Connectability problems

Mark H. Overmars

RUU-CS-88-21
April 1988

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

Connectability problems

Mark H. Overmars

Technical Report RUU-CS-88-21
April 1988

Department of Computer Science
University of Utrecht
P.O.Box 80.089
3508 TB Utrecht
the Netherlands

Connectability problems

Mark H. Overmars

April 1988

Abstract

In this paper we define a general class of problems in computational geometry that we call *connectability problems*. Connectability problems involve connecting objects by some kind of connections, avoiding obstacles. This includes many different types of problems like intersection problems, visibility problems, etc. Studying these problems in a general framework might lead to general solutions. Some solutions are presented. In particular, an $O(n \log n \log\log n)$ solution is given for determining all pairs of points in a set that can be connected with an axis-parallel rectangle, avoiding a set of obstacle points.

1 Introduction.

A large class of problems in computational geometry can be formulated as *connectability problems*. In these problems we ask whether objects can be connected in some kind of way. For example, when we ask whether two points can see each other among a set of obstacles we ask whether there is a line segment, not intersecting any obstacle, that connects the two points.

We can generalize this in the following way: Let $\mathcal{C}$, $\mathcal{S}$, $\mathcal{G}$ and $\mathcal{B}$ be classes of objects in some d-dimensional space. $\mathcal{C}$ is the collection of *connections*, $\mathcal{S}$ the collection of *sources*, $\mathcal{G}$ the collection of *goals* and $\mathcal{B}$ the collection of *barriers*. Let $s \in \mathcal{S}$, $g \in \mathcal{G}$ and $B \subset \mathcal{B}$.

Definition 1.1 *We call s and g $\mathcal{C}$-*connectable *with respect to B, denoted as $s \rightsquigarrow_{\mathcal{C}} g$, iff*

$$\exists_{c \in \mathcal{C}}\ s \cap c \neq \emptyset,\ g \cap c \neq \emptyset \text{ and } \forall_{b \in B}\ b \cap c \subseteq s \cup g$$

In other words, there exists an object in $\mathcal{C}$ that intersects both s and g and if it intersects an obstacle, this intersection is contained in $s \cup g$. (It would have been easier to state that $b \cap c = \emptyset$, but that would not allow for connections between e.g. points that lie on the obstacles.)

Using the notion of connectability we can define two different types of problems:

All pairs problem Let $S \subset \mathcal{S}$, $G \subset \mathcal{G}$ and $B \subset \mathcal{B}$. The problem asks for all pairs s, g with $s \in S$ and $g \in G$ such that $s \rightsquigarrow_{\mathcal{C}} g$ with respect to B. The complexity of the problem will be dependent on:

- The collections $\mathcal{C}, \mathcal{S}, \mathcal{G}$ and $\mathcal{B}$.
- The dimension d.
- The cardinality of the sets. For example $|S| = 1$, $|G| = 1$ and $|B| = n$ is quite different from $|S| = n$, $|G| = n$ and $|B| = 1$.
- Whether some sets are equal, for example, there exist instances of the problem in which $S = G = B$.
- Whether we want to enumerate the answers or just want their number.

Query problem Let $G \subset \mathcal{G}$ and $B \subset \mathcal{B}$. Store G and B in a datastructure such that for any object $s \in \mathcal{S}$ we can efficiently determine those objects in G that can be connected with s. Such a data structure can be static or dynamic. In the dynamic case we can distinguish between three different cases: dynamic with respect to G, dynamic with respect to B and dynamic with respect to both. Again the complexity highly depends on the type of connectability, the collections of objects, the cardinality of the sets G and B and whether they are equal, etc.

Many well-known problems can be formulated as connectability problems. Let us just mention two examples:

The *visibility problem* asks for those objects in 2- or 3-dimensional space that can be seen from a point. Seeing means that there is a line segment connecting the point with the object that does not intersect any other object. Hence, $\mathcal{C}$ is the collection of all line segments, $\mathcal{S}$ is the collection of points, and $\mathcal{G}$ and $\mathcal{B}$ are the collection of objects. We get an all pairs problems with $|S| = 1$ and $G = B$. Other types of visibility, like visibility from an edge, etc., can be formulated in terms of connectability as well.

Intersection problems ask which pairs of objects in some set intersect. Intersection means: having some point in common. So if we take S and G the set of objects, $\mathcal{C}$ the collection of all points, and B empty, we have formulated the intersection problem as a connectability problem.

The notion of connectability gives rise to thousands of different problems. Reasonable choices for $\mathcal{S}$, $\mathcal{G}$, $\mathcal{C}$ and $\mathcal{B}$ are: points, axis-parallel line segments, line segments, rectangles, squares, circles, polygons of fixed shape and arbitrary polygons, etc. and their extensions to higher dimensional space. For each combination of classes we get different problems that might need different solution methods.

Studying each of these individual problems is not very usefull (unless the problem has an immediate application). It is much more interesting to try and find general techniques that solve large classes of connectability problems. In this paper we will solve some types of connectability problems. In particular, we look at connections with axis-parallel rectangles and with homothetic objects. We will restrict ourselves to the case in which all three sets $\mathcal{S}$, $\mathcal{G}$ and $\mathcal{B}$ are collections of points in the plane. Moreover, we will only look at the all pairs problem in the case $S = G$. So we are given a set of points that we will call P and a set of barrier points B and we ask for all pairs $(p, q) \in P \times P$ such that $p \rightsquigarrow_{\mathcal{C}} q$ with respect to B. At some places it is indicated how results can be generalized.

2 Reducable connections.

We will first look at a very general technique for solving connectability problems. To this end we look at some properties of collections of connections.

Definition 2.1 *A collection of connections $\mathcal{C}$ is called* reducable *if for any pair of points p, q there exists one minimal connection $c \in \mathcal{C}$ such that c connects p and q and any $c' \in \mathcal{C}$ that connects p and q does contain c.*

A number of classes of connections are reducable. For example, when $\mathcal{C}$ is the class of line segments, then the minimal connection between p and q is $\overline{pq}$. When $\mathcal{C}$ is the class of axis-parallel rectangles, the minimal connection is the rectangle with p and q as opposite vertices. But there are also many classes that are not reducable, for example, the set of all circular disks is not reducable.

When the set of connections is reducable p and q can be connected if and only if they can be connected by the minimal connection. We assume that this minimal connection can be computed in O(1) time. This immediately leads to the following result:

Theorem 2.1 *Let S, G and B be sets of points and let $\mathcal{C}$ be a reducable set of connections. The pairs $(p, q) \in S \times G$ with $p \rightsquigarrow_{\mathcal{C}} q$ with respect to B can be computed in time $O(|S| * |G| * |B|)$.*

Proof. Simply, for each pair $(p, q) \in S \times G$ we compute the minimal connection and test whether it intersects any point in B. □

Obviously, for many problems this result can be improved. For example, when the connections are rectangles, we can use a scanline technique to remove the minimal rectangles that contain a barrier point in time $On^2 \log n$). (In the next section it will be shown that the problem can be solved much more efficient.)

When the connections are line segments we could use the following method. For each point $p \in S$ sort the points in $G \cup B$ by angle around p. For each direction d from p where there is a point $q \in G$ check whether there is a point $b \in B$ in the same direction that lies closer to p. If such a point b does not exist, report the pair (p, q) as an answer. Otherwise q can not be connected to p. After the sorting this takes time $O(|G| + |B|)$ per point p. The sorting can be done for all points simultaneously in time $O(n^2)$ where $n = |S| + |G| + |B|$ using the techniques from [2,8]. This leads to the following result:

Theorem 2.2 *Let S, G and B be sets of points in the plane and let $\mathcal{C}$ be the collection of line segments. The pairs $(p, q) \in S \times G$ with $p \rightsquigarrow_{\mathcal{C}} q$ with respect to B can be computed in time $O(n^2)$ where $n = |S| + |G| + |B|$.*

This is optimal in the worst case as the number of reported pairs can be $\Omega(n^2)$. But it might be possible to construct output sensitive algorithms that work faster when the number of answers is small. When $S = G = B$ the techniques of Ghosh and Mount[5] or Overmars and Welzl[8] for computing a visibility graph can be used to obtain an algorithm that runs in time $O(n \log n + k)$ where k is the number of reported pairs. No bounds are known when the sets are different.

3 Rectangular connections.

We will now concentrate on rectangular connections, i.e., $\mathcal{C}$ consists of all axis-parallel rectangles. The notion of rectangular connectability has been studied before. Overmars and Wood[9] call this rectangular visibility (see also [7]). Güting Nurmi and Ottmann[4] define the notion of direct dominance that is also (almost) equivalent to rectangular connectability. In both papers an $O(n \log n)$ algorithm is presented to find all pairs in a set P of n points where $B = P$. We will concentrate on the case in which B is a different set of points. The methods will only be described briefly. See de Berg and Overmars[1] for more details (and more results).

First note that rectangular connections are reducable. Hence, we only have to consider rectangles with points of P as opposite vertices. We will present a method that only reports the connectable pairs (p, q) where p is the left bottom vertex and q the top right vertex. The other pairs can be found in a similar way.

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

When $n' > 1$, 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 B in two halves. It is obvious that whether or not two points in one half of the plane form a connectable pair cannot be influenced by a barrier point from the other half. So we can recursively treat the two halves in the same way. After this it remains to compute the connectable pairs between the points in different halves. This merge step will be described below.

Finding the value x_{mid} can be done in time $O(n)$ if we have P, B and V sorted by x-coordinate. Also the splitting in halves takes time $O(n)$. 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 (i.e., when V contains one value) 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 this 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 connectable pairs, $0 \leq l \leq n$ and $f(n)$ is the time needed to perform the merge step. 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 can be performed efficiently. Let P_1 be the set of points from P to the left of the splitting line and P_2 be the set to the right of the line. Similar for B. We only have to consider pairs $(p, q) \in P_2 \times P_1$.

The idea is as follows: We move a scanline downward over the plane, halting at every point in $P \cup B$. When we encounter a point $q \in P_1$, we will report all pairs $(p, q) \in P_2 \times P_1$ with $p \rightsquigarrow_C q$. 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 *connectability interval* CI_p at the current position y^* of the scanline. This connectability interval consists of all x-values $< x_{mid}$ such that $x \in CI_p \Leftrightarrow p \rightsquigarrow_C (x, y^*)$. In other words, when the x-coordinate of a point $q \in P_1$ on the scanline lies in CI_p then $p \rightsquigarrow_C q$. Note that CI_p is indeed always an interval, which is of the form $]b_x, x_{mid}[$ where b_x is either $-\infty$ or the x-coordinate of some barrier point (or CI_p is empty).

Let us assume 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 $b \in B$ then b is treated first. If a point $q \in P_1$ coincides with a point $b \in B$ 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 $CI_p :=]-\infty, x_{mid}[$. If we encounter a point $q \in P_1$, for all $p \in P_2$ with $q_x \in CI_p$ we report the pair (p, q) as a connectable pair. Barrier points must be treated in the following way: A barrier point $b = (b_x, b_y)$ blocks all the points to the left and below of it, so when we encouter b we have to change the connectability intervals for all points p that $\rightsquigarrow_C b$ in the following way: if $b \in B_1$ then for all p with $b_x \in CI_p$ we must set $CI_p :=]b_x, x_{mid}[$ and if $b \in B_2$ then for all p with $b_x \leq p_x$ we must set $CI_p := \emptyset$.

Observe that after we have handled a barrier point some of the connectability intervals become identical. To avoid changing all these intervals again at some later barrier point we from now on treat them simultaneously. To this end we store identical CI's only once and associate a bag with it that contains all the points p for which $CI_p = CI$. 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 barrier points $b \in B_2$ efficiently, we must be able to determine all points in P_2 above the scan line that lie to the right of b 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 $CI_p := \emptyset$ for a point p to the right of b 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 the merge step in more detail:

5. Move a scanline downward over the set of points, halting at every point $(x, y) \in P \cup B$. (To do this we need a list of points $\in P \cup B$, sorted according to y-coordinate. This sorted list can be obtained from the sorted lists of $P_1 \cup B_1$ and $P_2 \cup B_2$ 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.) While we move the scanline we maintain the following two data structures:

   - A sorted list L of the different left endpoints of the connectability 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 connectability 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 connectable pair for each point p in these bags.

   $(x, y) \in B_1$: 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).

$(x, y) \in B_2$: 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 CI from L.)

Lemma 3.1 *The merge step can be performed in time $O(n \log n + k)$.*

Proof. Can be obtained by carefully charging costs to points. See [1] for details. □

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

Theorem 3.2 *All rectangular connectable pairs in a set of points P with respect to a set of barriers B can be computed in time $O(n \log^2 n + k)$, where $n = |P| + |B|$ and k is the number of answers.*

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 whether we can use another data structure that performs these operations more efficiently.

The crucial observation here is that we can normalize the problem (see e.g. [6]), i.e., convert it to the corresponding problem on a grid, without changing the connectable pairs. So we add as a preliminary step to the algorithm:

- Replace every point $(x, y) \in P \cup B$ 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.

This normalization maintains the connectability relation. The normalization step can be performed in time $O(n \log n)$ by sorting the points by x- and y-coordinate.

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., [10],[11]) 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 the merge step in $O(n \log \log n + k)$ time and we obtain the following improved result:

Theorem 3.3 *All rectangular connectable pairs in a set of points P with respect to a set of points B can be computed in time $O(n \log n \log \log n + k)$, where $n = |P| + |B|$ and k is the number of answers.*

The result can be extended in a number of ways. First of all, when the sets P and B have different sizes, we can tune the method to work more efficiently. Secondly, when $P \subseteq B$ the use of a VanEmdeBoas tree can be avoided completely, resulting in an optimal $O(n \log n + k)$ time bound. Finally, when B consists of more general obstacles than points, it can be shown that B can be reduced to a set of points, maintaining the connectable pairs. See [1] for details and results.

4 Homothetic connections.

In this section we study the case in which the set of obstacles consists of homothetic objects (i.e., the class consists of one object b and all object one can obtain by translating and scaling b). We will only consider the case in which $P = B$. Let us first consider circular connections, i.e., $\mathcal{C}$ is the collection of all disks.

Lemma 4.1 *Given a set of points P, $p \in P$ can be connected to $q \in P$ by a disk, with respect to P, if and only if in the Voronoi diagram of P the regions of p and q share an edge.*

Proof. $\Longleftarrow$ If the regions of p and q share an edge, let m be a point on this edge that is not a vertex of the Voronoi diagram. m has the same distance to p and q and all other points lie further away from m. Hence, the disk with m as center and $|mp|$ as radius does only contain p and q.

$\Longrightarrow$ If p and q are connectable, let m be the center of the connecting disk. Clearly m should be on a Voronoi edge between the regions of p and q because no other point can lie inside or on the boundary of the disk. $\square$

The lemma immediately shows that the number of such pairs in P is bounded by $O(n)$. We can now solve the connectability problem by first constructing the Voronoi diagram of the set of points P and next looking for each cell at it bordering cells. This immediately leads to the following result:

Theorem 4.2 *Given a set of points P, all pairs in P that can be connected with a disk with respect to P can be computed in time $O(n \log n)$.*

The method can be used for other types of homothetic connections as well. For example, when the connections are axis-parallel squares, one should use the Voronoi diagram with respect to the L_∞ metric. It is easy to show that the same result hold, i.e., two points can be connected by a square if and only if their Voronoi regions share an edge. This immediately leads to a $O(n \log n)$ solution for the problem. See [9] for more results on square connections.

To give a general result, let b be a convex object and let $\mathcal{C}$ consist of all object that are homothetic with b. In the terminology of Chew and Drysdale[3] b defines a convex distance function. In [3] it is shown that one can construct a Voronoi diagram based on a convex distance function in time $O(n \log n)$. It is easy to show that two points p and q can be connected by an object from $\mathcal{C}$ if and only if their regions in this Voronoi diagram share an edge. This leads to the following result.

Theorem 4.3 *Let $\mathcal{C}$ consist of all homothets of a convex object b in the plane. Let P be a set of points in the plane. All pairs $(p, q) \in P \times P$ such that $p \leadsto_{\mathcal{C}} q$ with respect to P can be determined in time $O(n \log n)$.*

5 More problems.

The notion of connectability studied so far can be generalized in a couple of ways.

Definition 5.1 *We call s and g* completely $\mathcal{C}$-connectable *with respect to B iff*

$$\forall_{p\in s, q\in g}\exists_{c\in\mathcal{C}}\, p\in c,\ q\in c \text{ and } \forall_{b\in B}\, b\cap c\subseteq s\cup g$$

In other words, for any pair of points in s and g there exists a connection.

Definition 5.2 *We call s and g* fully $\mathcal{C}$-connectable *with respect to B iff*

$$\exists_{c\in\mathcal{C}}\, s\cap c=s,\ g\cap c=g \text{ and } \forall_{b\in B}\, b\cap c\subseteq s\cup g$$

In other words, there exists a connection that covers both s and g. Both definitions are equal to the normal notion of connectability when $\mathcal{S}$ and $\mathcal{G}$ are sets of points only.

We can also define some other types of problems.

- When s and g are connectable but not completely connectable we might ask which parts of s and g are connectable. For example, in the hidden surface removal problem we have s being a point, $G = B$ a set of polyhedra and we ask which parts we can see from s, i.e., with what parts of the polyhedra can s be connected with a line segment.
- We could define a measure on the objects in $\mathcal{C}$. For example, when $\mathcal{C}$ contains line segments we could take their length. Now we can ask for such things as the smallest connection from a source to some goal or to all goals, for the largest connection, etc. In this way all kinds of shortest path problems fit in the scheme.
- You can of course extent the notion to connections between more than two objects. For example, which triples of points in a planar set can be covered by a triangle that does not contain any other point in the set.

We will not even make an attempt in solving all these various problems.

6 Conclusions.

In this paper we have given a large number of open problems. We have tried to define these problems in a general way, introducing the notion of connectability, that might lead to general solution that cover large classes of problems.

By means of example we have given a number of solutions that solve instances of the connectability problem. In particular the notions of rectangular connectability and connections with homothetic objects were studied.

We think it is worth studying connectability problems further and encourage people to do so.

References

[1] de Berg, M.T., and M.H. Overmars, Dominance in the presence of obstacles, Techn. Rep. RUU-CS-88-10, Dept of Computer Science, University of Utrecht, 1988.

[2] Edelsbrunner, H., and L. Guibas, Topological sweeping in an arrangement, *Proc. 18th Symp. Theory of Computing,* 1986, pp. 389-403.

[3] Chew, L.P., and R.L. Drysdale, III, Voronoi diagrams based on convex distance functions, *Proc. 1st ACM Symp. Computational Geometry,* 1985, pp. 235-244.

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

[5] 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.

[6] 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.

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

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

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

[10] 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.

[11] 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.