding the dominating set of points inside a query rectangle. Our solution is static and restricted to the plane. It will be interesting to see dynamic version of this problem and in higher dimensions.

7 Acknowledgements

We would like to thank the reviewers for their helpful and positive comments which have improved the paper and for pointing to [4] which handles the dynamic version of the problem discussed in this paper.

References

[1] P. Agarwal, S. Govindrajan, S. Muthukrishnan. Range Searching in Categorical Data: Colored Range Searching on Grid. *In Proceedings of ESA*, pp. 17–28, 2002

[2] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, v.23 n.4, p.214-229, April 1980

[3] M. de. Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. Computational Geometry: Algorithms and Applications. *ISBN 3-540-65620-0, Springer Verlag*, 2000

[4] G. S. Brodal, K. Tsakalidis. Dynamic Planar Range Maxima Queries. *In Proc. 38th International Colloquium on Automata, Languages, and Programming*, vol 6755 of LNCS, pp. 256-267. Springer Verlag, Berlin, 2011

[5] A. S. Das, P. Gupta, K. Srinathan, K. Kothapalli. Finding Maximum Density Axes Parallel Regions for Weighted Point Sets. *submitted to CCCG 2011*

[6] D. Kossmann, F. Ramsak, S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. *In Proc. International Conference on Very Large Data Bases*, pp. 275-286, 2002

[7] S. Rahul, P. Gupta, R. Janardan, K. S. Rajan. Efficient top-k queries for orthogonal ranges. *In Proc. International Workshop on Algorithms and Computation, Springer Verlag LNCS No. 6552*, pp. 110–121

[8] Sanjeev Saxena. Dominance made simple. *Information Processing Letters*, v.109 n.9, p.419-421, April, 2009

[9] C. C. Yu, W. K. Hon, B. F. Wang. Improved Data Structures for Orthogonal Range Successor Queries. *Computational Geometry: Theory and Applications* **44** , pp. 148– 159, 2011

[10] H. Yuan, M. Atallah. Data Structures for Range Minimum Queries in Multidimensional Arrays. *In Proceedings of SODA*, pp. 150–160, 2010