Dominance in the presence of obstacles

Mark T. de Berg and Mark H. Overmars

RUU-CS-88-10
March 1988

Rijksuniversiteit Utrecht

Vakgroep informatica

Budapestlaan 6 3584 CD Utrecht
Corr. adres: Postbus 80.012 3508 TA Utrecht
Telefoon 030-53 1454
The Netherlands

Dominance in the presence of obstacles

Mark T. de Berg and Mark H. Overmars

Technical Report RUU-CS-88-10
March 1988

Department of Computer Science
University of Utrecht
P.O.Box 80.012
3508 TA Utrecht
the Netherlands

Dominance in the presence of obstacles

Mark T. de Berg and Mark H. Overmars

March 1988

Abstract

Given two points p and q and a set of point O in the plane, p is said to dominate q with respect to O if p dominates q and there is no $o \in O$ such that p dominates o and o dominates q. In other words, O is a set of obstacles that might block the "rectangular view" from p to q. Given a set P of points and a set O of obstacle points we are interested in determining all pairs $(p,q) \in P \times P$ such that p dominates q with respect to O. This generalizes notions of direct dominance and rectangular visibility that have been studied before. An algorithm is presented that solves the problem in time $O(n \log n \log\log n + k)$, where n is the size of $P \cup O$ and k is the number of dominance pairs found. When $P \subseteq O$ a slight change in the algorithm reduces the time bound to $O(n \log n + k)$ which is optimal.

The problem is generalized by letting O exist of arbitrary objects. A reduction is presented that reduces the general case to the case in which O consists of points only. For many classes of obstacles this reduction can be carried out in $O(n \log n)$ time.

1 Introduction

The *dominance problem* in a set of points P asks for all pairs $(p,q) \in P \times P$, such that p dominates q, denoted as $p \succ q$. (A point $p = (p_1, \ldots, p_d)$ is said to *dominate* a point $q = (q_1, \ldots, q_d)$ if $p_i \geq q_i$ for all $1 \leq i \leq d$ and $p \neq q$.) In 2-dimensional space the dominance problem can easily be solved in time $O(n \log n + k)$, where $n = |P|$ and k is the number of reported pairs, using a scanline algorithm.

Since $\succ$ is a transitive relation a large number of the dominance pairs might be redundant, i.e. they might be computable from other dominance pairs. To this end the notion of *direct dominance* has been introduced. For a pair of points (p,q) we say that p *directly dominates* q, denoted as $p \succ\!\!\!\cdot\; q$, if $p \succ q$ and there exists no point $r \in P$ such that $p \succ r$ and $r \succ q$. Güting et al.[3] have given an algorithm that computes the pairs of directly dominating points in a planar set in time $O(n \log n + k)$ where k is the number of such pairs.

The direct dominance problem in the plane can also be viewed as follows: p dominates q if there exists an axis-parallel rectangle with p as top-right vertex and q

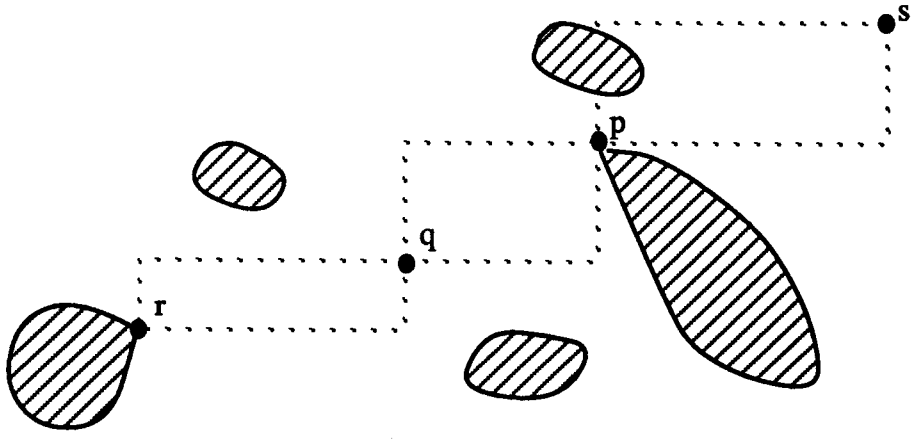


Figure 1: p dominates q and q dominates r but s does not dominate p with respect to the obstacles.

as bottom-left vertex. p directly dominates q iff this rectangle does not contain any other point of P. Overmars and Wood[8] (see also [6]) consider this as a special type of visibility, called *rectangular visibility* where the points are obstacles that might block the rectangular view from p to q. They obtain the same time bounds in a different way and consider also the query version of the problem.

In this paper we generalize this notion in the following way: we are not only given a set of points P, but also a set of objects, which we will call obstacles.

Definition 1.1 *Let P be a set of points, let O be a set of obstacles in d-dimensional space, and let $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ be two points in P. p is said to* dominate q with respect to O, *denoted as $p \succ_O q$, iff $p \succ q$ and there is no point r on any obstacle $o \in O$ such that $p \succ r$ and $r \succ q$.*

In other words, the rectangle with p and q as opposite vertices does not contain any part of any obstacle in O. (Note that it can contain other points from P.) See figure 1 for some examples. A pair $(p, q) \in P \times P$ such that $p \succ_O q$ is called a *dominance pair in P with respect to O*, or a *dominance pair* for short.

The problem we will tackle is the following: Given a set P of points and a set O of obstacles, determine all dominance pairs in P with respect to O. Note that the dominance problem is a special case by taking O to be empty. The direct dominance problem can be obtained by taking $O = P$, in other words, $p \succ q$ is equivalent to $p \succ_P q$.

In the sequel of this paper we will restrict our attention to the case where $d = 2$. From now on n will denote $|P| + |O|$ and k the number of dominance pairs in P with respect to O.

The paper is organized as follows.

In section 2 we present an algorithm that solves the dominance problem in the case O is a planar set of points. The method is an extension of [3] and uses a combination of divide-and-conquer and plane-sweep. It runs in time $O(n \log^2 n + k)$.

Using a normalization technique (see, e.g., Karlsson and Overmars[5]) the result is improved to $O(n \log n \log\log n + k)$ in section 3.

When the number of obstacles points n_o and the number of points in P n_p differ considerably the result can be improved. In section 4 it is shown that the problem can be solved in time $O(n \log n_p \log\log n_p + k)$ when n_p is small and in time $O(n \log n + n \log n_o \log\log n_o + k)$ when n_o is small.

In section 5 it is shown that in the case $P \subseteq O$ the result can be improved even further to obtain an optimal $O(n \log n + k)$ method. This immediately implies the known bounds for direct dominance and rectangular visibility in [3,8].

In section 6 we generalize the solution to the situation in which the obstacle set O contains obstacles of arbitrary size and shape. It will be shown that, provided that the obstacles satisfy some simple constraints and P is known, O can be reduced to a set O' of points only of size $O(n)$, such that for any $p, q \in P$ $p \succ_O q$ if and only if $p \succ_{O'} q$.

Finally, in section 7, we briefly review our results and indicate some directions for further research.

2 Points as obstacles.

Given two sets $P = \{p_1, \ldots, p_n\}$ and $O = \{o_1, \ldots, o_m\}$ of points in the plane, we are interested in computing all dominance pairs in P with respect to O.

To this end we will develop a divide-and-conquer method. Let V be the set of different x-coordinates of the points in $P \cup O$. If $|V| = 1$, the problem becomes 1-dimensional and is easy to solve: we sort the points in P and O according to y-coordinate and report all pairs of points in P that have no point of O between them in this sorted list. In this way all dominance pairs are reported in time $O(n \log n + k)$.

When $|V| > 1$, let $x_{mid} \notin V$ be such that the number of values in V that are $< x_{mid}$ and the number that is $> x_{mid}$ differ by at most 1. Let l be the vertical line with x-coordinate x_{mid}. l divides the plane in two halves and, hence, splits P and O in two halves. It is obvious that whether or not two points in one half of the plane form a dominance pair cannot be influenced by an obstacle point from the other half. So we can recursively treat the two halves in the same way. After this it remains to compute the dominance pairs between the points in different halves. This merge step will be described below. We thus arrive at the following algorithm:

1. Let V be the set of different x-coordinates in $P \cup O$ and let n' be the size of V.

2. If $n' = 1$ solve the problem as a 1-dimensional problem as described above.

3. Otherwise, let $x_{mid} \notin V$ be choosen such that the number of elements in V that are $< x_{mid}$ and the number of elements in $V > x_{mid}$ are both $\leq \lceil \frac{1}{2}n' \rceil$. Split P into $P_1 = \{p \in P | p_x < x_{mid}\}$ and $P_2 = \{p \in P | p_x > x_{mid}\}$. Split O into $O_1 = \{o \in O | o_x < x_{mid}\}$ and $O_2 = \{o \in O | o_x > x_{mid}\}$.

4. Report all dominance pairs in P_1 with respect to O_1 and in P_2 with respect to O_2 recursively in the same way.

5. Report all dominance pairs $(p, q) \in P_2 \times P_1$ with respect to O.

Step 1 of the algorithm can be performed in time $O(n)$ if we have P, O and V sorted by x-coordinate. After presorting on x-coordinate, which requires time $O(n \log n)$ these sets can be maintained sorted during the recursive calls.

When the recursion stops (in step 2) we have to sort the, say, n_i points that are in the sets by y-coordinate. This has to be done n' times, but, since we have $\sum_{i=1}^{n'} n_i = n$, the total time required for step 2 will be bounded by $O(n \log n)$ plus $O(1)$ time for every answer found.

Now let $T(n', n)$ be the time needed for the algorithm, then we have:

$$\begin{aligned} T(n', n) &= O(n \log n + k) + T'(n', n) \\ T'(n', n) &= T'(\lceil \tfrac{1}{2}n' \rceil, n - l) + T'(\lfloor \tfrac{1}{2}n' \rfloor, l) + O(n) + f(n) \end{aligned} \tag{1}$$

where k is the number of dominance pairs, $0 \leq l \leq n$ and $f(n)$ is the time needed to perform the merge step (step 5). Assuming that $f(n)$ is non-decreasing and at least linear this leads to

$$T(n', n) = O(n \log n + k + \log n'(n + f(n)) = O(f(n) \log n + k) \tag{2}$$

because $n' = O(n)$.

It remains to be shown how the merge step (step 5) can be performed efficiently. First note that we only have to look at pairs $(p, q) \in P_2 \times P_1$ since all points in P_2 have larger x-coordinate than the points in P_1. So $p \succ q$ iff $p_y \geq q_y$.

The idea is as follows: We move a scanline downward over the plane, halting at every point in $P \cup O$. When we encounter a point $q \in P_1$, we will report all pairs $(p, q) \in P_2 \times P_1$ with $p \succ_O q$. We know that p must lie above (or on) the scanline. To find these points, for every point $p \in P_2$ above the scanline we keep track of a so-called *dominance interval* DI_p at the current position y^* of the scan line. This dominance interval consists of all x-values $< x_{mid}$ such that $x \in DI_p \Leftrightarrow p \succ_O (x, y^*)$. In other words, when the x-coordinate of a point $q \in P_1$ on the scanline lies in DI_p then $p \succ_O q$. Note that DI_p is indeed always an interval, which is of the form $]o_x, x_{mid}[$ where o_x is either $-\infty$ or the x-coordinate of some obstacle point (or DI_p is empty). See figure 2 for some examples of dominance intervals.

Suppose for the moment that no two points have the same y-coordinate. (If points do have the same y-coordinate we handle them from right to left. If a point $p \in P_2$ coincides with a point $o \in O$ then o is treated first. If a point $q \in P_1$

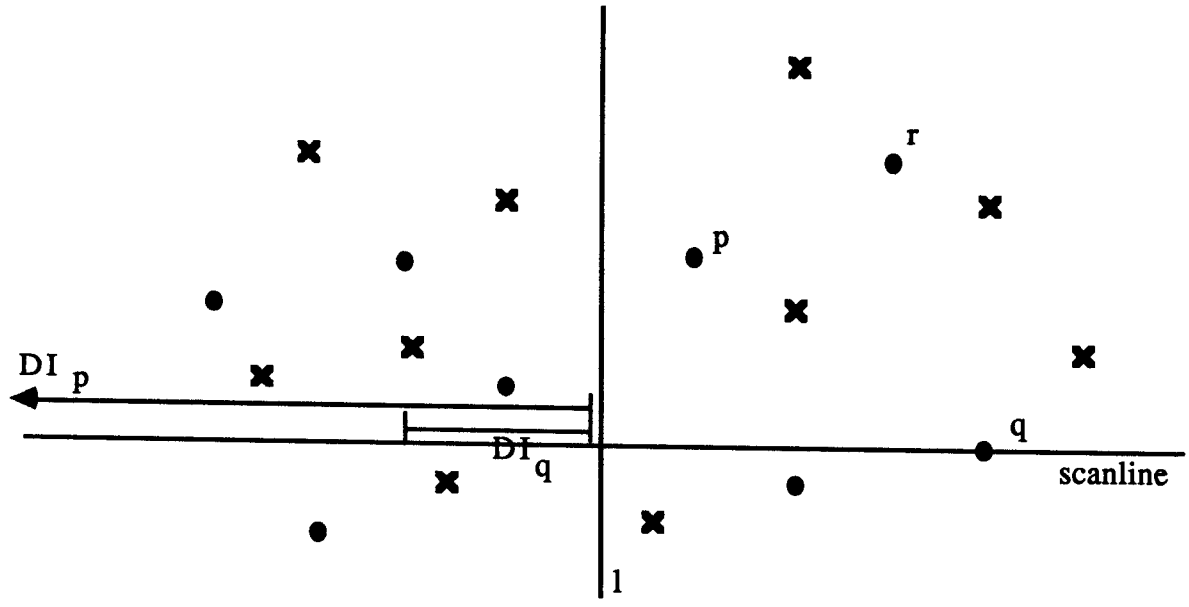


Figure 2: Some dominance intervals. $DI_r = \emptyset$. Points in P are indicated with a dot, point in O with a cross.

coincides with a point $o \in O$ then q is treated first. The reader can easily verify that this will be the correct order.) If we encounter a point $p \in P_2$ we must initialize $DI_p :=]-\infty, x_{mid}[$. If we encounter a point $q \in P_1$, for all $p \in P_2$ with $q_x \in DI_p$ we report the pair (p, q) as a dominance pair. Obstacle points must be treated in the following way: An obstacle point $o = (o_x, o_y)$ dominates all the points to the left and below of it, so when we encouter o we have to change the dominance intervals for all points p that dominate o in the following way: if $o \in O_1$ then for all p with $o_x \in DI_p$ we must set $DI_p :=]o_x, x_{mid}[$ and if $o \in O_2$ then for all p with $o_x \leq p_x$ we must set $DI_p := \emptyset$.

Observe that after we have handled an obstacle point some of the dominance intervals become identical. To avoid changing all these intervals again at some later obstacle point we from now on treat them simultaneously. To this end we store identical DI's only once and associate a bag with it that contains all the points p for which $DI_p = DI$. This bag must allow for the following operations in constant time: inserting an element, deleting an element when we have a pointer to it and joining two bags. Moreover, all the elements should be enumerated in time the number of elements. This can be implemented e.g. as a doubly linked list.

To be able to handle obstacle points $o \in O_2$ efficiently, we must be able to determine all points in P_2 above the scan line that lie to the right of o and remove them. A priority queue on (the x–coordinates of) the points in P_2 above the scanline will suffice for this purpose. Since we have to set $DI_p := \emptyset$ for a point p to the right of o in this case (i.e. remove p from the bag it is in) we also store a cross pointer from the place of p in the priority queue to the the place of p in the bag.

We now present step 5 of the algorithm in more detail:

5. Move a scanline downward over the set of points, halting at every point $(x, y) \in P \cup O$. (To do this we need a list of points $\in P \cup O$, sorted according to y-coordinate. This sorted list can be obtained from the sorted lists of $P_1 \cup O_1$ and $P_2 \cup O_2$ by a simple merge. This means that we only have to sort explicitly when the recursion halts. This sorting was already performed to compute the answers in step 2.) While we move the scanline we maintain the following two data structures:

   - A sorted list L of the different left endpoints of the dominance intervals of points in P_2 above the scanline. Every left endpoint has a bag associated with it, that contains all the points in P_2 that have that left endpoint as the left endpoint of their dominance interval.
   - A priority queue Q, containing the x-coordinates of the points in P_2 above the scanline.

   Furthermore we maintain cross pointers from the points in Q to the corresponding points in (the bags in) L. When we halt at a point (x, y) we have the following cases:

   $(x, y) \in P_1$: Walk with x along L as long as the left endpoint of the current bag is $< x$ and report $(p, (x, y))$ as a dominance pair for each point p in these bags.

   $(x, y) \in O_1$: Walk with x along L, joining all the bags with left endpoint $\leq x$ into a new bag with x as left endpoint.

   $(x, y) \in P_2$: Insert (x, y) in Q and add it to the bag in L with $-\infty$ as left endpoint (or, if necessary, create a new bag).

   $(x, y) \in O_2$: Remove all points with x–coordinate $\geq x$ from Q and, using the cross pointers, from the bags in L. (If a bag becomes empty, then remove the bag and its corresponding DI from L.)

The correctness of the algorithm follows from the above discussion.

Lemma 2.1 *The merge step can be performed in time $O(n \log n + k)$.*

Proof. To analyse the time complexity we have to look at the four different cases:

$(x, y) \in P_1$: We spend $O(1 + \#\text{answers})$ time.

$(x, y) \in O_1$: We spend $O(1 + (\#(\text{bags with left endpoint} \leq x)\text{-}1))$ time. We charge the costs for joining the bags as follows to points in P_2: Let q_b be the first point that is put in bag b. Then we charge the costs of joining bags $b_1, \ldots, b_s$ into bag b_s to the points $q_{b_1}, \ldots, q_{b_{s-1}}$. It is clear that a point can be charged costs only once this way. Thus every point $\in P_2$ gets an extra $O(1)$ at most. The remaining $O(1)$ time of step (ii) is simply charged to (x, y).

$(x, y) \in P_2$: We spend $O(\log n)$ time to insert the point in the priority queue.

$(x, y) \in O_2$: We spend $O(\#\text{deletions}(\log n + 1))$ time for deleting the points from the priority queue and from the bags. We charge these costs to the deleted points. Since a point can be deleted only once, every point $\in P_2$ is charged with this extra $O(\log n + 1)$ at most once.

The total time bound follows immediately. (Note that no sorting is required because the points where already presorted. We only have to merge the lists which requires time $O(n)$.) □

So we have $f(n) = O(n \log n)$ in equation 2. This leads to the following result:

Theorem 2.2 *All dominance pairs in a set of points P with respect to a set of points O can be computed in time $O(n \log^2 n + k)$, where $n = |P| + |O|$ and k is the number of answers.*

3 Normalizing the problem.

When we take a closer look at the time analysis of the above algorithm, we see that the operations on the priority queue Q form the bottleneck in the algorithm. The question thus arises wether we can use another data structure that performs these operations more efficiently. Such a data structure indeed exists for our application.

The crucial observation here is that we can normalize the problem (see e.g. [5]), i.e., convert it to the corresponding problem on a grid, without changing the dominance pairs. So we add as a preliminary step to the algorithm:

0. Replace every point $(x, y) \in P \cup O$ by $(r(x), r(y))$, where $r(x)$ and $r(y)$ are the ranks of x and y in the sorted order of the different x and y-coordinates, respectively.

Since

$$r(x) \leq r(x') \Leftrightarrow x \leq x',$$
$$r(y) \leq r(y') \Leftrightarrow y \leq y' \text{ and}$$
$$(r(x), r(y)) = (r(x'), r(y')) \Leftrightarrow (x, y) = (x', y')$$

we have

$$(r(x), r(y)) \succ_O (r(x'), r(y')) \Leftrightarrow (x, y) \succ_O (x', y').$$

Hence, normalization maintains the dominance relation.

The normalization step can be performed in time $O(n \log n)$ by sorting the points by x- and y-coordinate. As this sorting was already performed by the algorithm it does not increase the time bound.

Now that we have normalized the problem we can use data structures that work on a grid. This means that we can use a VanEmdeBoas tree (see, e.g., [9],[10])

as priority queue. In such a tree INSERT and EXTRACTMAX can be performed in time $O(\log \log U)$, where U is the size of the universe, in our case $U = O(n)$. Thus we can perform step 5 in $O(n \log \log n + k)$ time and we obtain the following improved result:

Theorem 3.1 *All dominance pairs in a set of points P with respect to a set of points O can be computed in time $O(n \log n \log \log n + k)$, where $n = |P| + |O|$ and k is the number of answers.*

4 Handling sets of different sizes.

In a number of applications the number of points in P and the number of points in O might differ considerably. In this section we will show that in such cases the method can be adapted to work more efficient. Let $n_o = |O|$ and $n_p = |P|$. We first treat the case when $n_o >> n_p$.

Theorem 4.1 *All dominance pairs in a set of points P of size n_p with respect to a set of points O of size n_o can be computed in time $O(n \log n_p \log \log n_p + k)$, where k is the number of answers.*

Proof. We first perform a normalization step in the following way: Let V_x be the set of different x-coordinates of points in P. We define $r'_x(x)$ as follows: If $x \in V_x$ then $r'_x(x)$ is twice the rank of x in V_x. If $x \notin V_x$ then determine the largest $x' \in V_x$ with $x' < x$. If x' does not exist $r'_x(x) = 1$, otherwise $r'_x(x) = r'_x(x') + 1$. Similar, let V_y be the set of all y-coordinates and define $r'_y(y)$ in an analogue way. Now replace each point $p \in P \cup O$ by $p' = (r'_x(p_x), r'_y(p_y))$. It is easy to verify that in this way the dominance relation between points in P with respect to O is maintained. We have mapped the problem onto a grid of size at most $2n_p + 1 \times 2n_p + 1$. See figure 3 for an example. As a result some obstacle points might become the same. We remove all except one of them. So the size of O has become $O(n_p^2)$. This normalization can be carried out in time $O(n \log n_p)$.

Now we just run the algorithm as described above. Because the number of different x-coordinates is $O(n_p)$ equation 2 becomes $T(n) = O(f(n) \log n_p + k)$. Each operation on the VanEmdeBoas tree will require $O(\log \log n_p)$ and the sorting in step 2 of the method will be bounded by $O(n \log n_p)$ in total. The bound follows. □

In particular, when the number of points in P is constant, the method runs in time $O(n)$ (something which can also be achieved in a more simple way by testing every pair of points in P with respect to all obstacle points).

Let us now consider the case when $n_p >> n_o$.

Theorem 4.2 *All dominance pairs in a set of points P of size n_p with respect to a set of points O of size n_o can be computed in time $O(n \log n + n \log n_o \log \log n_o + k)$, where k is the number of answers.*

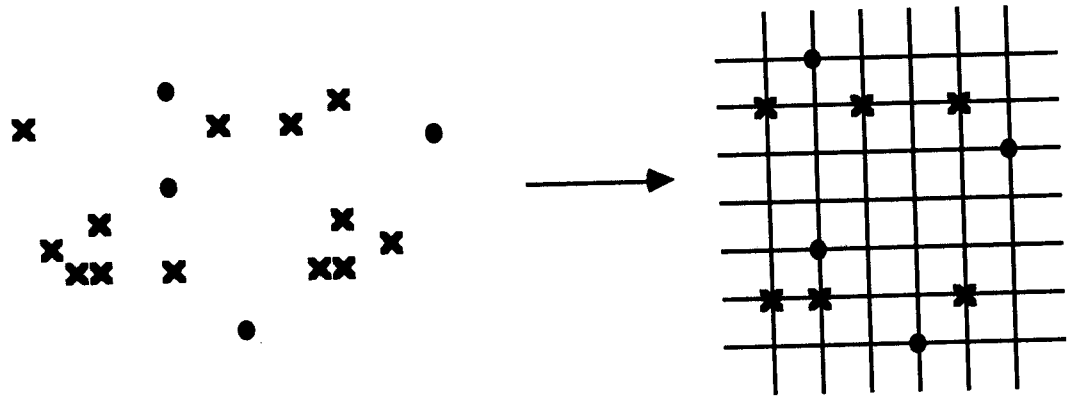


Figure 3: Normalizing the set of point when n_p is small.

Proof. To obtain the result we use a normalization similar to the previous theorem. Let V_x be the set of different x-coordinates of points in O. Define $r'_x(x)$ in a similar way as above. Now replace each point $p \in P \cup O$ by $p' = (r'_x(p_x), p_y)$. (Note that we use the normalization only on x-coordinate.) Unfortunately this does not maintain the dominance relation. Let P' and O' be the new sets. One easily verifies that $p \succ_O q \not\Leftrightarrow p' \succ_{O'} q'$ if and only if $p_x \neq q_x \wedge p'_x = q'_x$. This situation can only occur when the recursion stops, i.e., when we perform step 2 of the method. When we perform step 2 there are two possible cases: i) there is an obstacle with the x-coordinate. In this case also the original points were on a vertical line and we can proceed in the normal way. ii) there is no obstacle with this x-coordinate. In this case the original points might not lie on a vertical line. But we know that there is no obstacle point among them. So we can just compute all dominance pairs among the original points in time $O(n_i \log n_i)$ when we are left with n_i points. In this way, the time spend on step 2 will be no more than a total of $O(n \log n)$. So equation 2 turns into $T(n) = O(n \log n + f(n) \log n_o + k)$. It is easily verified that $f(n) = O(n \log \log n_o)$ because the number of different x-coordinates at a merge step is bounded by $2n_o + 1$. □

Note that both results hold for arbitrary n_p and n_o. Also note that it is easy to determine whether $n_p > n_o$ or not and choose the correct method in each case. Hence we can in fact solve the problem in the minimum of the two bounds without preknowledge on the sizes of P and O.

5 The case $P \subseteq O$.

In some cases, like the direct dominance problem, we have the situation that $P \subseteq O$. We will now show that in this case we can improve the result to $O(n \log n + k)$. To this end we will avoid the use of a priority queue completely.

Note that, when a point $p \in P_2$ did coincide with an obstacle point $o \in O$ we

treated the obstacle point first. As a result there will be no points left in the tree Q to the right of p. But when $P \subseteq O$ this will always be the case. This means that whenever we handle a point $p \in P_2$, we have just removed all points from Q with x-coordinate $\geq p_x$. This implies that we could as well use a stack for Q. When we encounter an obstacle point $o \in O_2$ we pop a number of points from the stack and when we encounter a point $p \in P_2$ we just push it on the stack. Thus we can perform both INSERT and EXTRACTMAX in Q in time $O(1)$. As a result the merge step can be performed in time $O(n+k)$. (Note that the normalization is not neccessary in this case.)

Theorem 5.1 *All dominance pairs in a set of points P with respect to a set of points $O \supseteq P$ can be computed in time $O(n \log n + k)$, where $n = |P| + |O|$ and k is the number of answers.*

6 Treating general obstacles.

We will now concentrate on the case O consists of arbitrary objects rather than points. We will reduce the problem to reporting all the dominance pairs in P with respect to a set O' that contains only points. Then we can apply the results of the previous sections. So we are interested in finding a set O' such that for all $p, q \in P : p \succ_O q \Leftrightarrow p \succ_{O'} q$.

Now let the objects in the obstacle set O satisfy the following constraints:

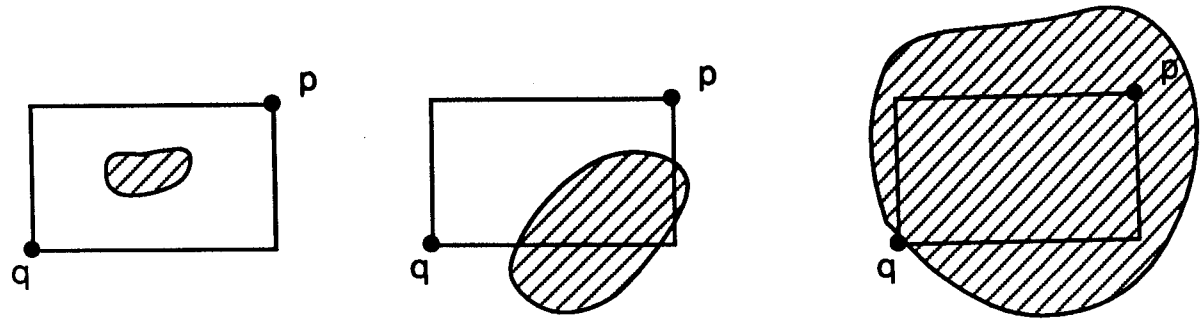
1. The objects are connected.
2. A point inside the object can be determined in constant time.
3. The first intersection between an object $o \in O$ and an axis-parallel ray can be determined in constant time (if any). If the starting point p of the ray lies in o p itself should be reported.
4. Whether or not a square and an object intersect can be computed in constant time.

We can compute O' as follows. Consider two points $p, q \in P$ with q to the left of and below p. As noted in the introduction, to say that $p \succ_O q$ is the same as to say that the intersection of the rectangle with p and q as opposite vertices with every obstacle in O is empty. Therefore we define:

Definition 6.1 *Let $p = (p_x, p_y)$ and $q = (q_x, q_y) \in \mathbf{R}^2$ with $p \succ q$. We define*

$$Rect(p, q) = \{(x, y) \in \mathbf{R}^2 | x \in [q_x, p_x], y \in [q_y, p_y]\} - \{p, q\}.$$

Lemma 6.1 *Let p and q be two points in P, $p \succ q$, then $p \succ_O q \Leftrightarrow Rect(p, q) \cap o = \emptyset$ for all $o \in O$.*



case i) case ii) case iii)

Figure 4: The different cases where $Rect(p,q) \cap o \neq \emptyset$.

Now note that if for two points p and q and an obstacle o we have $Rect(p,q) \cap o \neq \emptyset$ then o must either lie totally inside $Rect(p,q)$, or o must intersect the boundary of $Rect(p,q)$ somewhere, or o must totally cover $Rect(p,q)$. See figure 4 for some examples of the three cases. Of course we cannot afford to look at every pair of points to generate the correct obstacle points. Hence, we have to create points in a more careful way.

First of all, for each obstacle $o \in O$ we take some point inside the obstacle and add it to O'. This will solve the first case. Secondly, for each point p in P we shoot a ray upwards, downwards, left and right, and determine for each obstacle the first intersection. In each direction we add to O' the nearest such intersection point unequal to p (if such an intersection exists). Clearly, when an object intersects the boundary of $Rect(p,q)$ one of the obstacle points created lies on this boundary and, hence, is contained in $Rect(p,q)$.

It remains to avoid the third case. When the third case occurs both p and q lie on (the boundary of) an obstacle. It would seem correct to remove such points but this can only be done when they do not lie on the boundary. So we choose another way of solving this. Let

$$\varepsilon = \frac{1}{2}\min(\min_{p,q \in P, p_x \neq q_x} |p_x - q_x|, \min_{p,q \in P, p_y \neq q_y} |p_y - q_y|).$$

In words, ε is half of the smallest vertical or horizontal distance between points that is not equal to 0. Hence, if we draw a horizontal or vertical bar around a point $p \in P$ with width 2ε it does not contain any other point in P with different x- or y-coordinate. For every $p \in P$ do the following: Let S_p be the square with center p and sides of length 2ε. Let NE_p be the north-east quadrant of S_q and let SW_p be the south-west quadrant, not including the boundaries (i.e., they are open squares). If NE_p or SW_p has a non-empty intersection with an obstacle $o \in O$ add an arbitary point of this quadrant to O'. Also consider the four boundaries between

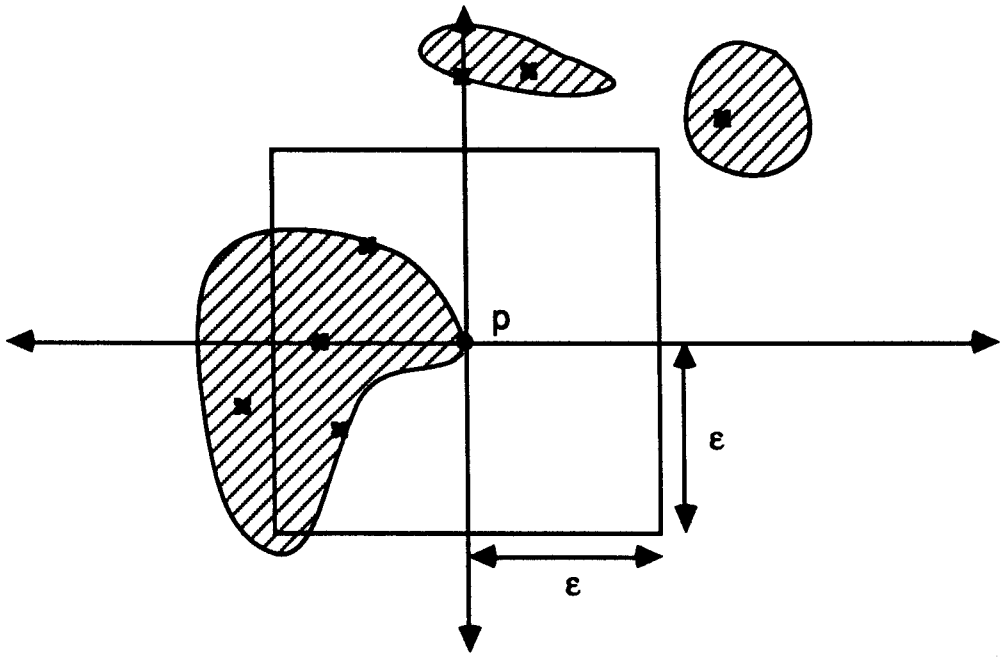


Figure 5: Points added to O'.

quadrants, not including p. For each boundary, if it has a non-empty intersection with an obstacle $o \in O$ then add an arbitrary point of this boundary to O'. See figure 5 for some examples.

Now it is clear that when some obstacle completely covers $Rect(p, q)$ some of the points created will lie inside $Rect(p, q)$. When $Rect(p, q)$ is empty none of the points created will lie inside $Rect(p, q)$ even when p or q lies on the boundary of an obstacle.

Note that $|O'| \leq |O| + 10|P|$. (By choosing the points a bit more careful we could reduce this to $|O| + 5|P|$.)

Lemma 6.2 *For all* $p, q \in P : p \succ_O q \Leftrightarrow p \succ_{O'} q$.

Proof. Follows from the above discussion. □

(Note that, using the same method it is easy to show that for any set of obstacles, not necessarily satisfying the constraints, a set O' of points does exist that represents the set of obstacles. The number of points in such a set would be proportional to the number of connected parts. The only problem is that it might be impossible to determine the set. Because of that we introduced the constraints.)

Now that we have reduced the obstacle set O to a set O' of size $O(n)$ that contains only points we can apply the results of the previous section. We thus find:

Theorem 6.3 *All dominance pairs in a set of points P with respect to an obstacle set O can be computed in time $O(R(n) + n \log n \log\log n + k)$, where $n = |P| + |O|$, k is the number of answers and $R(n)$ is the time needed to perform the reduction.*

If all points in P lie on some obstacle in O we can add the points in P to O'. We then obtain $P \subseteq O'$ and obtain:

Theorem 6.4 *All dominance pairs in a set of points P with respect to an obstacle set O with $P \subseteq O$ can be computed in time $O(R(n)+n\log n+k)$, where $n = |P|+|O|$, k is the number of answers and $R(n)$ is the time needed to perform the reduction.*

It is obvious that the O' can be computed in time $O(n^2)$. But this would make the method not very efficient. In a number of cases the reduction can be performed much more efficiently.

Theorem 6.5 *When O consists of non-intersecting convex objects, satisfying the above constraints plus the constraint that the leftmost, rightmost, topmost and bottommost point can be determined in $O(1)$ time, the reduction can be performed in time $O(n\log n)$.*

Proof. First of all we have to determine ϵ. This can be done by sorting P both horizontally and vertically in $O(n\log n)$ time. Next we determine a point in each obstacle which takes time $O(n)$.

To determine the first intersection with the rays from each point we move a scanline from left to right over the set of obstacles. With the scanline we keep a tree that holds the objects intersecting the scanline in sorted order. For each point $p \in P$ the scanline passes we can determine in $O(\log n)$ time the first intersection above and below p. Similar, using a vertical scan we determine the first intersection to the left and to the right of each point p. Obstacles can be inserted and deleted in $O(\log n)$ time.

Note that we only have to look at the square around p when p lies on an obstacle. Moreover this obstacle is the only one we have to consider for intersection with the square because the obstacles do not intersect. This obstacle can also be determined during the scan and points can be added if necessary. Details are left to the reader. □

In fact, we only use the property that every horizontal and vertical line intersects each object in one interval and these intervals do not intersect. Hence, the result also applies to other types of obstacles.

As an example consider the case in which O consists of a set of non-intersecting line segments and P is the set of endpoints of these line segments. The above results show that all dominance pairs can be determined in time $O(n\log n+k)$. The dominance pairs we get this way form a kind of *rectangular visibility graph* (compare the results on normal visibility graphs, see [4,7]).

Also for other sets of obstacles the reduction can be performed efficiently. For example when we have a set of (possibly intersecting) axis-parallel line segments the reduction can be performed in time $O(n\log n)$.

7 Concluding Remarks.

In this paper we studied a generalization of the direct dominance problem by considering dominance pairs with respect to a set of obstacles. In the case the obstacles were points we demonstrated that all dominance pairs in a set P with respect to an obstacle set O can be computed in time $O(n \log n \log \log n + k)$ where $n = |P| + |O|$ and k is the number of reported pairs. We improved this result in the case the two sets have unequal sizes and in the case $P \subseteq O$. In the later case we gave an $O(n \log n + k)$ solution which is optimal in the worst-case.

Finally, we generalized the result to the case the obstacle set O consists of other objects than points. It was shown that given a set P of points and a set O of arbitrary obstacles there always exists a set O' of points such that points p and q in P form a dominance pair with respect to O if and only if they form a dominance pair with respect to O'. When each obstacle is connected, the size of O' is $O(|O| + |P|)$. If the obstacles satisfy some constraints the set O' can also be found efficiently.

A number of open problems do remain. First of all it might be possible to improve out main result to $O(n \log n + k)$. The problem here is the fact that we have to maintain a priority queue for the points in P_2 above the scanline in the merge step of our divide-and-conquer algorithm. (Recall that P_2 is the subset of points in P that lie to the right of the dividing line.) A way to improve the efficiency here might be to exploit the correspondence between the subsequent levels of the recursion, for example by some sort of preprocessing or by transforming the recursive algorithm into an iterative one. Our attemps have failed so far.

A second question is whether the results can be extended to higher-dimensional space. The results in [3] and [8] are not very promising here. They are able to solve the 3-dimensional direct dominance problem in time $O((n+k)\log^2 n)$ but are unable to generalize their results any further.

Other extensions might be to look at maximal elements in the presence of obstacles (i.e., points that are not dominated by any other points with repect to O) or counting the number of points that dominate a particular point (a kind of ECDF-counting problem, see e.g. [1,2]). Also query versions of the problems are worth studying.

Acknowledgement

We would like to thank Derick Wood (University of Waterloo, Canada) for the helpful discussions on this topic during his stay in Utrecht in November '87.

References

[1] Bentley, J.L., Multidimensional divide-and-conquer, *Comm. ACM* 23 (1980), pp. 214-229.

[2] Bentley, J.L., and M.I. Shamos, A problem in multivariate statistics: algorithm, data structure and applications, *Proc 15th Allerton Conference on Communication, Control and Computing,* 1977, pp. 193-201.

[3] Güting, R.H., O. Nurmi and T. Ottmann, The direct dominance problem, *Proc. 1st ACM Symp. Computational Geometry,* 1985, pp. 81-88.

[4] Ghosh, S.K., and D.M. Mount, An output sensitive algorithm for computing visibility graphs, *Proc. 28th Symp. on Foundations of Computer Science,* 1987, pp. 11-19.

[5] Karlsson, R.G., and M.H. Overmars, Normalized divide-and-conquer: A scaling technique for solving multi-dimensional problems, *Inform. Proc. Lett.* 26 (1987/88) pp. 307-312.

[6] Munro, J.I., M.H. Overmars and D. Wood, Variations on visibility, *Proc. 3rd ACM Symp. Computational Geometry,* 1987, pp. 291-299.

[7] Overmars, M.H., and E. Welzl, New methods for computing visibility graphs, *Proc. 4th ACM Symp. Computational Geometry,* 1988, to appear.

[8] Overmars, M.H., and D. Wood, On rectangular visibility, *J. Algorithms* (1988), to appear.

[9] van Emde Boas, P., Preserving order in a forest in less than logarithmic time and lineair space, *Inform. Proc. Lett.* 6 (1977) pp. 80-82.

[10] van Emde Boas, P., R. Kaas and E. Zijlstra, Design and implementation of an efficient priority queue, *Math. Systems Theory* 10 (1977) pp. 99-127.