International Journal of Computational Geometry & Applications

THE RECTANGLE ENCLOSURE AND POINT-DOMINANCE PROBLEMS REVISITED

PROSENJIT GUPTA and RAVI JANARDAN
Department of Computer Science
University of Minnesota, Minneapolis, MN 55455, U.S.A.
e-mail: {pgupta,janardan}@cs.umn.edu

and

MICHIEL SMID
Max-Planck-Institut für Informatik
D-66123 Saarbrücken, Germany
e-mail: michiel@mpi-sb.mpg.de

and

BHASKAR DASGUPTA
Department of Computer Science
University of Waterloo, Waterloo, Ontario N2L 3G1, Canada
e-mail: bdasgupt@daisy.uwaterloo.ca

Received 10 August 1994
Revised 10 August 1995
Communicated by M. T. Goodrich

ABSTRACT

We consider the problem of reporting the pairwise enclosures in 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 $\mathbb{R}^4$. Over a decade ago, Lee and Preparata[6] gave an $O(n \log^2 n + k)$–time and $O(n)$–space algorithm for these problems, where k is the number of reported pairs. Since that time, the question of whether there is a faster algorithm has remained an intriguing open problem.

In this paper, we give an algorithm which uses $O(n)$ space and runs in $O(n \log n \log\log n + k \log\log n)$ time. Thus, although our result is not a strict improvement over the Lee–Preparata algorithm for the full range of k, it is, nevertheless, the first result since Ref. (6) to make *any* progress on this long–standing open problem. Our algorithm is based on the divide–and–conquer paradigm. The heart of the algorithm is the solution to a red–blue dominance reporting problem (the "merge" step). We give a novel solution for this problem which is based on the iterative application of a sequence of non–trivial sweep routines. This solution technique should be of independent interest.

We also present another algorithm whose bounds match the bounds given in Ref. (6), but which is simpler. Finally, we consider the special case where the rectangles have a small number, α, of different aspect ratios, which is often the case in practice. For this problem, we give an algorithm which runs in $O(\alpha n \log n + k)$ time and uses $O(n)$ space.

1. Introduction

Problems involving sets of rectangles have been studied widely in computational geometry since they are central to many diverse applications, including VLSI layout design, image processing, computer graphics, and databases. (See, for instance, Chapter 8 in each of the books[9,10].) For most of these problems, efficient (indeed, optimal) algorithms are known. In this paper, we investigate the following rectangle problem, whose complexity has not yet been resolved satisfactorily.

Problem 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'.*

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 (q, p) a *dominance pair*. Using this terminology, Problem 1 is transformed—in linear time—into the following one:

Problem 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[3] implies that Problems 1 and 2 are equivalent, i.e., in linear time, Problem 2 can also be transformed into Problem 1.

Here is a brief history of the problem. 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. The rectangle problem was first considered by Vaishnavi and Wood[14], who gave an $O(n \log^2 n + k)$–time and $O(n \log^2 n)$–space algorithm. This result was also obtained independently by Lee and Wong[7]. In 1982, Lee and Preparata[6] gave an algorithm which ran in $O(n \log^2 n + k)$ time and used only $O(n)$ space. Ever since, the question of whether there is a faster algorithm has remained intriguingly open, see page 371 of Ref. (10). (See also Bentley and Wood[1] and Bistiolas et al.[2])

1.1. Summary of contributions

Our main result is an algorithm for Problems 1 and 2 which uses $O(n)$ space and runs in $O(n \log n \log\log n + k \log\log n)$ time. While our result is not a strict improvement over Ref. (6) for the full range of k (it is an improvement for $k = o(n \log^2 n / \log\log n)$), it is, nevertheless, the first result since Ref. (6) to make *any* progress on this long-standing open problem, and we hope that our approach will spur further research on finally putting this problem to rest.

Our approach is to transform Problem 2 to a grid and then apply divide–and–conquer. The heart of the algorithm is the solution to a red–blue dominance reporting problem (the "merge" step). We give a novel solution to this problem which is based on the iterative application of a sequence of non–trivial sweep routines. We regard this solution technique as the second contribution of the paper since it could find applications in other grid–based problems.

We also present an alternative algorithm whose bounds match the bounds in Ref. (6), but which is simpler. In particular, our algorithm employs just one level of divide–and–conquer (as opposed to two levels in Ref. (6)) and uses simple data structures.

Finally, we consider the special case, where the rectangles have only α different aspect ratios, for some integer α, where the *aspect ratio* of a rectangle is its height divided by its width. For this problem, we give an algorithm which runs in $O(\alpha n \log n + k)$ time and uses $O(n)$ space. Thus this algorithm is faster than the $O(n \log^2 n + k)$ algorithm when α is small, i.e., when $\alpha = o(\log n)$. To our knowledge, no results were known previously for this special case. In VLSI design, the input objects are often rectangular, with aspect ratios lying in a bounded range, see page 182 of Ref. (11); we expect that our algorithm will be useful in this context.

1.2. Overview of the main result

Throughout the paper (except in Section 4) we consider Problem 2. Our first observation is that we can afford to (in $O(n \log n)$ time) normalize the problem to a grid. This allows us to bring into play efficient structures such as van Emde Boas trees[12,13]. Specifically, we map the n points in V to a set S of n points in U^4, where $U = \{0, 1, \ldots, n-1\}$, such that dominance pairs in V are in one–to–one correspondence with dominance pairs in S. We divide S along the fourth coordinate into two equal halves and recurse on these sets. In the merge step, we (effectively) have a set of red and blue points in U^3 and need to report all red–blue dominances. We solve this problem by an iterative sequence of sweeps, as follows:

We first "clean" the red set so as to remove those red points that are not dominated by any blue points. We also "clean" the blue set to eliminate those blue points that do not dominate any red point. Intuitively, this *cleaning step* gets rid of points that do not contribute to any dominance pair and this allows us to bound the running time of the next step—the reporting step.

In the *reporting step*, we report all red–blue dominances in the cleaned sets. Assume, wlog, that there are more red points than blue points in the cleaned sets. To do the reporting correctly and efficiently, we do not consider the blue points all at once. Instead, we report red–blue dominances involving only those blue points that are maximal (in three dimensions) in the blue set. We find these maximal points by a single sweep. Then we sweep in the opposite direction and incrementally reconstruct the blue contour using information computed in the first sweep. During this second sweep, we report red–blue dominances involving blue maximal points. In both the reporting step and the cleaning step, we need to dynamically maintain certain two–dimensional contours of maximal points. For this we use the van Emde Boas trees mentioned earlier.

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.

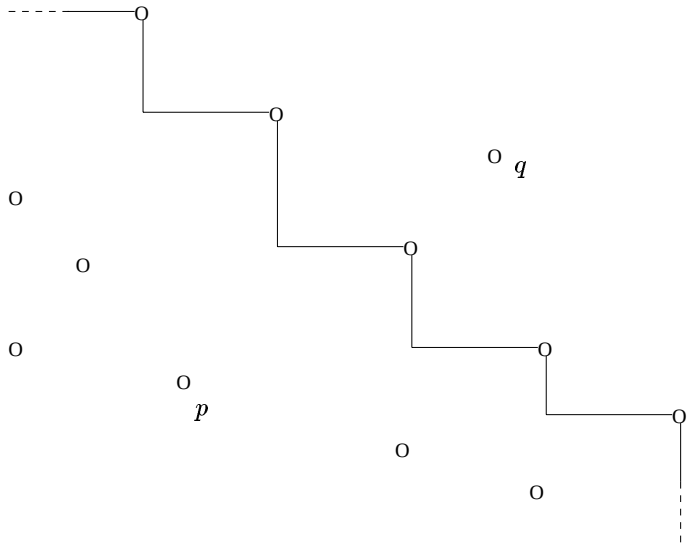

Figure 1: *The maximal layer of a planar point set S forms a contour. The point p is inside the contour, whereas the point q is outside. Note that q does not belong to S.*

1.3. Preliminaries

In the rest of this paper, we will mainly consider Problem 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.

Our algorithm will use the following data structures. First, we use the *priority search tree (PST)*, see McCreight[8]. This data structure stores a planar point set S, such that for any unbounded query rectangle of the form $R = [l_x, r_x] \times [b_y, \infty)$, we can in $O(\log n + k)$ time report all points of S that are in R. Here, n (resp. k) denotes the size of S (resp. the number of reported points). Points can be inserted and deleted in the PST in $O(\log n)$ time.

In case the coordinates of all points and query objects come from a finite universe of size u, we can use a *radix priority search tree*, which is a simpler structure not requiring any rebalancing during updates. This data structure has size $O(n+u)$ and query (resp. update) operations take $O(\log n + k)$ (resp. $O(\log u)$) time. Moreover,

it can be built in $O(n+u)$ time. (See Ref. (8) for details.) We remark here that even if the set S is empty it takes $O(u)$ time to build the radix PST. In this case, we will talk about the *empty radix PST*. (Note that although S is empty, it still uses $O(u)$ space.)

Let S be a set of integers in the range $U = \{0, 1, \ldots, u-1\}$. A *van Emde Boas tree (vEB-tree)*[12,13] storing S has size $O(u)$ and can be built in $O(u)$ time. Given this tree, we can search for an element of U in $O(\log\log u)$ time. Also, elements from U can be inserted and deleted in the vEB-tree in $O(\log\log u)$ time. Again, if the set S is empty, we will talk about the *empty vEB-tree*.

Later in our algorithms, the universe will be known in advance. Therefore, at the start, we will build an empty radix PST or vEB-tree once and for all. During certain stages of the algorithm, we will delete all points from the data structure, yielding an empty structure, which we then use in the next stage. In this way, we only pay the $O(u)$ preprocessing time once.

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

Let V be the set of points $\{p = (p_1, p_2, p_3, p_4)\}$. We sort the points of V lexicographically in increasing order. Then, we replace the first coordinate p_1 of each point p by its rank p_1' in this ordering. Next, we sort the vectors in the set

$$\{(p_2, p_1', p_3, p_4) : (p_1, p_2, p_3, p_4) \in V\}$$

lexicographically, and replace the second coordinate p_2 of each point p of V by its rank p_2'. We repeat this for the third and fourth coordinates: Sort the vectors

$$\{(p_3, p_1', p_2', p_4) : (p_1, p_2, p_3, p_4) \in V\},$$

yielding normalized coordinates p_3', and finally sort

$$\{(p_4, p_1', p_2', p_3') : (p_1, p_2, p_3, p_4) \in V\},$$

yielding normalized coordinates p_4'. This whole procedure gives a set of points $\{(p_1', p_2', p_3', p_4') : (p_1, p_2, p_3, p_4) \in V\}$, which we denote by S. This normalization procedure extends a method given in Ref. (4). The following lemma can easily be proved.

Lemma 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 (q, p) is a dominance pair in V iff the corresponding pair in S is also a dominance pair. Moreover, for each i, $1 \le i \le 4$, no two points of S have the same i-th coordinate.*

Note that because of the last claim of the lemma, the set S is in "general" position. This makes the implementation of the algorithms much easier.

2.2. The algorithm

Let S be the set of n points from Lemma 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 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 PST, see Section 1.3, 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'

denotes the number of red-blue dominance pairs in $R \times B$, then the algorithm takes time $O(n \log u + k')$.

Now we 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 $T(n, u) = 2\,T(n/2, u) + O(n \log u)$, 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 Refs. (6) and (10). Because of the normalization step, we can use a radix PST T. Immediately after the normalization step, we build this data structure 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 vEB-tree on the universe U. During the entire algorithm, elements will be inserted and deleted in this tree and we will perform queries on it. 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 Ref. (5), Karlsson and Overmars give an $O(n \log\log u)$ time and $O(u)$ space

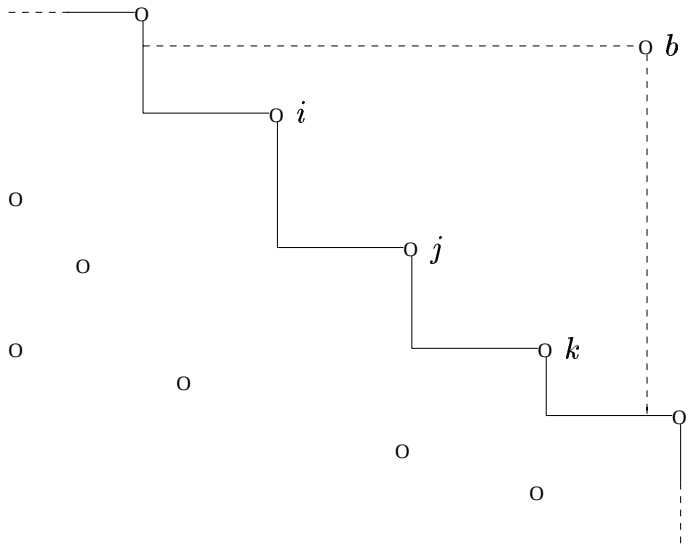

Figure 2: *Updating the blue contour when the sweep plane visits the point b. Points i, j and k are deleted and point b is inserted.*

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 2 *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 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 Section 1.3.) □

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 *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*

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

and

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

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

Remark 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. The amount of work in the sweep and report step will be related to $|R_1| + |B_1|$. If we want to charge this to the number of pairs reported, we must report at least $\max(|R_1|, |B_1|)$ pairs in this step. Therefore, the exact form of the sweep algorithm depends on the fact whether $|R_1| \geq |B_1|$ or not.

For the purpose of the discussion, let us assume w.l.o.g. that $|R_1| \geq |B_1|$. Let B_1' denote the three-dimensional maxima of B_1. The procedure $Sweep(R_1, B_1, 0)$, which will be described in this section, finds all red-blue dominance pairs (r, b), where $r \in R_1$ and $b \in B_1'$. The third parameter, i.e., 0, indicates that the pairs found should also be *reported* as (r, b). 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 $Sweep(\mathcal{F}(B_1), \mathcal{F}(R_1), 1)$. This procedure finds dominances $(\mathcal{F}(b), \mathcal{F}(r))$. The third parameter—which is 1 in this case—indicates that these pairs should be *reported* as pairs (r, b).)

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 4 below. Also, we will see that 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 2-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 4 *At the end of Step 1, the list L contains the set B_1' of three-dimensional maxima of B_1.*

Proof. 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 proves that $L \subseteq B_1'$.

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 $B_1' \subseteq L$. □

Remark 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 5 *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. Let $b_1, b_2, \ldots, b_s = b$ be points in B_1' such that (i) b_{i-1}'s projection on the sweep plane dominates b_i's projection, for $i = 2, 3, \ldots, s$, (ii) b_1's projection dominates r's projection, and (iii) there is no point $b' \in B_1'$ such that b_1's projection dominates the projection of b' and the projection of b' dominates r's projection. We prove by induction on s that $r \in C_{b_s}$ at the instant the sweep plane reaches b_s. This will prove that the pair $(r, b_s) = (r, b)$ is reported.

Consider the base case, $s = 1$. When the sweep plane reaches r, we insert r 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. Since $b_s = b$ is one of these points, we are done, because r stays in C_b until the sweep plane reaches b_s.

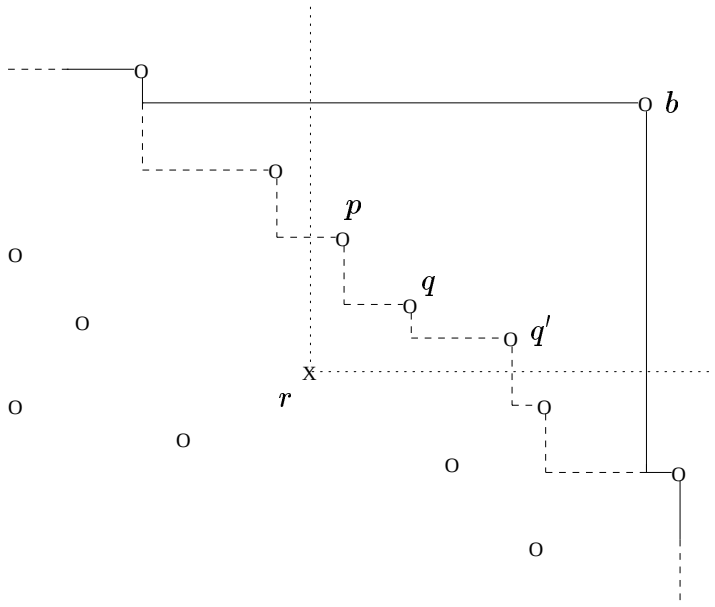

Figure 3: *Illustrating Step 2.3. Point r is inserted into the lists C_p, C_q and $C_{q'}$.*

Inductively, assume that $s > 1$ and that the claim holds for b_{s-1}. When the sweep plane reaches b_s subsequently, b_{s-1} is removed from the contour and b_s is included in the contour. Therefore, the points in $C_{b_{s-1}}$ whose projections are dominated by b_s's projection are inserted into the list C_{b_s}. Since $r \in C_{b_{s-1}}$ (by the induction hypothesis) and r is dominated by b_s (by assumption), it follows that $r \in C_{b_s}$. □

Lemma 6 *Let k_{RB} be the number of dominance pairs (r, b) such that $r \in R_1$ and $b \in B_1'$. Algorithm Sweep$(R_1, B_1, 0)$ 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 $O(\log\log u)$

```
Algorithm 3Ddom(R, B)
(* R and B are sets of points in U^3; the algorithm reports all pairs
   (r, b) such that r ∈ R, b ∈ B and b dominates r *)
begin
R_1 := Clean(R, B);
B_1 := F(Clean(F(B), F(R)));
i := 1;
while R_i ≠ ∅ and B_i ≠ ∅
do if |R_i| ≥ |B_i|
    then Sweep(R_i, B_i, 0);
         (* this procedure computes the set B'_i of three-dimensional
            maxima of B_i and reports all dominances (r, b)
            where r ∈ R_i and b ∈ B'_i *)
         H := B_i \ B'_i;
         R_{i+1} := Clean(R_i, H);
         B_{i+1} := H
         (* B_{i+1} is clean w.r.t. R_{i+1}; see Lemma 7 *)
    else Sweep(F(B_i), F(R_i), 1);
         (* this procedure computes the set R'_i of three-dimensional
            maxima of F(R_i) and reports all dominances (r, b)
            where r ∈ F(R'_i) and b ∈ B_i *)
         H := F(R_i) \ R'_i;
         B_{i+1} := F(Clean(F(B_i), H));
         R_{i+1} := F(H)
         (* R_{i+1} is clean w.r.t. B_{i+1}; see Lemma 7 *)
    fi;
    i := i + 1
od
end
```

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

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 $Sweep(R_1, B_1, 0)$, 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 $O(n)$.) □

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 7 *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}. □

Lemma 8 *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 5) 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 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 $Sweep(R_i, B_i, 0)$ for some i. Thus, when Step 2 of $Sweep(R_i, B_i, 0)$ commences, $r \in R_i$. By the correctness of the *Sweep* routine, (r, b) is reported as a dominance pair. □

Theorem 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 and 6, 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 $2\,T(n/2, u)$ time. By Theorem 1, Step 3—except for the reporting—takes time $O(n \log\log u)$. Hence, $T(n, u) = O(n \log\log u) + 2\,T(n/2, u)$, 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 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

Assume that there are only α different aspect ratios in the set $\mathcal{R}$ 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.)

Lemma 9 *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 $r' \leq r$.*

Proof. Assume that L intersects R. Since L moves from the northwest to the southeast, it is clear that its intersections with R must be at one of the four pairs of sides stated in Cases (1)–(4) of the lemma. (If the diagonal of R has slope ρ, then its left and bottom sides and its top and right sides will be intersected simultaneously—at the SW and NE corners of R. This case is covered by one of the four cases above since rectangle sides are closed line segments.)

Assume that Case (1) holds, i.e., L intersects the left and top sides of R. We show that $R' \subseteq R \Leftrightarrow l' \geq l$ and $t' \leq t$.

($\Rightarrow$) : Let $R' \subseteq R$. Then, by definition, $l' \geq l$, $t' \leq t$, $r' \leq r$, and $b' \geq b$.

($\Leftarrow$) : Let $l' \geq l$ and $t' \leq t$. Let R'' be the rectangle whose SW and NE corners are the intersection of L with the left and top sides of R, respectively. Note that (i) $R'' \subseteq R$, (ii) the left (resp. top) side of R'' is contained in the left (resp. top) side of R, and (iii) the diagonals of R'' and R' are both contained in L. Facts (ii) and (iii) together with the assumption $l' \geq l$ and $t' \leq t$ imply that $R' \subseteq R''$. Fact (i) then implies that $R' \subseteq R$.

The proofs for Cases (2)–(4) are similar and hence omitted. □

How does Lemma 9 help us? Note that it specifies the inclusion of R' in R as one of four two-dimensional range restrictions. Which one of these conditions applies depends on the relative positions of R and R'—thus at the time instant in question, a different condition may hold for each rectangle $R \in \mathcal{R}$ that L intersects. This suggests that we store information about the rectangles R satisfying each of the four conditions in a separate priority search tree.

Note that during the sweep, L meets the corners of each 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 the diagonal of R has slope ρ; as mentioned above, this case is covered by Lemma 9.) Therefore, we can update the four PST's in a coordinated manner during the sweep. Details are given below.

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 9.) 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 3 *Given a set $\mathcal{R}$ of n axes-parallel rectangles in $\mathbb{R}^2$ with at most α different aspect ratios, 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 9 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 *PST*s 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. □

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[6].

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 presented a new technique for solving 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.

Acknowledgements

The research of the first two 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. The third author was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

The authors would like to thank the referees for several suggestions that helped improve the presentation.

References

1. J.L. Bentley and D. Wood, "An optimal worst-case algorithm for reporting intersections of rectangles", IEEE Transactions on Computers **29** (1980) 571–577.
2. V. Bistiolas, D. Sofotassios and A. Tsakalidis, "Computing rectangle enclosures", Computational Geometry: Theory & Applications **2** (1993) 303–308.
3. H. Edelsbrunner and M.H. Overmars, "On the equivalence of some rectangle problems", Information Processing Letters **14** (1982) 124–127.
4. H.N. Gabow, J.L. Bentley and R.E. Tarjan, "Scaling and related techniques for geometry problems", Proc. 16th Annu. ACM Sympos. Theory Comput. 1984, pp. 135–143.
5. R.G. Karlsson and M.H. Overmars, "Scanline algorithms on a grid", BIT **28** (1988) 227–241.
6. D.T. Lee and F.P. Preparata, "An improved algorithm for the rectangle enclosure problem", Journal of Algorithms", **3** (1982) 218–224.
7. D.T. Lee and C.K. Wong, "Finding intersection of rectangles by range search", Journal of Algorithms **2** (1981) 337–347.
8. E.M. McCreight, "Priority search trees", SIAM Journal on Computing **14** (1985) 257–276.
9. B. Preas and M. Lorenzetti, *Physical Design Automation of VLSI Systems* (Benjamin/Cummings, Menlo Park, CA, 1988).
10. F.P. Preparata and M.I. Shamos, *Computational Geometry – An Introduction* (Springer–Verlag, Berlin, 1988).

11. N. Sherwani, *Algorithms for VLSI Physical Design Automation* (Kluwer Academic Publishers, 1993).

12. P. van Emde Boas, R. Kaas and E. Zijlstra, "Design and implementation of an efficient priority queue", Mathematical Systems Theory **10** (1977) 99–127.

13. P. van Emde Boas, "Preserving order in a forest in less than logarithmic time and linear space", Information Processing Letters **6** (1977) 80–82.

14. V. Vaishnavi and D. Wood, "Data structures for the rectangle containment and enclosure problems", Computer Graphics and Image Processing **13** (1980) 372–384.