Journal of Mathematical Modelling and Algorithms **3:** 31–38, 2004.
© 2004 *Kluwer Academic Publishers. Printed in the Netherlands.*

Planar Maximum Box Problem

MICHAEL SEGAL
Communication Systems Engineering Department, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel. e-mail: segal@cse.bgu.ac.il

(Received in final form and accepted: 12 February 2004)

Abstract. Given two finite sets of points X^+ and X^- in $\mathbb{R}^d$, the maximum box problem asks to find an axis-parallel box B such that $B \cap X^- = \emptyset$ and the total number of points from X^+ covered is maximized. In this paper we consider the version of the problem for $d = 2$ (and find the smallest solution box). We present an $O(n^3 \log^4 n)$ runtime algorithm, thus improving previously best known solution by almost quadratic factor.

Mathematics Subject Classifications (2000): 52C45, 68U05.

Key words: discrete optimization, sorted matrices, axis-parallel.

1. Introduction

We consider the following planar *maximum box* problem. Given two sets X^+ and X^- of n points each in the plane, find the smallest axis-parallel rectangle (box) B that maximizes the number of covered points from X^+ and does not contain any point from X^-. This problem has several applications in the numerous data analysis problems, where we are interested to find patterns which intersect exactly one of given two sets. A systematic study of criteria for selecting the most useful patterns for classification of data using logical analysis of data was presented in [7]. It was shown that the best results are obtained by using inclusion-wise maximal boxes. Also, the solution to the problem might be very useful in computer graphics sampling [3]. The maximum box problem in arbitrary dimension d has been considered by Eckstein *et al.* [4]. They [4] proved that when d is a part of the input, the problem is NP-hard and gave $O(n^{2d+1})$ runtime algorithm for a fixed d. In this paper we show how to improve the runtime of their solution by providing $O(n^3 \log^4 n)$ time algorithm, thus improving their result by almost quadratic factor for $d = 2$. Our algorithm is based on several geometric observations together with Frederickson and Johnson optimization technique [6] of sorted matrices, despite of non-monotonicity of the problem. This paper is organized as follows. In the next section we describe the basic algorithm for the planar maximum box problem followed by the optimization technique of Frederickson and Johnson [6] and a binary search. At the end we conclude the paper.

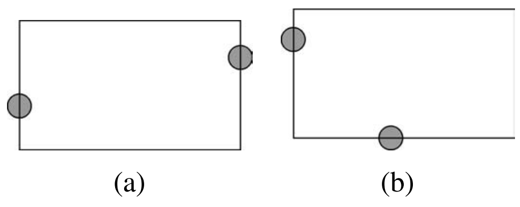

Figure 1. Two different configurations.

2. The Algorithm

We assume without loss of generality that the coordinates of all points are distinct, since we can enforce it by a small perturbation [5]. We distinguish between two basic types of the solution to our problem (see Figure 1): its edges (since otherwise we can shrink B) all its edges.

- Box B gets two points from X^+ lying on opposite sides of B.
- Box B gets two points from X^+ lying on adjacent sides of B.

We are interested in solving the decision version of both cases two and applying an optimization technique of sorted matrices developed by Frederickson and Johnson [6]. So, the decision version of the problem is defined as follows: Given an area A and a number $k > 2$, find whether exist a box B of area A that contains at least two points (otherwise, B can be shrunk) of X^+ on its edges such that $B \cap X^- = \emptyset$ and the total number of points of X^+ covered by B is at least k. If we can get an algorithm for the above problem, then the optimization algorithm will perform a simple binary search over the possible integer values of k (that vary between 0 and n) looking for a box with a smallest area. The largest value of k for which a box exists will provide an answer to our case – in fact, we will be able to find the smallest box that contains the maximal number of points from X^+ without any point from X^-.

Now we describe how to deal with these cases. For the first case we assume that B contains $p_i, p_j \in X^+$ such that p_i lies at the left side of B and p_j lies at the right side of B (the other case when points are located at the top and bottom sides of B is solved in the same fashion). The fixed width $|x(p_j) - x(p_i)|$ ($x(p_i)$ and $y(p_i)$ denote the x and y-coordinate of p_i, respectively) of B determines a maximum height h satisfying $|x(p_j) - x(p_i)| * h = A$. The algorithm slides B of maximal possible size from top to bottom over the sets $X^+ \cup X^-$, keeping the number of points of X^+ in the current position of B and checking whether $B \cap X^-$ is empty. There are $O(n)$ steps in this algorithm. It is obvious that if the order of points of the set X^+ and the set X^- along the y-axis (x-axis) is known then the ordered list of updates can be constructed in $O(n)$ time. For example, suppose that the y-coordinate of the bottom edge of B after we put B in its initial position is Y. We can determine which points of X^+ and X^- are outside of B with y-coordinate less than Y by two binary searches over the sorted lists of y-coordinates of the points from X^+ and X^-. Next, while sliding B from top to bottom, we can check the next point from X^+ and X^- to be covered by B in constant time. We also maintain a sorted list of points (according to the y-coordinate) that are currently inside of B in order to determine what point will leave B in the sliding process. Notice that the deletions of the elements from this list are performed only from its head, while points are inserted only at its tail. Thus, simple linked list will accomplish this task. We also will need to check $O(n^2)$ opposite pairs of points (we sort the points of X^+ and X^- only once).

In the second case we assign two adjacent points p_k and p_l of X^+ to B. Assume, without loss of generality, that p_k is located on the left side of B while p_l is located at the bottom side of B. Denote by c the lower corner of B, i.e. vertex of the intersection of edges of B containing p_k and p_l, respectively. The decision algorithm for this case is based on the same idea as in the previous case. Namely, it tries all placements of B of area A and checks whether $B \cap X^-$ is empty while maintaining the total number of the points of X^+ in B. As it can be seen, all the boxes of area A whose lower left corner corner is c, have their upper right corner on hyperbola h. Notice, that the remaining uncovered points from $X^+ \cup X^-$ can be divided into three subsets X_1, X_2 and X_2. The subset X_1 consists of the points of $X^+ \cup X^-$ lying to the right of h, out of B. The set X_2 consists of the points above the sliding box B and the set X_2 consists of the points to the right of B and to the left of h. Clearly, X_1 does not change during the running of the algorithm, since B cannot cover any point from X. The set X_2 undergoes insertions only and the set X_3 undergoes deletions only. As in the previous case, by knowing the order of the points of $X^+ \cup X^-$ along the x-axis (and y-axis), we can construct the ordered list of updates for sets X_2, X_3 in linear time. All we need is to check $O(n^2)$ pairs of points from X^+ lying on adjacent edges.

2.1. THE OPTIMIZATION STEP

In order to find an optimal solution in the second case of our algorithm we will follow an optimization technique proposed by Frederickson and Johnson. It is based on constructing and searching in monotone matrices. We give a brief explanation of this approach. Consider a set S of arbitrary elements. *Selection* in the set S determines, for a given rank k, an element that is kth in some total ordering of S. The complexity of selection in S has been shown to be proportional to the cardinality of the set [1]. Fredrickson and Johnson [6] considered selection in a set of sorted matrices. An $n \times m$ matrix M is a *sorted matrix* if each row and each column of M is in nondecreasing order. Fredrickson and Johnson have demonstrated that selection in a set of sorted matrices, that together represent the set S, can be done in time sublinear in the size of S. They have also observed that, given certain constraints on the set S, one can construct implicitly the set of sorted matrices representing S. For instance, the sums of the pairs in a Cartesian product of two input sets, denoted by $X + Y$, can be represented by means of the sorted vectors X and Y.

THEOREM 1 ([6]). *Let M be a sorted matrix of dimension $n \times m$, where $n \geqslant m$. Let T_s be the runtime of a decision algorithm which, given a value A, answers 'yes' or 'no'. Assume that the answers are monotone with respect to A. Then the total time needed to find the least element in M, for which the answer is 'yes', is* $O(T_s \log n + n)$.

Remark. Notice, that in our case the answers are not monotone with respect to A. More specifically, consider the following cases.

1. Suppose that answer for a given value A is "yes". Then, clearly, in order to find smaller solution we have to decrease A.
2. Suppose that answer for a given value A is "no" and the reason is that in every stage of our decision algorithm that the box of area A contained less than k points from X^+. In this case, we should definitely increase the value of A.
3. Suppose that answer for a given value A is "no" and the reason is that it was at least one stage when our algorithm found a box with at least k points from X^+, it also contained some of the points of X^-; in all other steps (if any) the box contained less than k points from X^+. This case is problematic: it seems that we should decrease it since it may lead that a box may still stay with at least k points from X^+ inside while being empty from points of X^-. On the other hand if we increase the box, it may happen that box may cover additional points from X^+, thus reaching a lower bound of k, without including any point of X^-.

We show how we can deal with the case 3. We will identify case 3 during the performance of our algorithm and perform some actions that will lead to an ultimate decision. We track our decision algorithm for the steps when the box is still empty from the points of X^- while containing less than k points from X^+. Our goal, is to check already during the decision step whether we can obtain a solution by increasing A without using Frederickson–Johnson technique. When we encounter such an event (call it E) we can check the existence of the larger solution using the orthogonal range trees.

Consider Figure 2. As we explained above, during the execution of the algorithm for a fixed value of A we can have two cases (a) and (b). In case (a) we

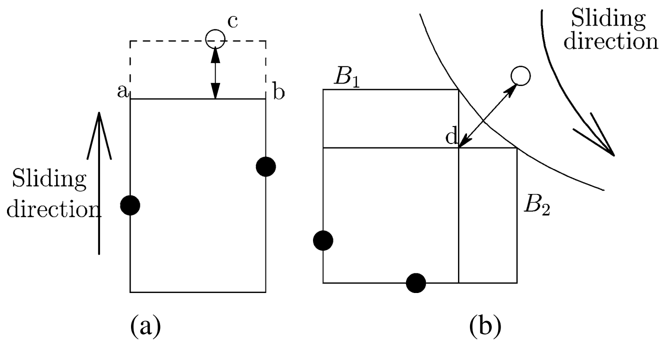

Figure 2. Checking phase in two cases.

slide our box vertically (or horizontally) until we encounter an event E when the box contains $k', k' < k$ points from X^+ (and still empty from points of X^-. We then allow our box to grow, in order to find the closest point c from X^- to the line segment $[a, b]$ that lies between a and b. We count the number of points k'' from X^+ that lie in the box B' bounded by a, b and c. If $k' + k'' \geqslant k$ than we keep a value of $A + \text{area}(B')$ as a possible value for our solution, and continue our decision algorithm. In case (b) we slide our box according to hyperbola until we encounter an event E with a box B_1. We continue to slide it until some of the points of X^- moves inside of B_2 (notice that during process of sliding all of the intermediate boxes cover less than k points of X^+). We are interested in finding the closest point c from X^- to the intersection point d of B_1 and B_2. We compute the number of points of X^+ inside of box B' that is defined by two black adjacent points in left and lower edges of B_1 whose upper edge has a y-coordinate equal to the minimum of y-coordinates of upper edge of B_1 and c, and whose right edge has a x-coordinate equal to the minimum of y-coordinates of right edge of B_2 and c. If we have at least k points of X^+ we keep a value of $A + \text{area}(B')$ as a possible value for our solution, and continue our decision algorithm. At the end we compute the smallest value between the values we evaluated during the algorithm and output it.

We can find the closest point from X^- to the line segment by constructing orthogonal range tree T [2]. The main structure of T is a balanced binary tree according to the x-coordinate of points from X^-. Each node v of this tree corresponds to the balanced binary tree (secondary tree) according to the y-coordinate of points from X^- whose x-coordinate belongs to the subtree rooted at v. We augment this data structure by keeping an additional value for each node w in the secondary data structures as the minimal value of the actual y-coordinates of the points corresponding to the nodes in the subtree rooted at w. In order to find the closest point from X^- to the line segment, we perform a query on T by taking a range as one side unbounded strip bounded by the line segment on other side. At most $O(\log^2 n)$ nodes of the secondary data structure are taken into account and we collect all the minimal y-values that are kept in these nodes. A point that has a minimal y-coordinate is an answer. When we are interested to find the closest point from X^- to the other query point d, our range is two-bounded (horizontally and vertically) plane with an apex at d. For this, we will need the following small observation. If we pass a bisector (45°) throw d it will cut the quadrant of a plane into two wedges: the distance from points in the upper edge to d will be determined by their y-coordinate, while the distance from points in the lower edge to d will be determined by their x-coordinate. We will, thus, two orthogonal range trees (each one corresponding to the wedge), where one of them will keep additional values as the minimal value of the y-coordinates of the points, while the other will keep additional values as the minimal value of the x-coordinates of the points. Therefore the query can be performed in $O(\log^2 n)$ time as well as counting the number of points from X^+ inside the given query box.

Thus, the answer obtained by the decision algorithm for a given area A can be classified as follows. If it is "yes", we have to decrease A for a smallest solution. If it is "no" with a fact that in every stage of the box of area A contained less than k points from X^+, we increase A. If it is "no" from other reason than above, our algorithm also we return another value A' if it exists ($A' > A$ and A' is a smallest such a value) for which the answer is "yes". We also decide to decrease the value of A' from the reason explained in case 3.

The algorithm performs a sequence of iterations which includes matrices of smaller size in each iteration. The matrices in any iteration are divided into sub-matrices called *cells*. In each iteration, two representative elements are chosen from each cell, the smallest value and the largest value. These representative elements are used to discard certain cells from further consideration. For ease of exposition it is assumed that M is a square matrix, whose dimension is a power of 2 (if not, we can extend the size of matrix). Hence every cell will be of size which is a power of 4. After a number of iterations all cells consist of single elements. Continue the iterations as before, except without cell division, until a single element remains.

The structure of the matrix induces a partition of the set of remaining cells into subsets called *chains*. Two cells belong to the same chain if and only if they are in the same diagonal of the cells obtained from the original matrix M by partitioning it into sub-matrices of the same dimensions as the cells. Let b_i be the ratio of the dimension of matrix M to the current cell dimension at the end of the ith iteration. Clearly, $b_i = 2^i$. The maximum possible number of chains after splitting cells on the ith iteration is $2b_i - 1$. Interesting that the number of cells does not increase too quickly as the iterations progress.

LEMMA 2 ([6]). *Let $B_i = 4b_i - 1$. For all iterations in which the cells are divided, the number of remaining cells after the ith iteration is not greater than B_i.*

From the preceding lemma, the number of cells remaining at the end of iteration $i - 1$ is no more than B_{i-1}. Hence no more than $O(B_i)$ work is done in dividing and selecting among cells on the ith iteration. Thus the total work for dividing and selecting cells is $O(\Sigma_i(B_i)) = O(\Sigma_i(b_i)) = O(n)$. Iterations with no cell division will begin when there are $O(n)$ elements. The number of remaining elements will decrease by half each time, yielding $O(n)$ time for the iterations, ultimately giving a least *feasible element* – the entry in matrix with the positive answer. For feasibility testing, $O(\log n)$ iterations with cell division are performed, and $O(\log n)$ iterations without cell division (cells with one element) are performed. Hence all feasibility testing requires $O(T_d \log n)$ time, yielding the total $O(T_d \log n + n)$ runtime of the algorithm.

We observe that the set of all distances measured along the x-axis (y-axis) between the points of X^+ can be represented implicitly as a sorted matrix M of size $n \times n$ if the order of the points of X^+ along the x-axis (y-axis) is known.

Let us consider the geometry of the optimal solution for the planar maximum box problem and determine the potential values of area A for the optimal solution.

There can be two kinds of solutions according to the two cases of positioning points on the edges of B. In the first case, box B has its width (height) fixed and thus its area A is defined by the second parameter – height (width). As we observed above we can represent all these values as a sorted matrix M of size $n \times n$. As one can point out M also will contain redundant values but they will have no affect on the correctness or runtime of the algorithm. In the second case the optimal box with the area A, has its width and height defined by two pairs of points lying on its adjacent sides, where one pair (p_k, p_l) is fixed. The distances from each point p_k, p_l to the other points of X^+, measured along the x-axis (y-axis respectively) define an array X (resp. Y) of linear size. Having the arrays X and Y available we can represent all potential values of A in the second case for fixed (p_k, p_l) as a sorted matrix M of size $n \times n$. Notice that for each pair of points of X^+ lying on the adjacent sides of B we can build a corresponding sorted matrix M in $\mathrm{O}(n)$ time. Finally we remark, that when we obtain an answer “no” by using Fredrickson and Johnson technique we also keep a list of other values (if exist) that we obtain from our decision algorithm (case 3). At the end we choose the smallest value between the one obtained by the Fredrickson and Johnson technique and the the other that belongs to the list of case 3.

By applying the optimization technique of Fredrickson and Johnson [6] we obtain that the second case of planar maximum box problem can be solved in $\mathrm{O}(n^3 \log^3 n)$ time.

Thus we have the following result:

THEOREM 3. *For a given number k, we can find the smallest area box (if exists) that cover at least k points from X^+ without any point from X^- in* $\mathrm{O}(n^3 \log^3 n)$ *time.*

Doing a binary search over the values of k yields the following theorem.

THEOREM 4. *The planar maximum box problem can be solved in* $\mathrm{O}(n^3 \log^4 n)$ *time.*

3. Conclusions

In this paper we presented an $\mathrm{O}(n^3 \log^4 n)$ algorithm for the planar maximum box problem by improving currently best solution with $\mathrm{O}(n^5)$ runtime. It is not trivial to generalize our solution for arbitrary dimension since the partition of the solution into different topological cases as it done in this paper does not lead to the improvement in the running time of the algorithm. In particular, instead of two basic cases that we dealt here, we will have four basic cases that require to spend more computational resources (time and space) in order to solve them. It also would be interesting to solve the variation of this problem by finding a maximum ball instead of a maximum box.

References

1. Blum, M., Floyd, R., Pratt, V., Rivest, R. and Tarjan, R.: Time bounds for selection, *J. Comput. System Sci.* **7**(4) (1973), 448–461.
2. de Berg, M., van Kreveld, M., Overmars, M. and Schwartzkopf, O.: *Computational Geometry, Algorithms and Applications*, Springer-Verlag, 1997.
3. Dobkin, D., Gunopulos, D. and Maass, W.: Computing the maximum bichromatic discrepancy with applications to computer graphics and machine learning, *J. Comput. System Sci.* **52**(3) (1996), 453–470.
4. Eckstein, J., Hammer, P. L., Liu, Y., Nediak, M. and Simeone, B.: The maximum box problem and its application to data analysis, *Comput. Optim. Appl.* **23** (2002), 285–298.
5. Edelsbrunner, H. and Mucke, E. P.: Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms, *ACM Trans. Graphics* **9** (1990), 66–104.
6. Frederickson, G. and Johnson, D.: Generalized selection and ranking: Sorted matrices, *SIAM J. Comput.* **13** (1984), 14–30.
7. Hammer, P., Kogan, A., Simeone, B. and Szedmak, S.: Pareto-optimal patterns in logical analysis of data, RUTCOR Research Report 7-2001, 2001.