The rectangle enclosure and point-dominance problems revisited

Prosenjit Gupta*† Ravi Janardan*† Michiel Smid‡
Bhaskar Dasgupta§†

August 22, 1995

Abstract

We consider the problem of reporting the pairwise enclosures among a set of n axes-parallel rectangles in $\mathbb{R}^2$, which is equivalent to reporting dominance pairs in a set of n points in $\mathbb{R}^4$. For more than ten years, it has been an open problem whether these problems can be solved faster than in $O(n \log^2 n + k)$ time, where k denotes the number of reported pairs. First, we give a divide-and-conquer algorithm that matches the $O(n)$ space and $O(n \log^2 n + k)$ time bounds of the algorithm of Lee and Preparata [LP82], but is simpler. Then we give another algorithm that uses $O(n)$ space and runs in $O(n \log n \log\log n + k \log\log n)$ time. For the special case where the rectangles have at most α different aspect ratios, we give an algorithm that runs in $O(\alpha n \log n + k)$ time and uses $O(n)$ space.

1 Introduction

The problem of computing intersections in a set of rectangles has received much attention. (See Chapter 8 of [PS88].) There are several variants of the problem depending on the notion of "intersection" that is used. In this paper, we consider the following version of the problem:

Problem 1.1 *Given a set $\mathcal{R}$ of n axes-parallel rectangles in the plane, report all pairs (R', R) of rectangles such that R* encloses *R'.*

*Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A. E-mail: {pgupta,janardan}@cs.umn.edu.

†The research of these authors was supported in part by NSF grant CCR-92-00270. Part of this work was done while PG was visiting the Max-Planck-Institut für Informatik. PG thanks the MPI and the International Computer Science Institute for partial support.

‡Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany. E-mail: michiel@mpi-sb.mpg.de. This author was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

§DIMACS, Rutgers University, Piscataway, NJ 08855, U.S.A. E-mail: bhaskar@dimacs.rutgers.edu.

This problem finds applications in the computer-aided-design of VLSI circuits [LP82].

By mapping each rectangle $R = [l, r] \times [b, t]$ to the point $(l, b, -r, -t)$ in $\mathbb{R}^4$, we can formulate this problem as a dominance problem: If $p = (p_1, p_2, p_3, p_4)$ and $q = (q_1, q_2, q_3, q_4)$ are points in $\mathbb{R}^4$, then we say that p *dominates* q if $p_i \geq q_i$ for all i, $1 \leq i \leq 4$. We call the pair (p, q) a *dominance pair.* Using this terminology, Problem 1.1 is transformed—in linear time—into the following one:

Problem 1.2 *Given a set V of n points in $\mathbb{R}^4$, report all dominance pairs in V.*

In fact, a result of Edelsbrunner and Overmars [EO82] implies that Problems 1.1 and 1.2 are equivalent, i.e., in linear time, Problem 1.2 can also be transformed into Problem 1.1.

Let k denote the number of pairs (R', R) of rectangles such that R encloses R', or, equivalently, the number of dominance pairs in V. In 1982, Lee and Preparata [LP82] showed how Problem 1.1 and, hence, also Problem 1.2, can be solved in $O(n \log^2 n + k)$ time and $O(n)$ space. They mention as an open problem whether this can be improved. Also Bentley and Wood [BW80], Preparata and Shamos [PS88] and Bistiolas et al.[BST93] mention this open problem. To our knowledge, the algorithm of Lee and Preparata has never been improved.

In this paper, we first give a divide-and-conquer algorithm that achieves the same running time as in [LP82] but which is simpler.

Then, we give an alternative algorithm having a running time of $O(n \log n \log\log n + k \log\log n)$ and using $O(n)$ space. This algorithm first *normalizes* the points of V in the sense that for each i, $1 \leq i \leq 4$, the i-th coordinate of each point is replaced by its rank among all i-th coordinates. This gives a set S of n points in U^4, where $U = \{0, 1, 2, \ldots, n-1\}$. Dominance pairs in V are in one-to-one correspondence with dominance pairs in S. We solve the problem for the set S by means of a divide-and-conquer algorithm. In the merge step of this algorithm, we have to report red-blue dominances in a set of red and blue points in U^3. This problem is solved using a sequence of non-trivial sweep steps: In a first sweep, we remove all red points that are not dominated by any blue point, and all blue points that do not dominate any red point. We call this the *cleaning step.* Assume wlog that after this step the number of red points is at least equal to the number of blue points. Then we perform a *reporting step*, in which we sweep again and find all red-blue dominance pairs (r, b) where b is maximal in the set of blue points. Because of the cleaning step, we are guaranteed to find a number of dominance pairs which is at least proportional to the number of red and blue points that remain after the cleaning step. Hence, we can charge the time for this reporting step to the number of reported dominance pairs. Since we have found all dominance pairs in which the maximal elements of the blue points occur, we can remove them. Then, we perform the cleaning step on the remaining red and blue points and, afterwards, we perform a reporting step again. We repeat this until either there are no red points left or there are no blue points left. At the end of the algorithm, we will have reported all red-blue dominance pairs.

Since the points have coordinates from a finite universe, we can use van Emde Boas trees [vEB77a, vEB77b] in order to search among them in $O(\log\log n)$ time rather than $O(\log n)$ time. In this way, the merge step of the divide-and-conquer algorithm takes

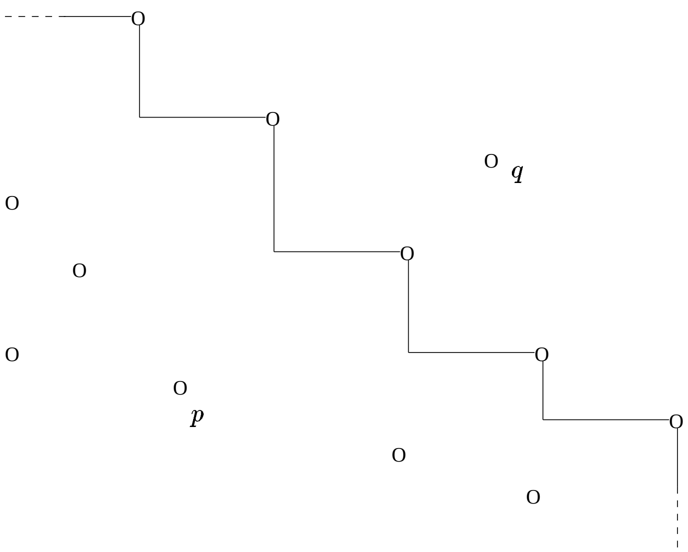

Figure 1: *The maximal layer of a planar point set S forms a contour. The point p is inside the contour, whereas the point q is outside. Note that q does not belong to S.*

$O((n+k')\log\log n)$ time, where k' is the number of dominance pairs that are reported in this step.

In the final part of the paper, we consider Problem 1.1 for the special case where the rectangles have at most α different *aspect ratios*, where the aspect ratio of a rectangle is defined as the ratio of its height to its width. We give a simple plane sweep algorithm that solves this case of the problem in $O(\alpha n \log n + k)$ time and $O(n)$ space. (Previously, no results were known for this special case.)

1.1 Preliminaries

In the rest of this paper, we will mainly consider Problem 1.2. We will need some terminology. Let S be a set of n points in $\mathbb{R}^d$, where $d \geq 2$. Point p *dominates* point q if $p_i \geq q_i$ for all i, $1 \leq i \leq d$. A point of S is called *maximal in S* if it is not dominated by any other point of S. The *maximal layer* of S is defined as the subset of all points that are maximal in S.

If S is a set of points in the plane, i.e., if $d = 2$, then the maximal points, when sorted by their x-coordinates, form a *staircase*, also called a *contour*. See Figure 1. The ordering of the maximal points by x-coordinate is the same as the ordering by y-coordinate. Consider the contour of S. Let p be any point in the plane. We say that p is *inside* the contour if it is dominated by some point of the contour. Otherwise, we say that p is *outside* the contour.

2 A divide-and-conquer algorithm

Let V be a set of n distinct points in $\mathbb{R}^4$. We want to find all pairs $p, q \in V$ such that p is dominated by q. It turns out to be useful to first normalize the points of

V. In this way, we have to solve the problem for points with integer coordinates in $\{0, 1, 2, \ldots, n-1\}$. In the next subsection, we show how to normalize the set V. Then we give a divide-and-conquer algorithm that solves our problem.

2.1 The normalization step

For each i, $1 \leq i \leq 4$, we sort the vectors in the set

$$\{(p_i, p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_4) : (p_1, p_2, p_3, p_4) \in V\}$$

lexicographically. Then we replace the i-th coordinate of each point of V by its rank in this ordering. We denote the resulting set of points by S. The following lemma can easily be proved.

Lemma 2.1 *The normalization step takes $O(n \log n)$ time. It produces a set $S \subseteq \{0, 1, 2, \ldots, n-1\}^4$ of n points such that (p, q) is a dominance pair in V iff the corresponding pair in S is also a dominance pair. Moreover, for each i, $1 \leq i \leq 4$, no two points of S have the same i-th coordinate.*

2.2 The algorithm

Let S be the set of n points from Lemma 2.1. Note that during the normalization we can obtain the points of S sorted by their third coordinates. Our algorithm for finding all dominance pairs in S follows the divide-and-conquer paradigm. Since in each recursive call the number of points decreases, but the size of the universe remains the same, we introduce the latter as a separate variable u. Note that in our case $u = n$—the initial number of points. However, to keep our discussion general, we will derive our bounds in terms of both u and n and finally substitute n for u to get our main result, namely Theorem 3.2.

Hence, S is a set of n points in U^4, where $U = \{0, 1, 2, \ldots, u-1\}$. No two points of S have the same i-th coordinate for any i, $1 \leq i \leq 4$, and the points are sorted by their third coordinates. The algorithm is as follows:

1. Compute the median m of the fourth coordinates of the points of S. By walking along the points of S in their order according to the third coordinate, compute the sets $S_1 = \{p \in S : p_4 \leq m\}$ and $S_2 = \{p \in S : p_4 > m\}$. Both these sets are sorted by their third coordinates.

2. Using the same algorithm recursively, solve the problem for S_1 and S_2.

3. Let R (resp. B) be the set of "red" (resp. "blue") points in U^3 obtained by removing the fourth coordinate from each point of S_1 (resp. S_2). Compute all dominance pairs (r, b), where $r \in R$ and $b \in B$.

It is clear that this algorithm correctly solves the dominance reporting problem on S. The main problem is how to implement the third step. In the next section, we show how this three-dimensional red-blue dominance problem can be solved by a simple sweep algorithm.

2.3 The merge step

We use x, y and z to denote the coordinate axes in U^3. In the merge step, we sweep a plane parallel to the xy-plane downward along the z-direction. (Note that the points are already sorted by their third coordinates.) During the sweep, we maintain a radix priority search tree (PST), see [McC85], for the projections onto the sweep plane of all points of B that have been visited already. If the sweep plane visits a point (b_x, b_y, b_z) of B, then we insert (b_x, b_y) into the PST. If a point (r_x, r_y, r_z) of R is encountered, we query the PST and find all points (b_x, b_y) such that $b_x > r_x$ and $b_y > r_y$. For each such point, we report the corresponding pair in $R \times B$, or, in fact, in $S_1 \times S_2$.

It is easy to see that this sweep algorithm correctly solves the problem. The query algorithm for a PST takes time proportional to $\log u$ plus the number of reported points. Also, the PST can be updated in $O(\log u)$ time. Therefore, if k_{RB} denotes the number of red-blue dominance pairs in $R \times B$, then the algorithm takes time $O(n \log u + k_{RB})$.

Now we can analyze the 4-dimensional divide-and-conquer algorithm: Let $T(n, u)$ denote the running time on a set of n points in U^3 that are sorted by their third coordinates. Here, we do not include the time for reporting the dominance pairs. Then $T(n, u) = 2T(n/2, u) + O(n \log u)$, which solves to $T(n, u) = O(n \log n \log u) = O(n \log^2 n)$. If k denotes the total number of dominance pairs in the set S, then the entire algorithm takes time $O(n \log^2 n + k)$. It is easy to see that the algorithm uses $O(n)$ space.

Now consider our original set V of n points in $\mathbb{R}^4$. Since the normalization step takes $O(n \log n)$ time, we can solve the dominance problem on this set in $O(n \log^2 n + k)$ time and $O(n)$ space.

We remark that this algorithm is simpler than the algorithm given in [LP82, PS88]. Because of the normalization step, we can use a radix PST [McC85], which is a simple data structure not requiring any rebalancing. Immediately after the normalization step, we build a radix PST T in time $O(u + n) = O(u)$. Subsequently, during the rest of the algorithm, we insert and delete in the same tree T and perform queries on it. After every merge step, we take care to delete all remaining elements from T, so that it can be reused. (Note, however, that the normalization step is not crucial in obtaining the $O(n \log^2 n + k)$ running time; even without this step, the same running time can be obtained by using a balanced PST.)

In the next section, we give an alternative algorithm for the three-dimensional red-blue dominance problem, taking $O(n \log\log u + k_{RB} \log\log u)$ time. This will lead to an $O(n \log n \log\log n + k \log\log n)$ time algorithm for finding the k dominance pairs in V. To obtain this running time, the normalization step is necessary.

3 Red-blue dominance reporting in three dimensions

Recall the problem: We are given a set R of red points and a set B of blue points in U^3, and we have to find all dominance pairs (r, b) where $r \in R$ and $b \in B$. The points in both sets R and B are sorted by their third coordinates.

In the final algorithm, we first construct an empty van Emde Boas tree (vEB-tree) on the universe U. (See [vEB77a, vEB77b].) During the entire algorithm, elements will be inserted and deleted in this tree and we will perform queries on it. Its construction time is $O(u)$, its query and update times are $O(\log\log u)$ and it uses $O(u)$ space. In the rest of this section, we assume that we have this tree available.

First we give two subroutines that will be used in the final algorithm. The final algorithm itself is given in Section 3.3.

3.1 The cleaning step

One of the essential steps in our algorithm is one that removes points that do not participate in any dominance. That is, we remove all red points that are not dominated by any blue point, and all blue points that do not dominate any red point. We denote this as the "cleaning" of the red (resp. blue) set w.r.t. the blue (resp. red) set.

In [KO88], Karlsson and Overmars give an $O(n\log\log u)$ time and $O(u)$ space algorithm, which given n points in U^3, computes the maximal elements. We modify this algorithm to report all red points that are not dominated by any blue point, within the same time and space bounds:

We sweep a plane parallel to the xy-plane downward along the z-direction, stopping at each point. (The points are already available sorted by z-coordinate.) During the sweep, we maintain the contour of the two-dimensional maximal elements of the projections (onto the sweep plane) of the blue points already seen. We store these maximal elements in the initially empty vEB-tree, sorted by their x-coordinates. When the sweep plane visits a blue point b, we update the contour and the vEB-tree, as follows: We search in the vEB-tree with the x-coordinate of b and determine if b's projection is inside or outside the blue contour. If it is outside, then we delete from the vEB-tree all blue points on the contour whose projections are dominated by b's projection and we insert b as a new contour point. Note that the points to be deleted can easily be found since they are contiguous in the vEB-tree. (See Figure 2.)

On visiting a red point r, we query the vEB-tree with the x-coordinate of r and determine if r's projection lies inside or outside the blue contour. If it is inside, we insert r into an initially empty set R_1.

At the end of the sweep, we delete all elements from the vEB-tree. The empty tree will be used later on in the algorithm.

Lemma 3.1 *At the end of the algorithm, we have*

$$R_1 = \{r \in R : \exists b \in B \text{ such that } r \text{ is dominated by } b\}.$$

Moreover, the algorithm takes $O(n\log\log u)$ time and uses $O(u)$ space.

Proof. Let R' denote the set on the right-hand side. Let r be a point of R that is dominated by a blue point b. Then the sweep plane visits b before it visits r. Consider the moment when point r is visited. If, at this moment, b is on the contour, then r is inserted into R_1. Otherwise, there is a point b' on the contour such that $b'_x > b_x$ and $b'_y > b_y$. Since $b_x > r_x$ and $b_y > r_y$, it follows that r's projection lies inside the blue contour and, hence, r is inserted into R_1. This proves that $R' \subseteq R_1$. It can be proved

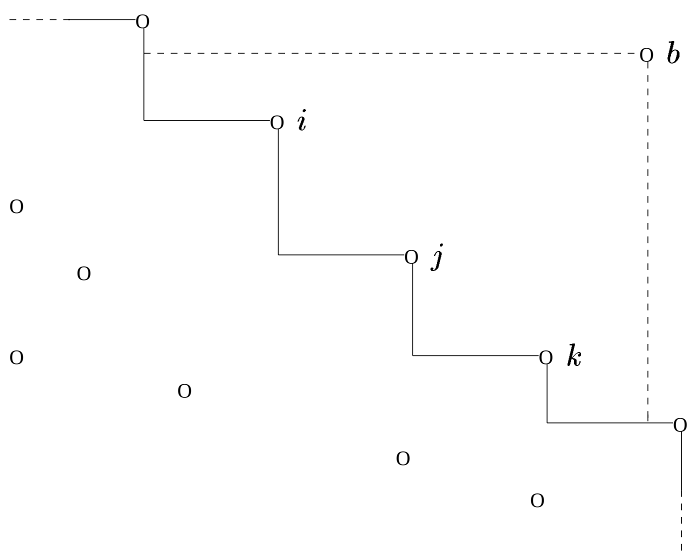

Figure 2: *Updating the blue contour when the sweep plane visits the point b. Points i, j and k are deleted and point b is inserted.*

---

in a similar way that $R_1 \subseteq R'$. The bounds on the running time and space follow from the complexity of the vEB-tree. (See [vEB77a, vEB77b].) $\Box$

The given algorithm cleans the red set R w.r.t. the blue set B. To clean B w.r.t. R, we use the mapping $\mathcal{F}$ that maps the point (a, b, c) in U^3 to the point $(u-1-a, u-1-b, u-1-c)$ in U^3. This mapping reverses all dominance relationships. Also, the mapping $\mathcal{F}$ is equal to its inverse. We run our sweep algorithm on the sets $\mathcal{F}(R)$ and $\mathcal{F}(B)$, maintaining a red contour and querying with the blue points. As a result, we get a set $B_0 \subseteq \mathcal{F}(B)$, where each point in B_0 is dominated by some point in $\mathcal{F}(R)$. Then the set $B_1 = \mathcal{F}(B_0)$ satisfies

$$B_1 = \{b \in B : \exists r \in R \text{ such that } b \text{ dominates } r\}.$$

Lemma 3.2 *Let R and B be sets of points in U^3 that are sorted by their third coordinates, and let $u = |U|$ and $n = |R| + |B|$. Assume that $n \leq u$. Also, assume we are given an empty vEB-tree on the universe U. In $O(n \log\log u)$ time and using $O(u)$ space, we can compute sets $R_1 \subseteq R$ and $B_1 \subseteq B$ such that*

$$R_1 = \{r \in R : \exists b \in B \textit{ such that } r \textit{ is dominated by } b\}.$$

and

$$B_1 = \{b \in B : \exists r \in R \textit{ such that } b \textit{ dominates } r\}.$$

The procedure that cleans the red set R w.r.t. the blue set B and returns the set R_1 will be denoted by $Clean(R, B)$. The set B_1 is obtained as $\mathcal{F}(Clean(\mathcal{F}(B), \mathcal{F}(R)))$.

Remark 3.1 Observe that it does not matter whether we first clean R w.r.t. B and then clean B w.r.t. R, or vice versa. In either case, we get the same clean sets R_1 and B_1.

3.2 The sweep and report step

Let R_1 and B_1 be the sets of Lemma 3.2. For the purpose of the discussion, let us assume that $|R_1| \geq |B_1|$. Let B_1' denote the three-dimensional maxima of B_1. The procedure *Sweep*(R_1, B_1), which will be described in this section, reports all red-blue dominance pairs (r, b), where $r \in R_1$ and $b \in B_1'$. Note that because of the cleaning step, there are at least $|R_1|$ such pairs. (If $|R_1| < |B_1|$, then we invoke the procedure *Sweep*$(\mathcal{F}(B_1), \mathcal{F}(R_1))$.)

Step 1: We sweep along the points of B_1 downwards in the z-direction and determine the set B_1': During the sweep, we maintain the contour of the 2-dimensional maximal elements of the projections of the points of B_1 already seen. These maximal elements are stored in the initially empty vEB-tree, sorted by their x-coordinates. We also maintain a list M, in which we store all updates that we make in the vEB-tree.

When the sweep plane visits a point b of B_1, we add b to an initially empty list L iff b's projection lies outside the current contour. In this case, we also update the contour by updating the vEB-tree, and we add the sequence of updates made to the list M.

At the end of the sweep, the list L contains the set B_1'. (See Lemma 3.3 below. Also, we will see that the during the sweep the contour changes iff the sweep plane visits a point of B_1'.)

Step 2: We now sweep along the points of $R_1 \cup B_1'$ upwards in the z-direction. Using the list M, we reconstruct the contour of the projections of the points of B_1' that are above the sweep plane. With each blue point b on the contour, we store a list $C_b \subseteq R_1$ of candidate red points.

Initially, the sweep plane is at the point having minimal z-coordinate and the vEB-tree stores the final contour from Step 1. For each blue point b on this contour, we initialize an empty list C_b.

When the sweep plane reaches a blue point b of B_1', we do the following:

2.1. Using M, we undo in the vEB-tree the changes we made to the two-dimensional blue contour when we visited b during the sweep of Step 1. Call each blue point which now appears on the contour a *new* point; call all remaining blue points on the contour *old*. Note that the new points form a single continuous staircase.

2.2. For each $r \in C_b$, we report (r, b) as a dominance pair.

2.3. For each new blue point q on the contour, we have to create a list C_q: We look at all points of C_b. For each such point r, we search with its x-coordinate in the vEB-tree. If r's projection is inside the new contour, then we find the leftmost blue point p of the new contour that is to the right of r. Starting at p we walk right along the contour. For each blue point q encountered such that q is new, we insert r into the list C_q. We stop walking as soon as we find a blue point q whose projection does not dominate r's projection or we reach the end of the contour. (See Figure 3.)

When the sweep plane reaches a red point r, we search with its x-coordinate in the vEB-tree and determine if its projection is inside or outside the current contour. If it is inside, we start walking along the contour from the point immediately to the right of r and insert r into the list C_q for each blue point q on the contour, until we reach a blue point whose projection does not dominate r's projection or we reach the end of the contour.

Note that at the end of Step 2, the vEB-tree is empty.

Lemma 3.3 *At the end of Step 1, the list L contains the set B_1' of three-dimensional maxima of B_1.*

Proof. Let $b \in L$ and assume that b is not maximal in B_1. Then there is a point b' in B_1 that dominates b. During the sweep, b' is visited before b. Since $b'_x > b_x$ and $b'_y > b_y$, b's projection lies inside the contour when the sweep plane reaches b. (Note that at that moment, b' does not necessarily belong to the contour.) Hence, b is not inserted into L. This is a contradiction and, hence, $L \subseteq B_1'$.

Conversely, let $b \in B_1'$. Consider the moment when the sweep plane reaches b. Assume that b's projection lies inside the contour. Then there is a point b' on the contour such that $b'_x > b_x$ and $b'_y > b_y$. Since the sweep plane has visited b' already, we must have $b'_z > b_z$. Hence, b' dominates b, which implies that $b \notin B_1'$. This is a contradiction. We have shown that when the sweep plane reaches b, the projection of this point lies outside the contour. Therefore, b is inserted into L. This proves that $B_1' \subseteq L$. $\square$

Remark 3.2 It follows from the proof that B_1' is the set of all points of B_1 that are added to the contour during Step 1. Note that once a point has been added, it may be removed again from the contour later on during the sweep in Step 1.

Lemma 3.4 *In Step 2 of the algorithm, all dominance pairs (r, b), where $r \in R_1$ and $b \in B_1'$, are reported. Moreover, only such pairs are reported.*

Proof. Suppose that (r, b) is reported. Then, $r \in R_1$ and $b \in B_1'$. Also, $r \in C_b$ when b is reached and so $r_z < b_z$. Also, by construction of C_b, b's projection dominates r's projection. Thus b dominates r.

Now let $r \in R_1$ and $b \in B_1'$ such that b dominates r. We prove that the pair (r, b) is reported. At the moment when the sweep plane reaches b, this point is removed from the contour. We have to show that r is contained in C_b at this moment.

When the sweep plane reaches r, we insert this point into the lists C_q for all points q that are on the contour at that moment and whose projection onto the xy-plane dominate r's projection. If b is one of these points, then we are done, because r stays in C_b until the sweep plane reaches b. Otherwise, b's projection lies inside the contour. Let q be the point with smallest z-coordinate that is on the contour at the moment when the sweep plane reaches r and whose projection onto the xy-plane dominates b's

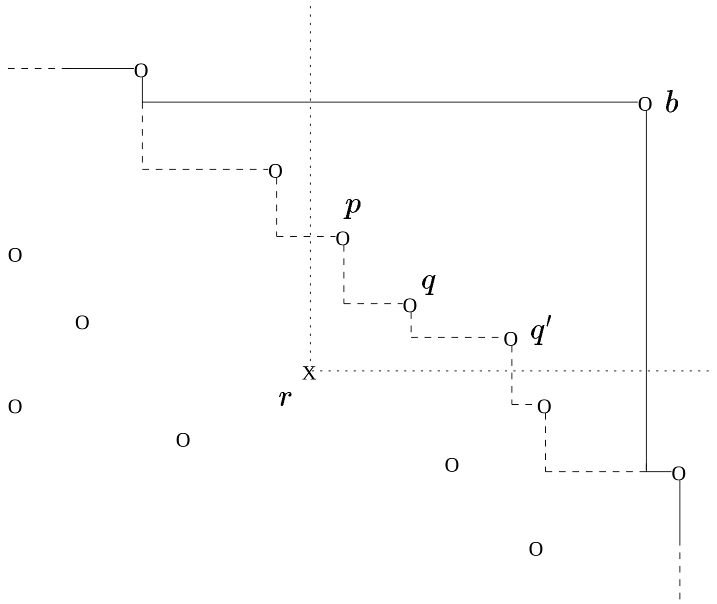

Figure 3: *Illustrating Step 2.3. Point r is inserted into the lists C_p, C_q and $C_{q'}$.*

projection. Then, r is inserted into C_q. Note that $b_z > q_z$, because otherwise b would be dominated by q, contradicting the fact that b belongs to B_1'.

When the sweep plane reaches q, point r is inserted into the lists C_p for all points p that appear on the contour at that moment and whose projections dominate r's projection. If b is one of these points, then we are done. Otherwise, let p be the point with smallest z-coordinate that is on the contour at the moment when the sweep plane reaches q and whose projection onto the xy-plane dominates b's projection. Point r is inserted into C_p. We have $b_z > p_z$. Now we consider the moment when the sweep plane reaches point p, and repeat the same argument. Continuing in this way, and observing that point b must appear on the contour, it follows that r will be inserted into C_b. $\square$

Lemma 3.5 *Let k_{RB} be the number of dominance pairs (r, b) such that $r \in R_1$ and $b \in B_1'$. Algorithm Sweep(R_1, B_1) takes $O(k_{RB} \log\log u)$ time and uses $O(u)$ space.*

Proof. Let $n = |R_1| + |B_1|$. First note that the points of R_1 and B_1 are sorted already by their third coordinates. Step 1 of the algorithm takes $O(n \log\log u)$ time. Consider Step 2. The total time for updating the contour in Step 2.1 is upper-bounded by the time for Step 1. The total time for Step 2.2 is obviously $\Theta(k_{RB})$. It remains to estimate the time for updating the C-lists in Step 2.3. Let $r \in C_b$ be a red point to be added to the C-lists of the new contour points that appear as a result of undoing the changes at b in Step 2.1. Deciding whether r's projection lies inside or outside the two-dimensional contour takes $O(\log\log u)$ time. If it lies outside, then we charge this cost to the pair (r, b) just reported in Step 2.2. The total number of such charges due to all red points is $O(k_{RB} \log\log u)$. If r lies inside the contour and if it is inserted into m C-lists (m will be at least one), then the time taken is $O(m + \log\log u) = O(m \log\log u)$. We charge $O(\log\log u)$ to each of the m instances of r thus inserted. Likewise, when we encounter a red point r in the upward sweep, if r's projection is inside the current contour, then we use a similar charging scheme. It follows that each red point r accumulates at most two charges for being inside the contour, for each dominance pair involving it that is output. If the projection of the red point is outside the current contour, then we charge the $O(\log\log u)$ cost incurred in determining this to the point itself. Since this point is never seen again during *Sweep*(R_1, B_1), the total charge thus accumulated is $O(n \log\log u)$.

Therefore the algorithm takes $O((n + k_{RB}) \log\log u)$ time. We know that $k_{RB} \geq |R_1|$. Also, since $|R_1| \geq |B_1|$, we have $n \leq 2|R_1| \leq 2k_{RB}$. This proves the bound on the running time. It is clear that the algorithm uses $O(u)$ space. (Note that the list M has size $O(n)$.) $\square$

3.3 The three-dimensional red-blue dominance algorithm

The algorithm for reporting all red-blue dominance pairs in $R \times B$ is given in Figure 4. This algorithm uses the procedures *Clean* and *Sweep* that were given in Sections 3.1 and 3.2, respectively. Also recall the mapping $\mathcal{F}$ that was defined in Section 3.1. We assume that we have constructed already the empty vEB-tree on the universe U.

Lemma 3.6 *At the end of the $(i-1)$-st iteration of the while-loop, $i \geq 1$, the set B_i (resp. R_i) is clean w.r.t. R_i (resp. B_i).*

Proof. The proof is by induction on i. For $i = 1$, the claim is true. Assume B_i is clean w.r.t. R_i and R_i is clean w.r.t. B_i. Suppose that the "if" part of the "if–then–else" statement is executed. It is clear that R_{i+1} is clean w.r.t. B_{i+1}. To prove that B_{i+1} is clean w.r.t. R_{i+1}, let $b \in B_{i+1}$. Then $b \in B_i$ and, hence, there is a point r in R_i that is dominated by b. Since $b \in B_{i+1} = H$, the procedure *Clean*(R_i, H) produces a set R_{i+1} which contains the point r. Thus each point in B_{i+1} dominates some point in R_{i+1} and hence B_{i+1} is clean w.r.t. R_{i+1}. If the "else" part is executed, then it is clear that B_{i+1} is clean w.r.t. $R_{i+1} = \mathcal{F}(H)$. Similarly, we can prove that R_{i+1} is clean w.r.t. B_{i+1}. $\square$

Lemma 3.7 *Algorithm 3Ddom(R, B) terminates and reports all dominance pairs (r, b), where $r \in R$ and $b \in B$. Moreover, if a pair (r, b) is reported, then it is a red-blue dominance pair.*

Proof. The algorithm terminates because after each iteration of the while-loop either $|B_{i+1}| = |B_i \setminus B_i'| < |B_i|$ (since $|B_i'| > 0$) or $|R_{i+1}| = |\mathcal{F}(\mathcal{F}(R_i) \setminus R_i')| < |R_i|$ (since $|R_i'| > 0$). We now prove that (r, b) is reported iff b dominates r.

Suppose that (r, b) is reported. Since a report happens only during one of the calls to *Sweep*, it follows from the correctness of this procedure (see Lemma 3.4) that b dominates r.

Conversely, suppose that b dominates r. Note that a point is discarded in algorithm *3Ddom*(R, B) either during a call to *Clean* or right after that call to *Sweep* during which it becomes a three-dimensional maximal element. Since b dominates r, it follows

Algorithm *3Ddom*(R, B)
(* R and B are sets of points in U^3; the algorithm reports all pairs (r, b) such that $r \in R$, $b \in B$ and b dominates r *)
begin
$R_1 := \mathit{Clean}(R, B)$;
$B_1 := \mathcal{F}(\mathit{Clean}(\mathcal{F}(B), \mathcal{F}(R)))$;
$i := 1$;
while $R_i \neq \emptyset$ **and** $B_i \neq \emptyset$
do if $|R_i| \geq |B_i|$
  then $\mathit{Sweep}(R_i, B_i)$;
    (* this procedure computes the set B'_i of three-dimensional maxima of B_i and reports all dominances (r, b) where $r \in R_i$ and $b \in B'_i$ *)
    $H := B_i \setminus B'_i$;
    $R_{i+1} := \mathit{Clean}(R_i, H)$;
    $B_{i+1} := H$
    (* B_{i+1} is clean w.r.t. R_{i+1}; see Lemma 3.6 *)
  else $\mathit{Sweep}(\mathcal{F}(B_i), \mathcal{F}(R_i))$;
    (* this procedure computes the set R'_i of three-dimensional maxima of $\mathcal{F}(R_i)$ and reports all dominances (r, b) where $r \in \mathcal{F}(R'_i)$ and $b \in B_i$ *)
    $H := \mathcal{F}(R_i) \setminus R'_i$;
    $B_{i+1} := \mathcal{F}(\mathit{Clean}(\mathcal{F}(B_i), H))$;
    $R_{i+1} := \mathcal{F}(H)$
    (* R_{i+1} is clean w.r.t. B_{i+1}; see Lemma 3.6 *)
  fi;
  $i := i + 1$
od
end

Figure 4: The three-dimensional red-blue dominance reporting algorithm.

that if neither r nor b has been discarded just before one of the calls to *Clean* within the while-loop then neither will be discarded during that call. (Similarly, if neither r nor b has been discarded just before the two calls to *Clean* outside the while-loop, then neither will be discarded during those two calls.) Moreover, at least one of r and b will be discarded sometime during the algorithm since the algorithm terminates. Wlog, assume that b is discarded. Then it follows that b becomes a three-dimensional maximal element before r becomes one (if ever). Let b become a three-dimensional maximal element in Step 1 of $Sweep(R_i, B_i)$ for some i. Thus, when Step 2 of $Sweep(R_i, B_i)$ commences, $r \in R_i$. By the correctness of the *Sweep* routine, (r, b) is reported as a dominance pair. □

Theorem 3.1 *Let R and B be two sets of points in U^3 that are sorted by their third coordinates. Assume we are given an empty vEB-tree on the universe U. Let $u = |U|$ and $n = |R| + |B|$, and let k' be the number of dominances (r, b), where $r \in R$ and $b \in B$. Assume that $n \leq u$. In $O((n + k') \log \log u)$ time and using $O(u)$ space, we can find all these dominance pairs.*

Proof. Let $n_i = |R_i| + |B_i|$ and let k_i be the number of dominance pairs that are reported during the i-th iteration. Because of the cleaning step and because we distinguish between the cases where $|R_i| \geq |B_i|$ and $|R_i| < |B_i|$, we have $n_i \leq 2k_i$. Also, since during each iteration, we output different dominance pairs, we have $\sum_i k_i = k'$. The initial cleaning of R and B takes $O(n \log \log u)$ time. By Lemmas 3.2 and 3.5, the i-th iteration takes time $O((n_i + k_i) \log \log u)$, which is bounded by $O(k_i \log \log u)$. It follows that the entire algorithm takes time

$$O(n \log \log u + \sum_i k_i \log \log u) = O((n + k') \log \log u).$$

The algorithm uses space $O(n + u)$, which is bounded by $O(u)$. □

3.4 Analysis of the four-dimensional dominance reporting algorithm

Consider again our divide-and-conquer algorithm of Section 2.2 for solving the four-dimensional dominance reporting problem on the normalized set $S \subseteq U^4$. We implement Step 3—the merge step—using algorithm *3Ddom*. Assume we have constructed already the empty vEB-tree on the universe U.

Let $T(n, u)$ denote the total running time on a set of n points in U^4, that are sorted by their third coordinates. Recall that it is assumed that $n = u$ (however, the sizes of the sets in the recursive calls will be smaller than u). We do not include in $T(n, u)$ the time that is charged to the output.

Step 1 of the algorithm takes $O(n)$ time, and Step 2 takes $2T(n/2, u)$ time. By Theorem 3.1, Step 3—except for the reporting—takes $O(n \log \log u)$ time. Hence, $T(n, u) = O(n \log \log u) + 2T(n/2, u)$, which solves to $T(n, u) = O(n \log n \log \log u)$. For each dominance pair, we spend an additional amount of $O(\log \log u)$ time. Since each such pair is reported exactly once, the total running time of the divide-and-conquer algorithm is bounded by $O(n \log n \log \log u + k \log \log u)$, where k denotes the number of dominance pairs in S. Moreover, the algorithm uses $O(u)$ space.

Our original problem was to solve the dominance reporting problem on a set V of n points in $\mathbb{R}^4$. In $O(n \log n)$ time, we normalize the points, giving a set S of n points in $U^4 = \{0, 1, \ldots, n-1\}^4$. Then, in $O(n)$ time, we construct an empty vEB-tree on the universe U. Finally, in $T(n,n) + O(k \log\log n)$ time we find all k dominance pairs in S. This gives all k dominance pairs in V. The entire algorithm takes $O(n \log n \log\log n + k \log\log n)$ time and it uses $O(n)$ space. This proves our main result:

Theorem 3.2

1. *Let V be a set of n points in $\mathbb{R}^4$ and let k be the number of dominance pairs in V. In $O(n \log n \log\log n + k \log\log n)$ time and using $O(n)$ space, we can find all these dominance pairs.*

2. *Let $\mathcal{R}$ be a set of n axes-parallel rectangles in $\mathbb{R}^2$ and let k be the number of pairs of rectangles (R', R) such that R encloses R'. In $O(n \log n \log\log n + k \log\log n)$ time and using $O(n)$ space, we can find all these pairs of rectangles.*

4 A faster algorithm for a special case

Let $\mathcal{R}$ be the set of n axes-parallel rectangles. Let the *aspect ratio* of a rectangle be its height divided by its width. Assume that there are only α different aspect ratios in $\mathcal{R}$, where α is a constant. This is a reasonable assumption in practice and it yields a simple and more efficient solution than Theorem 3.2, part 2. Specifically, we give an $O(\alpha n \log n + k)$-time and $O(n)$-space sweepline algorithm to enumerate the k enclosing pairs of rectangles.

By a *diagonal* of a rectangle we mean the line-segment joining its SW and NE corners. Clearly, there are α different slopes among the diagonals in $\mathcal{R}$. For some such slope ρ, let $\mathcal{R}' \subseteq \mathcal{R}$ consist of the rectangles whose diagonals have slope ρ. Let $R = [l, r] \times [b, t]$ and $R' = [l', r'] \times [b', t']$ be rectangles in $\mathcal{R}$ and $\mathcal{R}'$, respectively. (Throughout, we view rectangle sides as closed line segments, i.e., endpoints are included.) The following lemma is shown easily:

Lemma 4.1 *Let L be a line with slope ρ which moves over the plane from the northwest to the southeast. Consider the moment at which L coincides with the diagonal of R'. If L intersects R, then one of the following holds:*

1. *L meets the left and top sides of R. In this case, we have $R' \subseteq R$ iff $l' \geq l$ and $t' \leq t$.*

2. *L meets the left and right sides of R. In this case, we have $R' \subseteq R$ iff $l' \geq l$ and $r' \leq r$.*

3. *L meets the bottom and top sides of R. In this case, we have $R' \subseteq R$ iff $b' \geq b$ and $t' \leq t$.*

4. *L meets the bottom and right sides of R. In this case, we have $R' \subseteq R$ iff $b' \geq b$ and $r' \leq r$.*

Note that L meets the corners of R in a specific order, namely, NW, NE, SW, SE (resp. NW, SW, NE, SE), depending on whether R's diagonal has slope less (resp. greater) than ρ. The NE and SW corners will be met simultaneously if R's diagonal has slope ρ; this case is covered by Lemma 4.1 since rectangle sides are closed line segments.

4.1 The algorithm

For each diagonal-slope ρ we do the following:

We project all the rectangle corners in $\mathcal{R}$ onto a line $\hat{L}$ normal to L and sort them in non-decreasing order (ties are broken arbitrarily). Note that the SW and NE corners of each rectangle in $\mathcal{R}'$ projects to the same point on $\hat{L}$. We treat these two points as a composite point.

Using L, we sweep over $\hat{L}$ from $-\infty$ to $+\infty$, maintaining four balanced priority search trees, PST_i, $1 \leq i \leq 4$. (PST_i will handle condition i of Lemma 4.1.) Let v be the current event point. The following actions are taken:

1. v corresponds to the NW corner of $R = [l, r] \times [b, t]$. We insert (l, t) into PST_1.

2. v corresponds to the NE corner of $R = [l, r] \times [b, t]$. If the SW corner of R has not been seen so far then we delete (l, t) from PST_1 and insert (l, r) into PST_2. Otherwise, we delete (b, t) from PST_3 and insert (b, r) into PST_4.

3. v corresponds to the SW corner of $R = [l, r] \times [b, t]$. If the NE corner of R has not been seen so far then we delete (l, t) from PST_1 and insert (b, t) into PST_3. Otherwise, we delete (l, r) from PST_2 and insert (b, r) into PST_4.

4. v corresponds to the SE corner of $R = [l, r] \times [b, t]$. We delete (b, r) from PST_4.

5. v corresponds to the SW and NE corner of $R' = [l', r'] \times [b', t'] \in S'$. We query PST_1 with (l', t') and report all points (l, t) in it such that $l' \geq l$ and $t' \leq t$. Similarly, we query PST_2 with (l', r'), PST_3 with (b', t'), and PST_4 with (b', r'). Then we delete (l', t') from PST_1 and insert (b', r') into PST_4.

Theorem 4.1 *Given a set $\mathcal{R}$ of n axes-parallel rectangles in $\mathbb{R}^2$ with at most α different aspect ratios, where α is a constant, all k pairs of rectangles (R', R) such that R encloses R' can be reported in $O(\alpha n \log n + k)$ time and $O(n)$ space.*

Proof. When L coincides with the diagonal of R' (Case 5), then one of the cases of Lemma 4.1 holds w.r.t. L and R. Therefore, one of the queries done in Case 5 will discover the pair (R', R).

For each slope ρ, there is one sort, followed by $O(n)$ queries and updates on PSTs of size $O(n)$. Hence the total time is $O(n \log n + k_\rho)$, if k_ρ pairs are output. The claimed time bound follows. The space used is $O(n)$ per sweep and this can be re-used. $\square$

5 Concluding remarks

We have given an algorithm for solving the rectangle enclosure reporting problem, or, equivalently, the four-dimensional dominance reporting problem, that runs in $O(n \log n \log\log n + k \log\log n)$ time, where k is the number of reported pairs. Previously, the problem had been solved in $O(n \log^2 n + k)$ time by Lee and Preparata [LP82].

We leave open the question of whether the problem can be solved in $O(n \log n + k)$ time. It seems very difficult to remove the $\log\log n$ term that occurs in the "reporting" part of our running time.

We have given a new technique to solve the three-dimensional red-blue dominance reporting problem. Using the same approach we can solve the two-dimensional version of this problem, where the red and blue points are sorted by their x-coordinates, optimally, i.e., in $O(n + k)$ time.

References

[BW80] J.L. Bentley and D. Wood. An optimal worst-case algorithm for reporting intersections of rectangles. *IEEE Transactions on Computers*, 29:571–577, 1980.

[BST93] V. Bistiolas, D. Sofotassios and A. Tsakalidis. Computing rectangle enclosures. *Computational Geometry: Theory & Applications*, 2: 303–308, 1993.

[EO82] H. Edelsbrunner and M.H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters*, 14:124–127, 1982.

[KO88] R.G. Karlsson and M.H. Overmars. Scanline algorithms on a grid. *BIT*, 28:227–241, 1988.

[LP82] D.T. Lee and F.P. Preparata. An improved algorithm for the rectangle enclosure problem. *Journal of Algorithms*, 3:218–224, 1982.

[McC85] E.M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14:257–276, 1985.

[PS88] F.P. Preparata and M.I. Shamos. *Computational Geometry – An Introduction.* Springer–Verlag, Berlin, 1988.

[vEB77a] P. van Emde Boas, R. Kaas and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.

[vEB77b] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6:80–82, 1977.