On Piercing Sets of Axis-Parallel Rectangles and Rings

Michael Segal

Department of Mathematics and Computer Science,
Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel

Abstract. We consider the p-piercing problem for axis-parallel rectangles. We are given a collection of axis-parallel rectangles in the plane, and wish to determine whether there exists a set of p points whose union intersects all the given rectangles. We present efficient algorithms for finding a piercing set (i.e., a set of p points as above) for values of $p = 1, 2, 3, 4, 5$. The result for 4 and 5-piercing improves an existing result of $O(n \log^3 n)$ and $O(n \log^4 n)$ to $O(n \log n)$ time, and is applied to find a better rectilinear 5-center algorithm. We improve the existing algorithm for general (but fixed) p, and we also extend our algorithms to higher dimensional space. We also consider the problem of piercing a set of rectangular rings.

1 Introduction

Let $\mathcal{R}$ be a set of n axis-parallel rectangles in the plane, and let p be a positive integer. $\mathcal{R}$ is called *p-pierceable* if there exists a set of p *piercing points* which intersects every member in $\mathcal{R}$. Our problem, thus, is to determine whether $\mathcal{R}$ is p-pierceable, and, if so, to produce a set of p piercing points.

There are several papers in which the p-piercing problem for axis-parallel rectangles was investigated; let us mention only the very recent papers. The 1-piercing problem was easily solved in linear time using the observation that 1-piercing problem for rectangles is equivalent to finding whether the intersection of rectangles empty or not. In Sharir and Welzl [7] 2- and 3-piercing problems in the plane are solved in linear time, while they reach only $O(n \log^3 n)$ bound for the 4-piercing problem and $O(n \log^4 n)$ bound for the 5-piercing problem. Katz and Nielsen [2] present a linear time algorithm for d-dimensional boxes ($d \geq 2$), when $p = 2$. In this paper we present a new technique which allows to obtain simple linear time algorithms for $p = 1, 2, 3$, and obtain an $O(n \log n)$ time solution for $p = 4, 5$, thus improving the previous results of [7]. We improve the existing algorithm of [7] for general (but fixed) p, and we extend our algorithms to higher dimensional space. We also consider the problem of piercing the set of rectangular rings. The boundary of a *rectangular ring* consists of two concentric rectangles, where the inner rectangle is fully contained in the outer one, however, the vertical and horizontal widths of the ring, are not necessarily equal.

This paper is organized as follows. We first demonstrate our technique (Section 2) in the case of $p = 1$. We then describe this method (Section 3) for the case of $p = 2, 3$. In Section 4 an $O(n \log n)$ time algorithm for 4-piercing is given. In Section 5 we present an $O(n \log n)$ time algorithm for the case of $p = 5$ and describe

generalizations of the problem. Section 6 deals with piercing sets of rectangular rings. We conclude in Section 7.

2 Rectilinear 1-piercing

We are given a set $\mathcal{R}$ of n axis-parallel rectangles in the plane; The goal is to decide whether their intersection is empty or not. We begin with an observation due to Samet [5].

Let P be the set of 4 dimensional points representing the parameters of $\mathcal{R}$. Let $P_x = \{p_1^x, \ldots, p_n^x\}$ be the projections of the x-intervals of $\mathcal{R}$ into the plane (c_x, d_x), and let $P_y = \{p_1^y, \ldots, p_n^y\}$ be the projections of the y-intervals of $\mathcal{R}$ into the plane (c_y, d_y).

If a shape R is described by k parameters, then this set of parameter values defines a point in a k-dimensional space assigned to the class of shapes. Such a point is termed a *representative point*. Note, that a representative point and the class to which it belongs completely define all of the topological and geometric properties of the corresponding shape.

The class of two-dimensional axis-parallel rectangles in the plane is described by a representative point in four dimensional space. One choice for the parameters is the x and y coordinates of the centroid of the rectangle, denoted by c_x, c_y, together with its horizontal and vertical extents (i.e. the horizontal and vertical distances from the centroid to the relevant sides), denoted by d_x, d_y. In this case a rectangle is represented by the four-tuple (c_x, d_x, c_y, d_y) interpreted as the Cartesian product of a horizontal and a vertical one-dimensional *interval*: (c_x, d_x) and (c_y, d_y), respectively. A query that asks which rectangles contain a given point is easy to implement (see Figure 1).

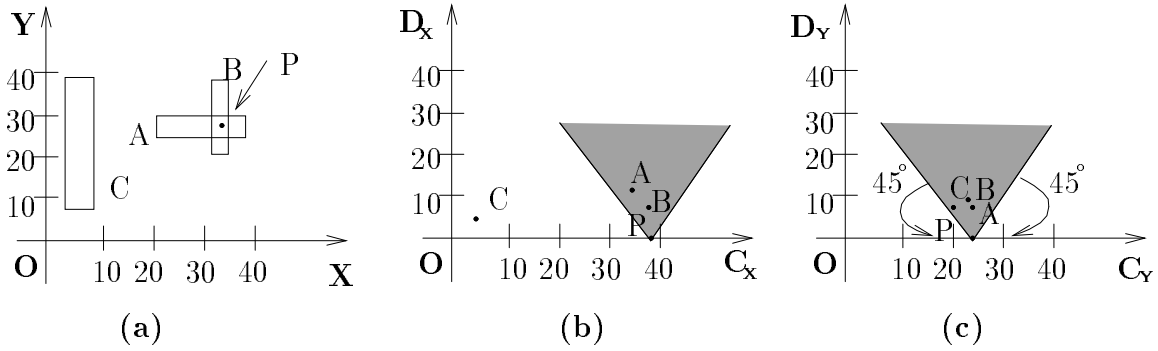

Fig. 1. (a) There are 3 rectangles, and P is a query point. All intervals containing P are in the shaded regions. Intervals appearing in the shaded regions of both (b) and (c) correspond to rectangles that contain P.

A query point P is represented by a four-tuple $(p_x, 0, p_y, 0)$. We transform the rectangles (A,B,C) in Figure 1(a) into the points in two 2-dimensional spaces ((c_x, d_x)

and (c_y, d_y)) (Figure 1(b) and 1(c)). There are two points representing P in these 2-dimensional spaces. $(p_x, 0)$ in (c_x, d_x)-space, and $(p_y, 0)$ in (c_y, d_y)-space. It is easy to see that all the rectangles that contain P must be transformed into two cones in these spaces respectively (the shades cones in Figure 1). These cones have apexes on the $(p_x, 0)$ and $(p_y, 0)$ respectively and are of slope 45° and 135°. In Figure 1, A and B are in both cones and thus P is in these rectangles.

In order to find whether the set $\mathcal{R}$ is 1-pierceable, we find in each 2-dimensional space the rightmost intersection point R_x (R_y) of the 45° lines through the points of P_x (P_y) with axis c_x (c_y), and the leftmost intersection point (L_x and L_y respectively) of the 135° lines with through P_x and P_y respectively, axes c_x and c_y respectively. If the intervals $[R_x, L_x]$ and $[R_y, L_y]$ exist (are not empty) then a point P whose projections are in these intervals is a piercing point.

Thus, we can conclude by the following theorem:

Theorem 1. *We can find whether a set of n axis-parallel rectangles is 1-pierceable in $O(n)$ time, and give a solution, if it exists, in the same runtime.*

3 Rectilinear 2- and 3-piercing

We begin with the 2-piercing problem. Similarly to the previous section, we have to find whether there exist four cones $C_1, C_2 \in (c_x, d_x)$ and $C_3, C_4 \in (c_y, d_y)$ such that:

1. $C_1 \cup C_2$ covers P_x.
2. $C_3 \cup C_4$ covers P_y.
3. Denote by $[C_i]$ the set of all the points of P that corresponded to the points of P_x (or P_y) covered by C_i.

At least one of the following two conditions is true:

(i) $([C_1] \cap [C_3]) \cup ([C_2] \cap [C_4])$ contains all the points of P. This will imply that the apexes of C_1, C_3 define one piercing point and apexes of C_2, C_4 define the other piercing point.
(ii) $([C_1] \cap [C_4]) \cup ([C_2] \cap [C_3])$ contains all the points of P. This will imply that the apexes of C_1, C_4 define one piercing point and apexes of C_2, C_3 define the other piercing point.

We can *constrain* the locations of the cones C_1, C_2, C_3, C_4. They are defined by minimal and maximal points of intersection of the 45° and 135° lines with the horizontal axes in the two planes (c_x, d_x) and (c_y, d_y) respectively. It is easy to see that in order for the rectangles to be 2-pierceable, we put, wlog, the apex of C_1 on R_x, C_2 on L_x, C_3 on R_y and C_4 on L_y. Clearly, if these cones cover all the points then the set $\mathcal{R}$ is 2-pierceable.

In the case of 3-piercing, we have to find six cones $C_i, 1 \leq i \leq 6$, which will define three piercing points with the following properties:

1. $C_1 \cup C_2 \cup C_3$ covers P_x.
2. $C_4 \cup C_5 \cup C_6$ covers P_y.

3. For $i,k,z \in \{1,2,3\}$, pairwise disjoint and $j,l,h \in \{4,5,6\}$, pairwise disjoint

$$|([C_i] \cap [C_j]) \cup ([C_k] \cap [C_l]) \cup ([C_z] \cap [C_h])| = n$$

for at least one combination of i,k,z (there are at most 6 combinations), where the union is taken is without repetitions.

W.l.o.g., we can find the constrained cones C_1, C_3, C_4, C_6 as in the algorithm for 2-piercing. Namely, the left boundary of C_1 (C_4) is constrained by the leftmost 135° line through the points of P_x (P_y), and the right boundary of C_3 (C_6) is constrained by the rightmost 45° line through the points of P_x (P_y).

To fulfill condition (3) we look at each combination: $[C_1] \cap [C_4]$ or $[C_1] \cap [C_6]$ or $[C_3] \cap [C_4]$ or $[C_3] \cap [C_6]$ and for these four possibilities we check in linear time, whether the rest the points is 2-pierceable. Thus we conclude:

Theorem 2. *We can check in linear time whether set of n axis-parallel rectangles is 2- or 3-pierceable and give a solution, if exists, in the same runtime.*

4 Rectilinear 4-piercing

Now we have to find eight cones $C_i, 1 \leq i \leq 8$ with the following properties:

1. $C_1 \cup C_2 \cup C_3 \cup C_4$ covers P_x.
2. $C_5 \cup C_6 \cup C_7 \cup C_8$ covers P_y.
3. For some pair of cones $C_i, C_j, i \in \{1,2,3,4\}, j \in \{5,6,7,8\}$ the set of all rectangles without those covered by $[C_i] \cap [C_j]$ is 3-pierceable.

As before, assume wlog that C_1, C_4, C_5, C_8 are constrained, so condition (3) when we choose $i \in \{1,4\}$ and $j \in \{5,8\}$ is easily checked in linear time, because we can find the location of C_1, C_4, C_5, C_8 in linear time and then answer the 3-piercing problem in linear time. If $i \in \{2,3\}$ and $j \in \{6,7\}$ then there exist $i' \in \{1,4\}$ and $j' \in \{5,8\}$ such that if the set of rectangles is 4-pierceable then one piercing point must be determined by the cones $C_{i'}$ and $C_{j'}$. So this case is also computed in $O(n)$ time. The worst (and the more interesting) case is when each constrained cone in one plane corresponds to a non-constrained cone in the other plane. Let us look at one such pair (there is a finite number of such pairs), wlog, C_3 in (c_x, d_x) and C_5 in (c_y, d_y). The analysis for all other such pairs is almost identical.

We sort all the 45° (135°) lines determined by P_x in (c_x, d_x) plane, and do the same to the lines determined by P_y in (c_y, d_y) plane. Clearly, the apex of C_3 is between the apexes of C_1 and C_4. So, we fix the apex of C_3 to coincide with the apex of C_1 and begin to move it rightwards towards C_4. We define an *event* when a point of P_x is inserted or deleted from C_3. Initially, we compute the set of points $A \subseteq P$ covered by $[C_3] \cap [C_5]$ (when the apex of C_3 is determined by the leftmost 135° line through the point of P_x) and apply the 3-piercing algorithm for the rest of the points $S = P - A$, allowing, only this time, C_1 to move freely. If we have a positive answer, we are done; otherwise we continue.

We move C_3 rightwards to the next event and change S accordingly (as in Figure 2). The first next event is when the leftmost point of P_x is deleted from C_3. Then

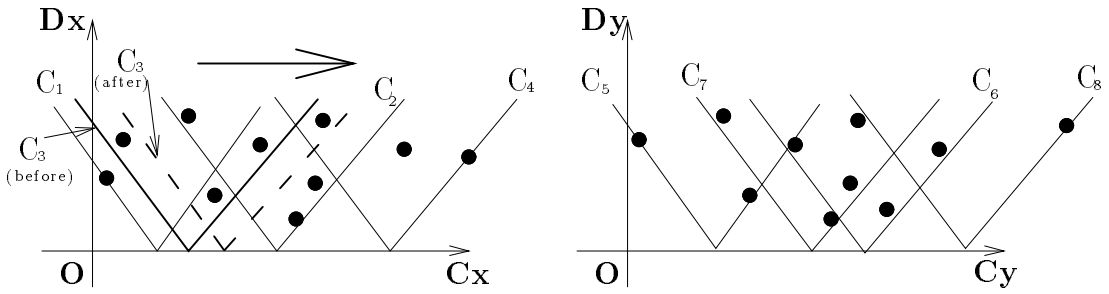

Fig. 2. Moving apex of C_3 from apex of C_1 towards apex of C_4.

we run again the 3-piercing algorithm for S. Here, too, if S is 3-pierceable then we are done. Clearly, from now on the location of the apexes of C_1, C_4 and C_8 will not change during the whole algorithm because these cones are defined by the extreme points of P_x and P_y that will never appear in both C_3 and C_5. Let C_7 be the leftmost cone covering S in (c_y, d_y). The location of C_7 will change since C_7 will move towards C_8 and back to cover points. But once C_7 moves back from C_8 it will never move towards C_8 again. This is because C_5 is constrained and C_7, the second cone from the left, moves back to cover points that got out of A. Since the leftmost point has to be covered in order to have 4-piercing, once C_7 got back to its leftmost position, it will never move to the right again. Thus, the number of changes that we perform on C_7 is $O(n)$. Our goal is to determine the location of the cones C_2 in (c_x, d_x) and C_6 in (c_y, d_y). We will check the possible combinations of pairing the cones to create piercing points. Assuming the cones C_3 and C_5 describe a piercing point, we have the following combinations for the rest of the piercing points:
(a) $(C_1, C_7), (C_2, C_6), (C_4, C_8)$, (b) $(C_1, C_7), (C_2, C_8), (C_4, C_6)$,
(c) $(C_1, C_8), (C_2, C_7), (C_4, C_6)$, (d) $(C_1, C_8), (C_2, C_6), (C_4, C_7)$,
(e) $(C_4, C_7), (C_1, C_6), (C_2, C_8)$, (f) $(C_4, C_8), (C_1, C_6), (C_2, C_7)$.

Observation 3 *The combinations of the cones at each step of the* 4*-piercing algorithm are* **independent**, *meaning that we check* 3*-pierceability for each fixed combination of the cones throughout all the steps of the* 4*-piercing algorithm. If we get a negative answer for a combination, we check the other combinations. If there is a solution it will be found by the algorithm in one of the steps of one combination.*

According to observation 3, because we have finite number of combinations, we can perform the 4-piercing algorithm for each one of the combinations separately. For each of the combinations, the 4-piercing algorithm is slightly different. Denote by $C_{ij} = [C_i] \cap [C_j] \cap S$. Recall that $S = P - A$, A being the points covered by (C_3, C_5).

We present a skeleton of the algorithm and then give the additional technical details.

For every combination of cones the following events happen during the 4-piercing algorithm: (exemplified by C_3 and C_5 as a piercing point and the 3-piercing combinations (a)-(f) as above):

1. Initially the left boundary point q of C_3 is getting out of C_3. If $q \in A$, we re-run the 3-piercing algorithm, otherwise no update is needed, since q did not belong to A, it was, and remains, in S..
2. If, when we move the apex of C_3 towards C_4, a point q' is inserted to C_3, we first check if the corresponding point to q' in (c_y, d_y) is covered by C_5. If it is not covered, then we continue moving C_3 to the next event; otherwise we have the following cases:
   2.1 If q' defines the new left boundary of the middle cone C_2 in (c_x, d_x), or q' defines the left boundary of the left cone C_7, or the left boundary of the middle cone C_6 in (c_y, d_y) for the combinations (a)-(d) (similarly, right boundary for the combinations (e)-(f)), then, for the given combination we perform the following updating scheme: we first check if q' defines the left boundary of C_7. If yes then we have to find, by binary search over S, the new left boundary for C_7 and:
      i. For combination (a). Find the new boundaries of the middle cones C_2 and C_6 in both planes and check whether they cover the rest of the points by simply computing and examining the set $S^{(1)} = S - C_{17} - C_{48}$. Note that the cones C_4 and C_8 are both constrained and do not move during the whole algorithm.
      ii. For combination (b) (similarly (e)). By computing and examining the set C_{17} (C_{14}) we found the new left boundaries of C_2 and C_6. Only thing we have to do now is to check whether the pairs (C_2, C_8) and (C_4, C_6) $((C_1, C_6))$ cover the set of all points of P not covered by (C_3, C_5) and (C_1, C_7) $((C_4, C_7))$. This could be done by computing the sets $S^{(2)} = C_{48} - C_{17}$ $(C_{18} - C_{47})$, C'_4 and C'_8 and updating T_1 and T_2 as described below. Using the updating scheme below, we check whether C_2 covers C'_8, C_6 covers C'_4, and together C_2 and C_6 cover $S^{(2)}$.
      iii. For combination (c) (similarly (f)). By computing and examining the set $S^{(3)} = S - C_{18} - C_{27}$ we find the leftmost and rightmost points of this set in both planes that should be covered in both planes by C_6 and C_4. We find the new boundaries of C_2 and C_6 and check whether C_6 and C_4 cover the leftmost and rightmost points in both plane that we just found. Note that the number of updates on C_2 in the whole algorithm is $O(n)$. This is because the left boundary of C_2 is defined by the leftmost point (of S) in (c_x, d_x) not covered by C_{18} and thus C_2 moves towards and back from C_4, but when it moves back it will never move rightwards again.
      iv. For combination (d). By computing and examining the set $S^{(4)} = S - C_{18} - C_{47}$ we find the leftmost and rightmost points of this set in both planes that should be covered in both planes by C_6 and C_2. We find the new boundaries of C_2 and C_6 and check whether C_6 and C_2 cover the leftmost and rightmost points in both plane that we just found.
   2.2 If q' does not define a left boundary of a cones as above, then for each combination we perform an identical updating scheme as in 2.1 but without computing a new left boundary of the middle cones C_2 and C_6.

   After the updates we check whether there is a 3-piercing combination for S..
3. If, when we move apex of C_3, a point q'' is deleted from C_3, then

3.1 If $q'' \notin C_5$ we proceed to the next event.

3.2 If in the past q'' was the left boundary of the middle cone C_2 in (c_x, d_x), or q' was the left boundary of the left cone C_7, or the left boundary of the middle cone C_6 in (c_y, d_y) for the combinations (a)-(d) (similarly, right boundary for the combinations (e)-(f)), then, for the given combination we perform the following updating scheme: If q'' defines a new left boundary C_7, then we compute a new location of C_7 and:

i. For combination (a), find the new boundaries of the middle cones C_2 and C_6 in both planes and compute the rightmost and leftmost points of the set $S^{(1)}$ in both planes.

ii. For combination (b), by examining the set C_{17} find the new left boundaries of C_2 and C_6, compute the sets $S^{(2)}, C_4'$ and C_8' and update T_1 and T_2.

iii. For combination (c), find the new boundaries of C_2 and C_6. By examining the set $S^{(3)}$ find the leftmost and rightmost points of this set in both planes.

iv. For combination (d), find the new boundaries of C_2 and C_6. By examining the set $S^{(4)}$ we find the leftmost and rightmost points of this set in both planes.

3.3 If q' does not define a left boundary of a cones as above, then for each combination we perform an identical updating scheme as in 3.2 but without computing a new left boundary of the middle cones C_2 and C_6. Notice that in this case (when q'' is deleted from C_3) the 4-piercing of P is not possible, because it wasn't possible in previous step of the algorithm.

Now we describe the technical details of the 4-piercing algorithm given above. For combination (a) we compute C_{48} at the beginning of the 4-piercing algorithm. The cones C_4 and C_8 are both constrained and do not move during the whole algorithm. For each step of the 4-piercing algorithm we maintain the set $S^{(1)} = S - C_{17} - C_{48}$. To determine whether C_2 and C_6 can cover $S^{(1)}$ we are only interested in the maxima and minima of the 45° and 135° lines through the points of this set ($S^{(1)}$) in both planes respectively. Note that the total number of updates on C_{48} and on C_7 is at most $O(n)$, thus if we maintain the points of the dynamically changing set $S^{(1)}$ sorted according to the 45° and 135° lines we can update $S^{(1)}$ and find the maxima and minima in both planes by a simple binary search. Consequently, in $O(1)$ time we check whether there exist two cones C_2 and C_6 with boundaries on these maximal and minimal values that cover these points.

Combinations (b) and (e) are similar in the sense that C_7 (that has $O(n)$ updates) is paired with a constrained cone, C_1 in (b) and C_4 in (e), and the non constrained cones C_2 and C_6 are each paired with a constrained cone. For combination (b) (similarly (e)) we compute the set C_{48} (C_{18}) at the beginning of the 4-piercing algorithm. In each step of the 4-piercing algorithm we compute the set $S^{(2)} = C_{48} - C_{17}(C_{18} - C_{47})$. Observe the set of all points of P not pierced by (C_3, C_5) and (C_1, C_7) $((C_4, C_7))$. They will have to be pierced by (C_2, C_8) and (C_4, C_6) $((C_1, C_6))$. Now the points in $S^{(2)}$ should be covered by either C_2 or C_6, whereas the points of $C_4' = [C_4] - S^{(2)}$ $(C_1' = [C_1] - S^{(2)})$ must be covered by C_6 and the points of $C_8' = [C_8] - S^{(2)}$ must be covered by C_2. C_1 and C_7 (C_4 and C_7) determine the

left (right) boundary of C_2 and C_6, which are found by a binary search over the points of $S - C_{17}$ ($S - C_{47}$). As for combination (a) the number of updates on C_{48} (C_{18}), and C_7 is at most $O(n)$. The sets C'_4 (C'_1) and C'_8 are maintained sorted according to the lines throughout the whole algorithm. To check how $S^{(2)}$ is pierced, we maintain balanced binary trees T_1, T_2. The leaves of $T_1(T_2)$ contain the set $S^{(2)}$ sorted according to the 45° (135°) lines in the plane (c_x, d_x). Let T be T_1 or T_2. Initially, the leaves of T contain the sorted points of C_{48} (C_{18}) in the plane (c_x, d_x). After we compute C_{17} (C_{47}) for the first time we empty the leaves that contain the points that belong to $C_{17}(C_{47})$. Now T contains the sorted lines through the points of $S^{(2)}$. Let p be a point of $S^{(2)}$. A leaf corresponding to p contains the x value of the point of intersection of the 45° (135°) line through p with the c_x axis in (c_x, d_x). It will also contain the y value of the point of intersection of a 45° (135°) line through p with the c_y axis in (c_y, d_y). An inner node $v \in T$ will contain the maximum of the y values corresponding to 135° lines of the leaves of the subtree rooted at v, and the minimum of the 45° lines. During the algorithm we perform a sequence of updates, namely insertions and deletions, on the tree T. When a point q is add to $S^{(2)}$, then we insert it into T in a sorted x-order and update the minimum and maximum y values on the nodes of path from the leaf q to the root of T. If a point q is deleted from T, then we find the leaf of q, delete it and update the y values of the nodes on path from the leaf to the root of T. Each update of T takes $O(\log n)$ time. We can check, using the tree T, whether C_2 together with C_6 cover all the points in $S^{(2)}$. For combination (c) (similarly, (f)) at the beginning of the 4-piercing algorithm we compute C_{18}. The cones C_1 and C_8 are constrained and do not move during the whole algorithm. At the next step of the 4-piercing algorithm we work with the set $S^{(3)} = S - C_{18} - C_{27}$ and find the leftmost and rightmost points in this set that should be covered in both planes by C_6 and C_4 respectively. We maintain $S^{(3)}$ by incremental updates according to the motion of C_3.

For combination (d) we perform a scheme almost identical to that of (c), but with the difference that at each step of the 4-piercing algorithm we work with the set $S^{(4)} = S - C_{18} - C_{47}$ and find the leftmost and rightmost points that should be covered in both planes by C_6 and C_2. Again, we update $S^{(4)}$ at each motion of C_3 in logarithmic time.

From the analysis of this algorithm it follows that we have $O(n)$ updates in the whole algorithm and we can perform each update in logarithmic time. Thus,

Theorem 4. *We can determine whether set of n axis-parallel rectangles is 4-pierceable or not in $O(n \log n)$ time, and give the solution (if it exists) in the same runtime.*

5 Rectilinear 5-piercing

Now we have to find ten cones $C_i, 1 \leq i \leq 10$ with the following properties:

1. $C_1 \cup C_2 \cup C_3 \cup C_4 \cup C_5$ covers P_x.
2. $C_6 \cup C_7 \cup C_8 \cup C_9 \cup C_{10}$ covers P_y.
3. For some pair of cones $C_i, C_j, i \in \{1,2,3,4,5\}, j \in \{6,7,8,9,10\}$ the set of all rectangles without those covered by $[C_i] \cap [C_j]$ is 4-pierceable.

Due to the duality relation between our analysis and that in [7] (see Section 7) we follow the case analysis in [7]. Assume, wlog, that C_1, C_5, C_6, C_{10} are constrained and the order of the cones is from left to right. We may also assume that one of the following situations occurs:

(a) There is one pair of constrained cones C_i, C_j, $i \in \{1, 5\}$ and $j \in \{6, 10\}$. We try all of these possibilities, find the set of rectangles not covered by the given pair of cones, and test whether this set is 4-pierceable, using the preceding algorithm. This takes $O(n \log n)$ time.

(b) Every constrained cone is paired with a non-constrained cone. Since there are four constrained cones there are two pairs with the same constrained cones. We proceed as follows. First, we guess a *unique* constrained cone, say C_1, which is paired with a non-constrained, say C_7. Then we proceed as in 4-piercing algorithm, i.e. slide C_7 from left to right, starting at the apex of C_6 and stopping when we reach the apex of C_{10}. In each move, we check whether the set of the rest of the rectangles is 4-pierceable using the following observation by Sharir and Welzl [7]. They observe that the 4-piercing problem (that is solved at each move of C_7) has always a pair of two constrained cones in its solution. In our case they are either C_2 and C_6, or C_2 and C_{10} (C_2 becomes constrained after computing $S - C_{17}$). We process each of these cases separately. Assume, wlog, we process C_2 and C_6. Then at each move of C_7 we update C_{26} and check whether the rest of rectangles is 3-pierceable as in the update step in the 4-piercing algorithm. Omitting the easy missing details, we obtain a procedure that runs in $O(n \log n)$ time.

(c) There is some pair of the cones, where an unconstrained cone is paired with another unconstrained cone. Assume, wlog, the cones are C_4 and C_8. We also assume wlog that we have paired C_6 and C_3, C_{10} and C_2, C_1 and C_7, C_5 and C_9 (a constrained cone with an unconstrained cone). Then, as was observed in [7], either at least one of the c_x-coordinates of the apexes of C_2 and C_3 is smaller then the the c_x-coordinate of the apex of C_4 or at least one of the c_x-coordinates of the apexes of C_2 and C_3 is larger than the c_x-coordinate of the apex of C_4. Suppose one of them is smaller than C_4. Then we slide C_7 (that is paired with C_1) from left to right, starting at the apex of C_6 and stopping when we reach the apex of C_{10}. In each move, we check whether the set of the rest of rectangles is 4-pierceable. As was claimed in [7] again in each move of C_7 it has to be that C_2 is paired with either C_6 or C_{10}. Thus we have a situation identical to case (b). This can be computed as in case (b) above, implying that the computing for case (c) can also be done in $O(n \log n)$ time. Hence we obtain:

Theorem 5. *We can determine whether set of n axis-parallel rectangles is 5-pierceable or not in $O(n \log n)$ time, and give the solution (if it exists) in the same runtime.*

The result for 5-piercing improves an existing result of [7] that runs in $O(n \log^4 n)$ time, and can be applied to find a better rectilinear 5-center algorithm in time $O(n \log^2 n)$.

Higher dimensions and $p > 4$.

Our technique immediately implies a linear time algorithm for 2-pierceability of a set of axis-parallel rectangles for arbitrary (fixed) dimension $d, d \geq 2$ (there are only constrained cones) and an $O(n \log n)$ time algorithm for 3-pierceability of a set of axis-parallel rectangles for dimension $d, 3 \leq d \leq 5$ (the same result was obtained by [1] independently). In the later problem there is always a combination where $d-1$ cones are constrained and one (at most) is a non-constrained cone. At each step of the algorithm there is a finite number of the d-coupling combinations of the cones. An algorithm similar to the 4-piercing algorithm in the plane is used to solve the piercing problem. We also obtain an improved formula for general (but fixed) $p \geq 6$ (for the plane) using our approach. The general observation is that a constrained cone is always paired with a constrained or unconstrained cone. Thus for solving $p+1$-piercing problem we have to consider two cases. In the first case there is two constrained cones paired together, we can determine the rest of the (uncovered) rectangles in linear time and apply algorithm for p-piercing for the rest of rectangles. In the second case, a constrained cone is paired with a non-constrained. We move the apex of the non-constrained cone between the apexes of the constrained cones in its plane. Thus we have $O(n)$ steps (when a point is either inserted or deleted from the non-constrained cone). In each step we run the $p-4$-piercing algorithm for the rest of the points. Thus we improve the algorithm of [7] for $p \geq 6$ in the plane from $O(n^{p-4} \log^5 n)$ to $O(n^{p-4} \log n)$.

6 Piercing sets of rectangular rings

A *rectangular ring* is a ring, defined by two boundaries, the outer boundary and the inner boundary. Both boundaries are axis parallel concentric rectangles, where the inner rectangle is fully contained in the outer rectangle. We do not require that the horizontal width of the ring be identical to the vertical width. We pose the piercing question on a set of rectangular rings.

1-piercing

The 1-piercing problem is equivalent to the question: Given a set R of n axis-parallel rectangular rings, is their intersection empty or not. This problem can be easily solved by decomposing the rings into $4n$ rectangles and applying the segment tree [3] to compute *the depth* of the set of rectangles in $O(n \log n)$ time. Our method, that is easily extendable to higher dimensions, also uses the *Klee measure* (the depth and the union of a set of rectangles are examples for the Klee measure, see [3, 4]).

First we use the algorithm from Section 2 to find whether the set R_r of the external rectangles defining the given rings is 1-pierceable. If R_r is not 1-pierceable then neither is the set R. Otherwise, we find the region Q (also a rectangle) where all the rectangles from R_r intersect. In our notations Q is determined as follows. Let P be the set of 4 dimensional points representing the parameters of R_r. Let $P_x = \{p_1^x, \ldots, p_n^x\}$ be the projections of the x-intervals of R_r into the plane (c_x, d_x), and let $P_y = \{p_1^y, \ldots, p_n^y\}$ be the projections of the y-intervals of R_r into the plane (c_y, d_y). We find in each plane (c_x, d_x) and (c_y, d_y) the rightmost intersection point

R_x (R_y) of the 45° lines through the points of P_x (P_y) with axis c_x (c_y), and the leftmost intersection point (L_x and L_y respectively) of the 135° lines through P_x and P_y respectively, with axes c_x and c_y respectively. The intervals $[R_x, L_x]$ and $[R_y, L_y]$ (if they exist, namely, $R_x < L_x$ and $R_y < L_y$) define Q. Now we check whether the union of the rectangles, defined by the internal boundaries of the rectangles in R, covers Q. If it does not cover Q, then R is 1-pierceable; otherwise it is not 1-pierceable.

In higher dimensional space Q is easily found as above in time $O(dn \log n)$. In order to find the union of the internal rectangles we use the algorithm of Overmars and Yap [4] who solve the Klee measure problem in higher dimensions in time $O(n^{\lfloor \frac{d}{2} \rfloor} \log n)$. Thus, this is the runtime of our 1-piercing algorithm for rings for $d \geq 2$.

2 and 3-piercing

For two and three pierceability problems a non-trivial but quadratic algorithm is as follows. We first check whether the set of the external rectangles is 2-piercable. If it is, then we continue to work with the combination of the cones that define the 2-piercing points (there might be more than one combination of cones, and we consider all of them).

Assume that the combination is (C_1, C_3) and (C_2, C_4). First we check whether there exist points of P covered by both (C_1, C_3) and (C_2, C_4). We call such points of P *joint points*. The case with no joint points is easy. We only need solve two separate 1-piercing subproblems for each pair of cones, find the rectangular regions Q' and Q'' (as was Q region in the 1-piercing algorithm) and check whether the internal rectangles corresponding to each subproblem cover Q' and Q'' respectively.

If there are joint points of P in (C_1, C_3) and (C_2, C_4) then we proceed as follows. Initially we assign the joint points to (C_1, C_3), and the points not covered by (C_1, C_3) we assign to (C_2, C_4). Similarly to the described above, we compute the intervals $[R_x, L_x]$ and $[R_y, L_y]$ for each pair of cones according to the points they cover, the joint points belonging only to (C_1, C_3). We denote these intervals by $I_{C_1}, I_{C_2}, I_{C_3}, I_{C_4}$. The intervals I_{C_1}, I_{C_3} define the rectangular region Q' where the first piercing point should be found, and the intervals I_{C_2}, I_{C_4} define the region Q'' for the second piercing point. We now check whether the internal rectangles corresponding to the points assigned to (C_1, C_3) and (C_2, C_4), respectively, cover Q' and Q'' respectively. If both Q' and Q'' are not wholly covered by the internal rectangles then we are done.

In the next steps we slide C_1 from right to left, stopping whenever a joint point q leaves C_1. The joint point q is deleted from C_1 (thus, it is now assigned to (C_2, C_4). We compute the intervals $I_{C_1}, I_{C_2}, I_{C_3}, I_{C_4}$, and check whether the corresponding internal rectangles cover Q' and Q''. We stop sliding C_1 either when it gets to the last of the joint points or when it gets to a point in (C_1, C_3) which is not a joint point. We call the latter event a *stop event*.

After we finish sliding C_1 we return it to its starting position. We now perform similar steps with C_3 moving from left to right and stopping at each joint point till the end of joint points or till a point covered by just (C_2, C_4) is met by the sliding cone C_3. There are $O(n)$ sliding steps in this algorithm. In each step we check whether the internal rectangles cover the regions Q' and Q''. This can be done in $O(n \log n)$ time using standart sweep-line algorithm. So we conclude that our algorithm runs in time $O(n^2 \log n)$.

For the 3-piercing problem we can apply a similar technique and obtain an $O(n^3 \log n)$ algorithm for solving the 3-pierceability of rings. We can improve these running time using the following observation.

Observation 6 *Throughout the motion of C_1 in the 2-piercing algorithm above, the rectangular region Q' does not shrink in any dimension, while the region Q'' does not expand in any dimension.*

More precisely, at each stop of C_1 the intervals I_{C_3} and I_{C_1} do not shrink, and if they change they can only grow, while the intervals I_{C_2} and I_{C_4} can only shrink. This is because the joint point that was deleted from C_1 has to be covered by C_2 (and also C_4) thus decreasing the freedom of movement of the apex of C_2 (C_4). Moreover, the number of internal rectangles of the rings that can cover Q' (Q'') does not increase (decrease) in each step of the algorithm.

This observation provides a monotonicity to the problem and allows us to improve the running time of our algorithm. Instead of sliding C_1 (C_3) from right to left stopping at each joint point, we move $C_1(C_3)$ in the range of joint points (between the rightmost joint point and the rightmost stop event) by a binary search, checking at each such move whether both Q' and Q'' are covered or not by the union of the corresponding inner rectangles. When we find a move where they are not covered then we done. If Q' is covered C_1 jumps to the left. If Q' is not covered but Q'' is covered, then C_1 jumps to the right. Thus we get a factor $O(\log n)$ instead of $O(n)$ for the 2-piercing algorithm, and the problem is solved in time $O(n \log^2 n)$.

The 3-piercing algorithm works just as we sketched above. Here, too, we can do the motions of C_1 and C_4 in binary skips, and then apply the just described 2-piercing algorithm, getting us to a $O(n \log^3 n)$-time algorithm.

We can solve the 2-piercing problem in higher dimensional space. We get an $O(n^{\lfloor \frac{d}{2} \rfloor} \log^2 n)$, $d \geq 3$ runtime algorithm for determining 2-pierceability of the set of input rings using as a subroutine algorithm in [4].

7 Conclusions

In this paper we present an efficient technique for solving the p-piercing problem for a set of axis-parallel rectangles. There is some duality between the analysis of [7] and that of our paper. A constrained cone in our algorithms corresponds to an edge on the boundary of the search region in [7], and two paired constrained cones in our algorithms correspond to a corner in the search region of [7]. We are looking into

applying a similar technique for sets of triangles, rhombi, etc. The most intriguing question is whether we can improve the runtime of the presented algorithm for p-piercing problems where $p > 5$. We hope that our approach can help in to obtaining a better solution to these problems.

Acknowledgements: I thank Matya Katz, Klara Kedem and Yuri Rabinovich for useful discussions.

References

1. E. Assa, M. Katz, private communication.
2. M. Katz, F. Nielsen, "On piercing sets of objects", In *Proc. 12th ACM Symp. on Computational Geometry*, 1996.
3. K. Mehlhorn, *Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry*, Springer-Verlag, 1984.
4. M. Overmars, C. Yap, "New upper bounds in Klee's measure problem", In *Proc. 29 Annual IEEE Symp. on the Found. of Comput. Sci.*, 1988.
5. H. Samet, "The design and analysis of spatial data structures", *Addison-Wesley*, 1990
6. M.Segal, K.Kedem, "Enclosing k points in the smallest axis parallel rectangle", *8th Canadian Conference on Computational Geometry*, 1996.
7. M. Sharir, E. Welzl, "Rectilinear and polygonal p-piercing and p-center problems", In *Proc. 12th ACM Symp. on Computational Geometry*, 1996.

This article was processed using the LaTeX macro package with LLNCS style