The 2-Center Problem with Obstacles*

Dan Halperin† Micha Sharir‡ Ken Goldberg§

August 21, 2000

Abstract

Given a set S of n points in the plane and a set O of pairwise disjoint simple polygons with a total of m edges, we wish to find two congruent disks of smallest radius whose union covers S and whose centers lie outside the polygons in O (referred to as *locational constraints* in facility location theory). We present an algorithm to solve this problem in randomized expected time $O(m\log^2(mn) + mn\log^2 n\log(mn))$. We also present an efficient approximation scheme that constructs, for a given $\varepsilon > 0$, two disks as above of radius at most $(1+\varepsilon)r^*$, where r^* is the optimal radius, in time $O(1/\varepsilon\log(1/\varepsilon)(m\log^2 m + n\log^2 n))$ or in randomized expected time $O(1/\varepsilon\log(1/\varepsilon)((m + n\log n)\log(mn)))$.

1 Introduction

For a set S of n points in the plane, the (standard) 2-center problem is to find two congruent disks of minimum radius that cover the points of S. This is a special case of the p-center problem, where the goal is to cover S with p congruent disks of minimum radius. When p is part of the input the problem is known to be NP-complete [14]. For $p = 1$ this is the "smallest enclosing disk" problem which can be solved in linear time [13]. The case $p = 2$ has been intensively studied, and after a series of papers presenting near-quadratic algorithms, Sharir presented an algorithm that solves the problem in $O(n\log^9 n)$ time [16]. The latter solution has been improved by Eppstein who solves the problem in randomized expected time $O(n\log^2 n)$ [8]. Another efficient variant has recently been proposed by Chan [2].

*Work on this paper by Ken Goldberg and Dan Halperin has been supported by a grant from the U.S.-Israeli Binational Science Foundation. Work by Dan halperin and Micha Sharir has been supported by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), by a Franco-Israeli research grant "factory of the future" (monitored by AFIRST/France and The Israeli Ministry of Science), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

†Department of Computer Science, Tel Aviv University, Tel-Aviv 69978, Israel. *halperin@math.tau.ac.il.*

‡School of Mathematical Sciences, Tel Aviv University, Tel-Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA. *sharir@math.tau.ac.il.*

§Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720. *goldberg@ieor.berkeley.edu.*

We study a variant of the 2-center problem, where in addition to the set S we are given a set O of pairwise disjoint simple polygons with a total of m edges, whose interiors constitute forbidden regions for placing the centers of the covering disks. (These obstacles are referred to as *locational constraints* in the standard theory of facility location [10].) We are not aware of existing efficient solutions to this problem. The analogous 1-center problem, namely the minimum enclosing disk problem with obstacles has been recently studied by Halperin and Linhart [9] who give an algorithm with running time $O((m+n)\log(mn))$ to solve it. They also provide an approximation algorithm for the problem and a publicly accessible Java applet that implements the approximation algorithm [9]. See Figure 1 for an illustration.

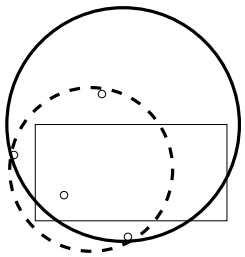

Figure 1: The minimum enclosing disk problem with and without obstacles. S consists of 4 points and O consists of one rectangle. The minimum enclosing disk without obstacles (whose center is inside the rectangle) is shown in dashed line and the minimum enclosing disk centered at a free location (on the top edge of the rectangle) is shown in bold line.

We present two algorithms to solve the 2-center problem with obstacles. The first algorithm follows the general approach of Sharir [16] for the standard 2-center problem, but it has to handle the presence of the polygons of O, which requires the introduction of several new and nontrivial techniques. A major innovation in our algorithm is a data structure that dynamically maintains a set P of points in the plane, under insertions and deletions of points, such that after each update we can efficiently (in polylogarithmic time) determine whether the intersection of fixed-radius congruent disks centered at the points of P and the complement F of the obstacles in O is nonempty (any point in this intersection can serve as the center of a 'free' disk that contains P). In contrast, the parallel procedure in [16] only has to determine whether the intersection of those disks is nonempty—a considerably simpler task. As in the improved solutions of Eppstein and Chan for the standard 2-center problem, we also aim to minimize the use of parametric searching in our algorithm, so that (a) we keep it simpler, (b) we improve its running time by a polylogarithmic factor, and (c) we gain more insight into the structure of the problem. To achieve this, we note that one can use an interplay between the standard 2-center problem and the one in the presence of obstacles. Informally, we show that, after various reductions and restrictions are imposed on the problem, one can solve it by solving the standard version of the (restricted) problem in which there are no obstacles. This will become clearer later on. The algorithm runs in randomized expected time $O(m\log^2(mn) + mn\log^2 n\log(mn))$.

Since the exact algorithm does not have a near-linear complexity, we develop a second algorithm that is an efficient approximation scheme for the problem. Given $\varepsilon > 0$ we find two congru-

ent disks that cover the points in S, whose centers lie outside the obstacles in O and whose radii are each at most $(1+\varepsilon)r^*$, where r^* is the optimal radius, in time $O(1/\varepsilon \log(1/\varepsilon)(m \log^2 m + n \log^2 n))$ or in randomized expected time $O(1/\varepsilon \log(1/\varepsilon)((m + n \log n) \log(mn)))$. The algorithm is driven by a variant of binary search for the optimal radius, within a range that depends on the diameter of S, and uses a simplified decision procedure for the search, which makes it more efficient. A variant of this algorithm also yields an approximation scheme for the standard 2-center problem, which is more efficient than the exact algorithm and is considerably simpler.

Discussing the introduction of obstacles in facility location problems, the authors in [10] argue that "such [locational] constraints are ubiquitous and important in practice". Our work however has been inspired by the following problem in robotics. Several CAD vendors are developing software for rapid setup of automated workcells [6]. A fundamental problem is where to place robot arms and other devices so that certain desired points can be reached. The distance the robot reaches for each point influences robot precision and settling time; it is therefore desirable to minimize this distance. Furthermore, operations such as inspection and assembly must be performed in tight quarters, so it is also necessary to position robots such that their base does not intersect with other devices in the workcell. The real-life (two) robot placement problem is complicated by many factors, and each factor raises new and more involved problems (e.g., the robot joints may have mechanical limits so that the robot's workspace is a disk-sector rather than a full disk). Nevertheless, we view our current work as a first step in approaching this family of problems.

In the next section we present the dynamic data structure to maintain the intersection of disks and free space. The exact algorithm is described in Section 3 and the approximation scheme is presented in Section 4. Direction for future work are proposed in Section 5.

2 Dynamic Maintenance of the Intersection of Disks and Free Space

In this section we describe a dynamic data structure which is a key ingredient of our solution to the 2-center problem with obstacles.

Let F denote the free space, namely the complement of O. We assume here that r is a fixed radius. The following notation is borrowed from [16]. $B_r(p)$ denotes the closed disk of radius r centered at p. For a set P of points, $K(P)$ denotes the intersection $\bigcap_{p \in P} B_r(p)$. We will maintain two structures $K^+(P) = \bigcap_{p \in P} B_r^+(p)$ and $K^-(P) = \bigcap_{p \in P} B_r^-(p)$, where $B_r^+(p)$ (resp. $B_r^-(p)$) is the region consisting of all the points that lie in or above (resp. in or below) $B_r(p)$.

Our goal is to dynamically maintain a set P of points in the plane, under insertions and deletions of points. After each update we need to determine whether the intersection $K(P) \cap F$ is nonempty. If this intersection is nonempty this means that we can cover the set P by a disk of radius r whose center lies outside O.

We prepare auxiliary static data structures. The first is for point location in the planar map induced by the polygons in O. For a query point q we determine with this data structure the polygon whose interior contains q, or otherwise that q lies in F (i.e., outside all polygons).

Construction of this standard data structure takes $O(m \log m)$ time, requires $O(m)$ space, and a point location query takes $O(\log m)$ time.

Secondly, we construct for each polygon a fixed-radius circular-shooting structure as described in [3]. We summarize its performance in the following theorem.

Theorem 2.1 (Cheng et al. [3]) *Let Q be a simple polygon with k edges. A data structure for fixed radius circular shooting inside Q can be constructed in $O(k \log k)$ time using $O(k)$ space, such that the first intersection point of a query arc of the given radius with the boundary of Q along the arc, if any, can be found in $O(\log k)$ time.*

The radius of all the circular arcs we are shooting is the fixed r we will be using throughout this section. By Theorem 2.1 the construction of these structures for all polygons in O together requires $O(m \log m)$ time, $O(m)$ space, and a query is answered in $O(\log m)$ time, once we are given the polygon containing the initial point of the query arc. Below we will always query with x-monotone arcs whose starting point is the leftmost point of the arc. For an x-monotone arc α, we will denote its leftmost point by $l(\alpha)$.

We denote all the static data structures collectively by $\mathcal{S}$, and we use them as follows. We query $\mathcal{S}$ with an x-monotone arc α. The answer is the leftmost point of $\alpha \cap F$, or NULL if α is completely contained inside a single polygon of O. We do this by first querying the point location structure with $l(\alpha)$. If $l(\alpha)$ lies in F we return $l(\alpha)$. Otherwise we obtain the polygon in which $l(\alpha)$ lies and we shoot with α in the respective circular-shooting data structure to obtain the desired answer. Thus, querying $\mathcal{S}$ with an arc takes $O(\log m)$ time.

The Overmars-van Leeuwen-like data structure developed in [16] maintains the intersections $K^+(P)$ and $K^-(P)$. The boundary of either intersection consists of a sequence of circular arcs, with *breakpoints* between each consecutive pair of arcs. We augment the structure describing $K^+(P)$ so that we can efficiently obtain the following information: for each breakpoint b on the boundary of $K^+(P)$ what is the leftmost point on the boundary whose x-coordinate is greater than or equal to that of b and that lies in F, or NULL if there is no such point. In other words we wish to determine for a breakpoint b what is the first free point that we will encounter when walking along the boundary from b to the right, if there is such a point. We consider the points on the boundaries of the polygons in O to be free. The same augmentation will be applied to $K^-(P)$.

Before we give details on how we augment the structures, we describe how we use them. Recall that we wish to determine whether $K(P) \cap F$ is nonempty. We first look for the left and right intersection points of the boundaries of $K^+(P)$ and $K^-(P)$—this can be done in $O(\log^2 n)$ time. If $K^+(P)$ and $K^-(P)$ do not intersect then we are done (the intersection $K(P)$ is empty). If they intersect, let ξ and η denote the left and right intersection points of their boundaries, respectively. See Figure 2. Let b be the first breakpoint along $\partial K^+(P)$ (the boundary of $K^+(P)$) to the right of ξ. Let α be the arc which is the portion of $\partial K^+(P)$ between ξ and b, or between ξ and η if b lies to the right of η. Assume that b lies to the left of η; we first have to find b which we do in $O(\log n)$ time (we defer the description of this simple operation to the full description of the data structure below). We query the static structure $\mathcal{S}$ with α, in $O(\log m)$ time. If we get a free point in return, then $K(P) \cap F$ is 'nonempty', and we are done. Otherwise α lies completely inside an obstacle.

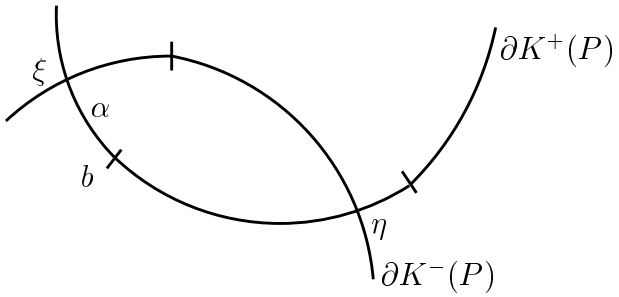

Figure 2: The arc α on the boundary of $K(P)$. The small dashes mark breakpoints.

Note that in this case, since our obstacles are simple polygons, $K(P) \cap F$ is nonempty if and only if there is a free point along the boundary of $K(P)$. We query the augmented dynamic structure with the breakpoint b to get the first free point to the right of b on $\partial K^+(P)$. As explained below, this can be done in $O(\log n)$ time. If the answer is a point lying to the left of or coinciding with η then again the answer to our original question is 'nonempty'. If in all the above queries we found no free point along $\partial K^+(P)$, we turn to the boundary of $K^-(P)$ and repeat the same sequence of operations from ξ to the right. If we find a free point along the portion of $\partial K^-(P)$ that bounds $K(P)$ then the answer to our question is 'nonempty', otherwise the answer is 'empty'. The overall time to decide whether $K(P) \cap F$ is nonempty is thus $O(\log^2 n + \log m)$.

Next we describe how to augment the dynamic data structure describing $K^+(P)$ so that it efficiently answers a query of the type: Given a breakpoint b along $\partial K^+(P)$, what is the first point on $\partial K^+(P)$ that is free and whose x-coordinate is greater than or equal to that of b, if one exists? The dynamization is under insertions and deletions of points to and from the set P, where all the points are from the original set S. The structure describing $K^-(P)$ and its augmentation are analogous.

We first recall the structure describing $K^+(P)$ [16]. We sort the points of S by their x-coordinates and store them, from left to right, in the leaves of a minimum-height binary tree T. Each leaf of T maintains a flag that indicates whether the point p of S associated with it is currently in P. To conform with the structure of internal nodes, each leaf stores the x-range of $B_r^+(p)$ if p belongs to the current set P, and stores the full x-axis otherwise. A node v of T (indirectly) maintains $K^+(P_v)$, where P_v is the subset of points of P stored at the leaves of the subtree of T rooted at v. Let w_l and w_r be the left and right children of v, respectively; then v stores the x-range of $K^+(P_v)$, which is the intersection of the x-ranges of $K^+(P_{w_l})$ and $K^+(P_{w_r})$. If the x-range of $K^+(P_v)$ is nonempty then we also store at v the single intersection point q_v between $\partial K^+(P_{w_l})$ and $\partial K^+(P_{w_r})$, with pointers to the pair of circles that intersect at q_v. For more details, see [16].

In the augmented structure T^*, at each node v, besides the intersection point q_v, we will also store the first free point $\phi = \phi(q_v)$ along $\partial K^+(P_v)$ such that $x(\phi) \geq x(q_v)$ or NULL if no such point exists. At a leaf corresponding to point p, the role of q_v will be played by the leftmost point l_p of the semicircle $\gamma(p)$ bounding $B_r^+(p)$, namely we will store the first free point $\phi = \phi(l_p)$

along $\gamma(p)$ such that $x(\phi) \geq x(l_p)$ or NULL if no such point exists.

We now describe how we compute, update and search for the information $\phi(q)$, where q is either the breakpoint stored at an internal node, or the leftmost point l_p in case of a leaf. We start with the computation of $\phi(q)$ for all the relevant nodes of T for a given initial set $P \subset S$, which we compute bottom-up. At a leaf associated with point p, we query the static structure $\mathcal{S}$ with the arc $\gamma(p)$ and store the answer at the leaf; this requires $O(\log m)$ time. Next we compute $\phi(q_v)$ for an internal node v with a left child w_l and a right child w_r, where for all nodes in the subtree rooted at v besides v we already have the information $\phi(q)$. As above, let q_v denote the intersection point of $\partial K^+(P_{w_l})$ and $\partial K^+(P_{w_r})$. Let b denote the next breakpoint when moving from q_v along $\partial K^+(P_v)$ rightwards (possibly the rightmost point of $\partial K^+(P_v)$). Next we show how to find b.

We are looking for the leftmost breakpoint of $\partial K^+(P_v)$ to the right of q_v. We search down the subtree rooted at v and maintain the leftmost breakpoint to the right of q_v that we have found so far, call it $\bar{b}$. We know that b is a breakpoint along $\partial K^+(P_{w_l})$ hence we start the search at w_l. At a node w along the search path (where originally $w := w_l$) we compare $x(q_w)$ and the interval $[x(q_v), x(\bar{b})]$. If $x(q_w)$ falls inside the interval then we let $\bar{b} := q_w$ and we continue to the right child of w. Else, if $x(q_w) > x(\bar{b})$ we move to the right child of w. Else (namely, $x(q_w) < x(q_v)$) we move to the left child of w. Once we have reached the end of the search path (a leaf), the final value of $\bar{b}$ is the desired breakpoint b. The cost of this search is $O(\log n)$.

We query $\mathcal{S}$ with the arc which is the portion of $\partial K^+(P_v)$ between q_v and b (in time $O(\log m)$). If the answer is *not* NULL then we store it at $\phi(q_v)$. Else, if b is the rightmost point of $\partial K^+(P_v)$ we store NULL at $\phi(q_v)$. Otherwise, namely if b is an internal breakpoint of $\partial K^+(P_v)$ and the answer to the query was NULL, we 'jump' to the node u containing the breakpoint b.

We denote the nodes along the search path from v to u in reverse order $v_0 = u, v_1, v_2, \ldots, v_k = v$. At the node v_0, the information $\phi(q_{v_0})$ is correct for $K^+(P_{v_0})$, namely $\phi(q_{v_0})$ is the first free point along $\partial K^+(P_{v_0})$ such that $x(\phi(q_{v_0})) \geq x(b)$, if such a point exists. We now move up from v_0 to v_k along the reverse of the search path and 'correct' this information so that when we reach v_k we will have the first free point ϕ along $\partial K^+(P_{v_k})$ such that $x(\phi) \geq x(b)$. We maintain an auxiliary variable ϕ'. At the beginning of the process we let $\phi' := \phi(q_{v_0})$. When we move from v_i to v_{i+1} we distinguish two cases (see Figure 3):

- The node v_i is a *left* child of v_{i+1}. Then ϕ' remains unchanged, since from b rightwards the boundary of $K^+(P_{v_{i+1}})$ is the same as the boundary of $K^+(P_{v_i})$.
- The node v_i is a *right* child of v_{i+1}. If ϕ' is NULL or $x(\phi') \geq x(q_{v_{i+1}})$, we put $\phi' := \phi(q_{v_{i+1}})$. Otherwise, ϕ' remains unchanged.

At the end of the process, we set $\phi(q_v) := \phi'$.

Lemma 2.2 *The overall construction time of the augmented structure T^* is $O(n \log(mn))$.*

Proof: At each of the $O(n)$ nodes of the tree we spend $O(\log n)$ time for searching the breakpoint b, and we do one circular shooting query at the cost of $O(\log m)$ time, for an overall time

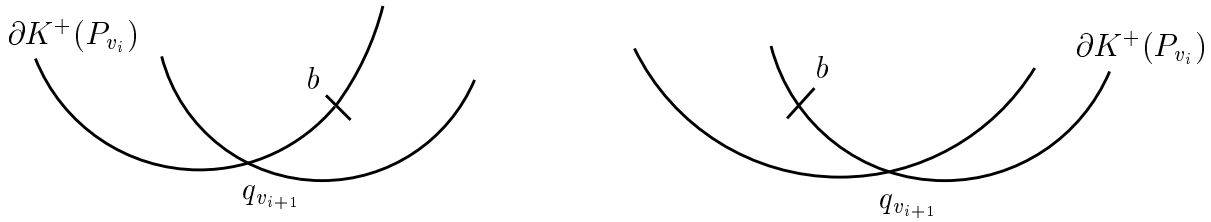

Figure 3: Moving up the search path: on the left v_i is the left child in which case the portion of the boundary to the right of b does not change when we move up; on the right v_i is a right child and $\phi(b)$ may be affected by the other child of v_{i+1}.

$O(n \log n + n \log m) = O(n \log(mn))$. □

We query T^* with a breakpoint b for the first free point ϕ along $K^+(P)$ such that $x(\phi) \geq x(b)$ in the same way that we compute $\phi(q_v)$ originally. Only this time v is the root of the tree. Namely, we first search the tree for the node containing the breakpoint b and then we move back up along the search path and maintain a variable ϕ' so that when we come back to the root we get in ϕ' the desired answer. Since we spend constant time at each node along the search path, the query time is $O(\log n)$.

It remains to show how we update T^* when a point p is inserted to or deleted from P. We first search the tree for the leaf corresponding to p. It is easily seen that we need to update the information $\phi(q_v)$ only at nodes v along this search path $\pi(p)$, and we update it bottom up. As we reach a node v along $\pi(p)$, we need to update both the intersection point q_v and the information $\phi(q_v)$. The latter may require circular shooting from the new q_v, and computing the information $\phi(b)$ at the breakpoint b to the right of q_v, which in turn requires searching for b in the subtree rooted at v and going back up along this 'local' search path to get the desired output. This means that at each of the $O(\log n)$ nodes along $\pi(p)$ we spend $O(\log n)$ time for the search and $O(\log m)$ time for circular shooting, for a total of $O(\log^2 n + \log n \log m) = O(\log n \log(mn))$ time per update.

Note that the resources required by the extra information (the $\phi(q_v)$'s) in T^* dominate the resources required by the original dynamic structure T [15], [16]. We conclude:

Theorem 2.3 *Given a set of points S and a set of polygonal obstacles O as above, we can construct a dynamic data structure to determine whether $K(P) \cap F$ is nonempty, where F is the complement of the obstacles and $P \subset S$ is the current active set of points. Each such test takes $O(\log^2 n + \log m)$ time. The construction of the structure takes $O(m \log m + n \log(mn))$ time. An update of the structure when a single point is inserted to or deleted from P takes $O(\log n \log(mn))$ time. The space required by the dynamic structure is $O(n)$ and additional $O(m)$ space is required by the auxiliary static structure $\mathcal{S}$.*

In our algorithm we will also need to report whether there is a solution with radius smaller than the given r. To this end, we augment our dynamic structure as follows. When we obtain a

positive answer to the intersection query, the structure in fact finds a "witness" point w in the intersection $K(P) \cap F$. We check in $O(1)$ time if locally w is an isolated point of the intersection; if the answer is no, then we report that there is a solution with radius smaller than the given r—this is justified by Lemma 3.1 below. Otherwise, if w is isolated and it is the rightmost point η of $K(P)$ there is no solution with smaller radius. Finally, if w is an isolated point which is not the rightmost point of the intersection, we continue querying our data structure as above from w (i.e., we shoot a circular ray from w along the boundary of $K(P)$, etc.). The general position assumption, that will be made in the following section, assures us that if we get another free point along the boundary of $K(P)$ then this cannot be an isolated point, in which case we report that there is a solution with a smaller radius.

Corollary 2.4 *The structure of Theorem 2.3 can also report whether there is a solution with radius smaller than the given r, within the same resource bounds.*

3 The Overall Exact Algorithm

The overall exact algorithm follows closely the technique of [16] and its enhancements in [2, 8] for the standard 2-center problem, but it has to handle additional configurations that can arise because of the existence of obstacles.

3.1 Preliminaries

Lemma 3.1 *Let P be a finite point set and F a closed polygonal free region. Let D be a smallest disk that contains P and is centered at F. If D is centered at $int(F)$ then it is the (unconstrained) smallest enclosing disk of P.*

Proof: We use the following well-known approach. Parametrize the space of all disks in the plane so that a disk centered at (x, y) and having radius r is represented by the triple $(x, y, (r^2 - x^2 - y^2))$. The set of disks that contain a point (a, b) is the halfspace $z \geq -2ax - 2by + (a^2 + b^2)$. Hence the set of disks that contain P is a convex polyhedron K formed by the intersection of $|P|$ such halfspaces, one for each point of P. An (unconstrained) smallest enclosing disk of P is a point of K that minimizes the convex function $x^2 + y^2 + z$, and hence it is unique. Any other point of K can be moved continuously within K in a direction where $x^2 + y^2 + z$ strictly decreases.

Now if D is a disk as in the lemma, and is not equal to the (unconstrained) smallest enclosing disk of P, then D is represented by a point in K which can thus be moved slightly within K so that $x^2 + y^2 + z$ decreases. In other words, we can slightly shift and *shrink* D so that its center remains in F and it continues to contain P. This contradicts the assumed minimality of D and thus establishes the lemma. □

Lemma 3.2 *Let S and F be as above. Let Δ denote the diameter of S. If the radius of the optimal solution of the two-center problem with obstacles is larger than Δ then at least one of the centers must lie on an edge of F.*

Proof: Since S can be enclosed in a disk of radius Δ, it follows that the radius of the (unconstrained) smallest enclosing disk of any subset of S is at most Δ. Let D_1, D_2 be the two disks of an optimal solution of the two-center problem with obstacles for S and O. It follows that D_1 is not the (unconstrained) smallest disk enclosing $S \cap D_1$, and similarly for D_2. If the centers of both D_1 and D_2 lie in $int(F)$ then Lemma 3.1 implies that we can shrink both of them and obtain a solution with a smaller radius, contrary to assumption. □

Lemma 3.3 (Eppstein [7]) *Let S be a set of n points in the plane and let $S_1, S_2, \ldots, S_k$ be a sequence of subsets of S such that for each $i < m$, S_{i+1} is obtained from S_i by inserting or deleting a single point. Then the smallest enclosing disks of the sets S_i can all be computed in overall randomized expected time $O((n+k)\log^2 n \log^2 \log n)$.*

In what follows, we assume that S and O are in *general position*. Loosely speaking, this means that we rule out any coincidence between various unrelated quantities that are defined in terms of S and O and that would not coincide for randomly chosen values of the parameters that specify S and O. For example, we assume that the radii of disks passing through three points of S or having two points of S as a diameter are all distinct. Several additional requirements of this sort will be noted when they arise in the analysis (one of them has already been made at the end of the preceding section). The algorithms can also handle inputs in degenerate position, using a variety of known techniques (such as symbolic perturbations).

3.2 Centers lying on obstacle edges

The first (and new) stage of the algorithm aims to compute the smallest radius r_1 such that S can be covered by two disks of radius r_1 so that at least one of them is centered at a point of ∂F (and the other at any point of F).

To this end, we consider the following subproblem: Let e be a fixed edge of F. For simplicity, assume that e is the unit interval $[0,1]$ along the x-axis. Let S be the given set of points. We want to find the smallest r such that S can be covered by two disks of radius r, so that one of them is centered at a point of e and the other is centered at a point of F.

Enumerate the points of S as $(s_1, \ldots, s_n)$, where the coordinates of s_i are (a_i, b_i), for $i = 1, \ldots, n$. For each i, define a function f_i on e, parametrized by $\xi \in [0,1]$, as follows:

$$f_i(\xi) = d^2((\xi, 0), s_i) = (\xi - a_i)^2 + b_i^2 = \xi^2 - 2a_i\xi + (a_i^2 + b_i^2).$$

For simplicity, we will regard these functions as defined over the entire ξ-axis.

Consider the arrangement $\mathcal{A}(\mathcal{F})$ of the set $\mathcal{F}$ of the graphs of the functions f_i, represented in a coordinate frame (ξ, η). We will regard each point (ξ, η) in this frame as representing the disk $D(\xi, \eta) = B_{\sqrt{\eta}}((\xi, 0))$. Clearly, a point s_i lies in $D(\xi, \eta)$ if and only if its associated function graph f_i lies below the point (ξ, η). Let $S(\xi, \eta)$ denote the subset of S consisting of those points that lie outside $D(\xi, \eta)$ (i.e., points whose associated function graphs lie above (ξ, η)).

Lemma 3.4 *(a) Each pair of functions f_i, f_j intersect at most once.*

(b) A pair of functions f_i, f_j intersect below the horizontal line $\eta = \eta_0$ if and only if the pair of ξ-ordinates where $f_i(\xi) = \eta_0$ and the pair where $f_j(\xi) = \eta_0$ define two intervals (denoted as $I_i(\eta_0)$ and $I_j(\eta_0)$) that are overlapping (i.e., neither disjoint nor nested).

(c) The number of vertices of $\mathcal{A}(\mathcal{F})$ that lie below a given horizontal line $\eta = \eta_0$ can be counted in $O(n \log n)$ time.

(d) Given any number $0 \leq k \leq \binom{n}{2}$, one can find the k-th highest vertex of $\mathcal{A}(\mathcal{F})$ in $O(n \log^2 n)$ time.

Proof: The claim in (a) is immediate. Concerning (b), if $I_i(\eta_0)$ and $I_j(\eta_0)$ are overlapping then it is clear, using a continuity argument, that f_i and f_j intersect below $\eta = \eta_0$. Conversely, if f_i and f_j intersect below $\eta = \eta_0$ (at a unique point, whose ξ-ordinate is denoted as ξ_{ij}), then $I_i(\eta_0)$ and $I_j(\eta_0)$ cannot be disjoint (they both contain ξ_{ij}) and they cannot be nested either, for this would force the two graphs to intersect twice. Hence, counting the number of intersections below $\eta = \eta_0$ is equivalent to counting the number of pairs of overlapping intervals in a given system of n intervals on a line. This can be done in time $O(n \log n)$ using, e.g., a standard tree-based algorithm for counting inversions in a permutation (see, e.g., [5]). Indeed, sort the left endpoints of the given intervals in increasing order, and sort the right endpoints in decreasing order. Regarding the first permutation as the identity, the number of inversions in the second permutation is equal to the number of overlapping pairs of intervals. This establishes (c). The final assertion (d) follows by using an appropriately modified variant of the algorithm of [5] for the slope selection problem. Specifically, this algorithm applies parametric searching to the inversion counting algorithm. The simplest version runs in $O(n \log^3 n)$ time, which is reduced to $O(n \log^2 n)$ time using an enhancement technique due to Cole [4]. Both of these approaches are applicable in our case too. (The final improvement in the algorithm of [5], which reduces its complexity to $O(n \log n)$, does not seem to be applicable in our case, and has no effect anyway on the overall asymptotic bound on the running time of the algorithm.) $\Box$

The first stage of the algorithm proceeds through the following steps.

(i) Maintain a horizontal slab $\sigma : \ \eta^- \leq \eta \leq \eta^+$. Initially, this is the whole $\xi\eta$-plane.

(ii) Find a horizontal line $\gamma : \ \eta = \eta_0$ that bisects the subset of vertices of $\mathcal{A}(\mathcal{F})$ that lie in σ.

(iii) Determine whether there exists a point $(\xi, \eta_0) \in \gamma$ such that $S(\xi, \eta_0)$ can be covered by a disk of radius $\rho_0 = \sqrt{\eta_0}$ centered at F. (To accomplish this step, we intersect e with the circles bounding the disks $B_{\rho_0}(s_i)$, for $i = 1, \ldots, n$, sort these intersections along e, and iterate over these points in sorted order. Whenever we pass from one point to the next, a single element is added to or deleted from $S(\xi, \eta_0)$, and we use the dynamic data structure of the previous section to determine whether the new set can be covered by a disk of radius ρ_0 centered at F.) If such a point was found, replace σ by its portion below γ; otherwise replace it by its portion above γ.

(iv) Repeat these steps until σ contains no vertex of $\mathcal{A}(\mathcal{F})$ in its interior. Suppose that this final σ is $\eta_1 \leq \eta \leq \eta_2$. This means that step (iii) was successful at η_2 and failed at η_1. The line $\eta = \eta_2$ contains a single vertex v of $\mathcal{A}(\mathcal{F})$, induced by two points $p, q \in S$. We examine all the solutions found when step (iii) processed $\eta = \eta_2$. Two subcases can arise:

(iv.a) v is the only possible center along this line. Assuming general position, the radius of the smallest (constrained) disk enclosing $S(v)$ is strictly smaller than $\rho_2 = \sqrt{\eta_2}$. This is easily seen to imply that there is no disk of radius smaller than ρ_2 and centered at e that contains both p and q (for otherwise $\eta = \eta_2$ would have contained other solutions in the vicinity of v). This in turn implies that the solution with the minimum radius and with one center lying on e is at v itself. Hence in this case we return $D(v)$ and the sibling disk that covers $S(v)$.

(iv.b) There are other solutions along $\eta = \eta_2$. This means that any of the subsets of S that the disk $D(\xi, \eta_2)$ contains, as its center moves along e, with the possible exception of the subset defined by $D(v)$, are also contained by smaller disks centered at e; in fact, the radii of these smaller disks can be made (at least) as small as $\rho_1 = \sqrt{\eta_1}$; see Figure 4 for an illustration. Since step (iii) failed at η_1, this can only be because the smallest (constrained) disks containing the complementary sets $S(\xi, \eta_2)$ all have radii strictly greater than ρ_1. In view of Lemma 3.1, this implies that, for each such set $S(\xi, \eta_2)$, either the radius of its (unconstrained) smallest enclosing disk is greater than ρ_1, or the radius of its constrained smallest enclosing disk is greater than ρ_1 and this disk is centered at an edge e' of F. We can ignore the latter kind of situation because, as is easily seen, it will be detected in subcase (iv.a) when e' is processed. (More precisely, either it will be detected, or a solution with a smaller radius will be found.)

(v) We thus proceed as follows. The data gathered so far implies that the sequence of critical events defined in step (iii) is combinatorially the same for all horizontal lines $\eta = \eta_0$ through σ. We apply the algorithm in Lemma 3.3 to the fixed sequence of subsets $S(\xi, \eta_0)$ of S that arise in step (iii) (and is independent of η_0), to obtain the (unconstrained) smallest enclosing disk of each of these subsets. From among those disks whose centers lie in F, we take the smallest one, and return this disk and the corresponding disk $D(\xi, \eta_1)$. Note that the largest of the radii of these two disks is between ρ_1 and ρ_2 (for otherwise step (iii) would not have failed at $\eta = \eta_1$).

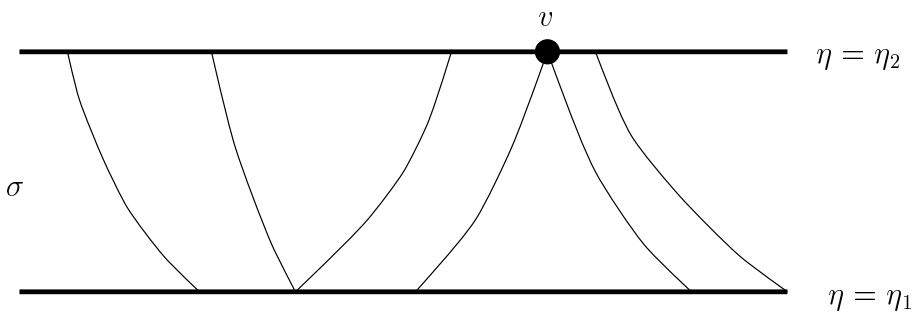

Figure 4: The arrangement $\mathcal{A}(\mathcal{F})$ within the final slab σ

We apply the procedure to each of the m edges of F, and output the solution with the smallest radius. The overall (expected) running time of this procedure is $O(mn \log^2 n \log(mn))$

(note that the only randomized part of the algorithm is Eppstein's procedure used in step (v)). Its correctness follows from the analysis given above.

In conclusion, we can, in $O(mn \log^2 n \log(mn))$ time, find the smallest radius r_1 such that S can be covered by two disks of radius r_1 so that at least one of them is centered at a point of ∂F (and the other is centered somewhere in F). From now on, our goal is to determine whether there exist a pair of congruent disks of radius smaller than r_1 which are both centered at points in the interior of F and whose union covers S. Recall that, by Lemma 3.1, each of these disks D is the (unconstrained) smallest disk containing $D \cap S$. Recall also that, by Lemma 3.2, we have $r_1 \leq \Delta$.

Remark: This stage is the only part of the algorithm that requires more than near-linear time. We leave it as an open problem to find a subquadratic implementation of this stage.

We now proceed following the general approach of [16]. It treats separately three subcases: (a) the case where the disks in an optimal solution are far apart (the distance between their centers is at least $3r^*$, where r^* is their common radius); (b) the case where the disks are 'nearly concentric' (the distance between their centers is smaller than r^*); and (c) the case where the disks are 'nearly tangent' (the distance between their centers is between r^* and $3r^*$). We consider the same three cases, and show how to replace or modify the procedures used in the previous papers in order to handle the presence of obstacles.

3.3 The case where the disks are far apart

We next compute the smallest $r^* < r_1$ such that S can be covered by two disks of radius r^* that are centered at points of $int(F)$ and the distance between their centers is at least $3r^*$.

Suppose that there exist a pair D_1, D_2 of disks of radius $r^* < r_1$ so that their union covers S and their respective centers c_1, c_2 satisfy $|c_1c_2| \geq 3r^*$. Let Δ denote the diameter of S. Clearly, we have $\Delta \leq |c_1c_2| + 2r^* \leq 5/3|c_1c_2|$. On the other hand, $|c_1c_2| - 2r^* \leq \Delta$, which implies that $|c_1c_2| \leq 3\Delta$ and $r^* \leq \Delta$ (the latter inequality is of course also a consequence of Lemma 3.2). Suppose that we guess, as in [16], that the orientation of c_1c_2 is nearly horizontal. Project S onto the x-axis. The span of this projection is at most Δ, and c_1 and c_2 project to two points, each lying at most $r^* \leq \Delta$ away from this span. Hence, we can construct an interval on the x-axis, of length at most 3Δ, which contains the projections of both c_1 and c_2. Since $|c_1c_2| \geq 0.6\Delta$, it follows that we can construct a constant number of vertical lines (spaced $\alpha\Delta$ apart, for some sufficiently small constant $\alpha > 0$), such that at least one of them is guaranteed to separate D_1 and D_2.

The current stage of the algorithm is thus straightforward: Construct a set of $O(1)$ candidate separating lines (a constant number of lines for a constant number of orientations). For each of these lines ℓ, obtain the partition of S into two subsets induced by ℓ, and compute the (unconstrained) smallest enclosing disk for each of the two subsets (this can be done in $O(n)$ time). Discard any output in which one of the radii of the two enclosing disks is larger than or equal to r_1 or one of the centers is not in F. For any surviving output, take the largest of the two radii, and return the solution which minimizes this value.

The correctness of this procedure is an immediate consequence of Lemma 3.1. Specifically,

suppose that the optimal solution (in the presence of obstacles) is a pair of disks D_1, D_2, which are well-separated in the above sense, whose common radius r^* is smaller than r_1, and which are centered at points of $int(F)$. (If there is no such solution, it is clear that the procedure will not report any solution.) As argued, one of the lines that the procedure processes separates D_1 and D_2, and thus separates the sets $S_1 = S \cap D_1$, $S_2 = S \cap D_2$. Let D_1' (resp. D_2') denote the smallest (unconstrained) disk enclosing S_1 (resp. S_2). By Lemma 3.1, either $D_1 = D_1'$ or it can be shrunk continuously while continuing to contain S_1. This shrinking process must eventually reach D_1', for otherwise the center of the shrinking disk would reach ∂F and this would contradict the definition of r_1. A similar argument applies to D_2, and thus establishes the correctness of this procedure. The overall cost of this step is $O(n + \log m)$.

3.4 The case where the disks are nearly concentric

In this subsection we handle the case in which the optimum solution consists of two disks D_1, D_2, that are centered at points $c_1, c_2 \in F$, have common radius $r^* < r_1 \leq \Delta$ and satisfy $|c_1c_2| \leq r^*$. In this case we proceed in a manner similar to that in [2, 8, 16]. We briefly describe the approach, and refer the reader to these papers for more details. Specifically, in this case there is a large overlap between the disks, and we can guess a point z, out of a set of $O(1)$ candidate points, that lies in $D_1 \cap D_2$. We can also guess an orientation, out of $O(1)$ possible ones, so that the line ℓ passing through z at that orientation separates the two intersection points of ∂D_1 and ∂D_2. Without loss of generality, assume ℓ to be horizontal.

Let S^+ (resp. S^-) be the subset of the points of S that lie above (resp. below) ℓ. Sort the points of S^+ (resp. of S^-) in counterclockwise (resp. clockwise) angular order about z. Define a matrix structure M whose rows (resp. columns) correspond to the points of S^+ (resp. of S^-) in sorted order. The (i, j)-th entry of M represents the partition of S into a left subset $S_L(i, j)$ and a right subset $S_R(i, j)$, separated by the ray emanating from z upwards and passing between the i-th and $(i + 1)$-st points of S^+ and the ray emanating from z downwards and passing between the j-th and $(j + 1)$-st points of S^-. Clearly, the two disks D_1, D_2 of the optimum solution satisfy $S_L(i, j) \subset D_1$ and $S_R(i, j) \subset D_2$, for the entry (i, j) induced by the two rays that emanate from z towards the two points of intersection of ∂D_1 and ∂D_2. See Figure 5 for an illustration.

For each (i, j), we associate four values with the (i, j)-th entry of this structure:

- $r_L(i, j)$ = radius of the smallest disk that encloses $S_L(i, j)$ and is centered at a point of F;
- $r_R(i, j)$ = radius of the smallest disk that encloses $S_R(i, j)$ and is centered at a point of F;
- $r'_L(i, j)$ = radius of the (unconstrained) smallest disk enclosing $S_L(i, j)$;
- $r'_R(i, j)$ = radius of the (unconstrained) smallest disk enclosing $S_R(i, j)$.

Note that all four resulting matrices are monotone. Specifically:

- $r_L(i_1, j_1) \geq r_L(i_2, j_2)$ for $i_1 \leq i_2$, $j_1 \leq j_2$.
- $r_R(i_1, j_1) \leq r_R(i_2, j_2)$ for $i_1 \leq i_2$, $j_1 \leq j_2$.

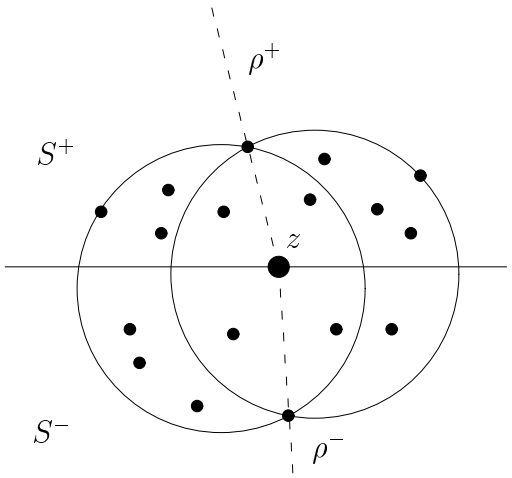

Figure 5: The case where $|c_1c_2| < r$

- $r'_L(i_1, j_1) \geq r'_L(i_2, j_2)$ for $i_1 \leq i_2$, $j_1 \leq j_2$.
- $r'_R(i_1, j_1) \leq r'_R(i_2, j_2)$ for $i_1 \leq i_2$, $j_1 \leq j_2$.

We first run a matrix searching step that computes all entries (i, j) for which $r_L(i, j) < r_1$ and $r_R(i, j) < r_1$. This is done as in [16], by tracing two monotone paths through the structure, scanning in total a linear number of entries, and using the dynamic data structure of Section 2 to navigate through the matrix. (As noted in that section, the data structure is also able, under the general position assumption, to detect the case where there is a strict inequality between the optimal solution and the given r_1.) The output of this phase is a collection of matrix substructures $M_1, \ldots, M_t$ that have pairwise-disjoint sets of rows and of columns and are arranged in row-increasing and column-decreasing order; see Figure 6 for an illustration.

We now solve within these submatrices the unconstrained two-center problem, using the techniques of Eppstein [8] or Chan [2]. We note that these techniques treat the matrices in a fully abstract manner, and do not rely on any specific geometric structure. All that they require is that the matrices be monotone, as above, and that there are 'black-box' routines that return any specific value $r'_L(i, j)$ or $r'_R(i, j)$, or that compare any of these values with any given r (where the latter operation is cheaper than the former one). We modify the r'_L and r'_R matrices as follows: Connect all the substructures $M_1, \ldots, M_t$ by a sequence of row-increasing and column-decreasing paths. For any entry (i, j) that lies above this path and outside the matrices M_q, we put $r'_L(i, j) = +\infty$, $r'_R(i, j) = 0$, and for any entry (i, j) that lies below this path and outside the matrices M_q, we put $r'_L(i, j) = 0$, $r'_R(i, j) = +\infty$. The values of $r'_L(i, j)$ and $r'_R(i, j)$ remain unchanged within each of the M_q submatrices. See Figure 6. It is easy to see that the new matrices remain monotone, and that the black-box routines mentioned above retain their asymptotic complexity for the modified matrices (we simply augment the old routines with an initial binary search step that determines whether the query entry lies in one of the matrices M_q, above the connecting paths, or below the paths).

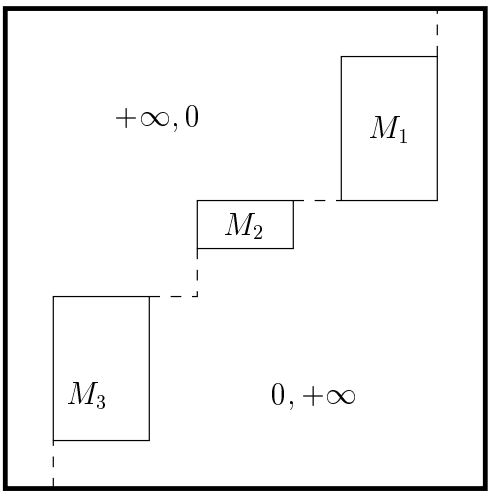

Figure 6: The matrix substructures M_q

Suppose that the optimal solution (to the unconstrained problem) is attained at entry (i, j) (because of the way we have modified the matrices, this entry must belong to one of the submatrices M_q). We claim that it must coincide with the optimal solution in the presence of obstacles. In other words, we claim that the smallest disks D_L, D_R enclosing, respectively, $S_L(i, j)$, $S_R(i, j)$ are centered at points of F. This is argued as in the previous steps of the algorithm, using Lemma 3.1. Specifically, suppose to the contrary that, say, D_L is centered at a point of O. Since (i, j) belongs to M_q, we know that there exists another disk D_L^* that contains $S_L(i, j)$, is centered at a point of $int(F)$, and has radius smaller than r_1. By Lemma 3.1, since D_L^* is not the smallest enclosing disk of $S_L(i, j)$, we can shrink it continuously, while keeping $S_L(i, j)$ contained in it, until its center reaches ∂F. This is easily seen to contradict the definition of r_1, and thus implies the claim.

The overall cost of this procedure is thus (see [2])

$$O(n \log n \log(mn) + n \log^2 n \log^2 \log n).$$

(The second term reduces to $O(n \log n)$ if randomization is allowed [2].) After running this procedure, and the preceding ones, we obtain a threshold radius $r_2 \leq r_1$, which is the smallest radius for which there exists a solution to the two-center problem with obstacles in which the common radius of the disks is r_2 and either at least one center lies on ∂F or the centers are far apart or there is a large overlap between the disks.

In the remainder of the algorithm, we seek a solution (if one exists) where the radius r^* is smaller than r_2 and where the distance between the centers is between r^* and $3r^*$. Handling this 'nearly-tangent' situation is the most involved stage of the algorithm.

3.5 The NT2DC decision procedure

To facilitate the handling of the nearly-tangent case by the algorithm, we first describe a decision procedure that will be used in the algorithm. We are given S and F, as above, and a radius $r < r_2$. The goal is to determine whether there exist two congruent disks of common radius r which are centered at $int(F)$, whose union covers S, and where the distance between the centers c_1, c_2 is between r and $3r$. We refer to this subproblem as the *nearly-tangent two-disk covering* problem, or the NT2DC problem for short.

We follow exactly the same technique as in [16]. That is, we guess, in a constant number of possibilities, an approximate orientation of the directed line c_1c_2 (henceforth assumed to be the positive horizontal orientation) and a (vertical) line ℓ that separates the left center c_1 from the two intersection points of the disks (if these points exist) or from the leftmost point of the right disk, otherwise. See Figure 7 for an illustration.

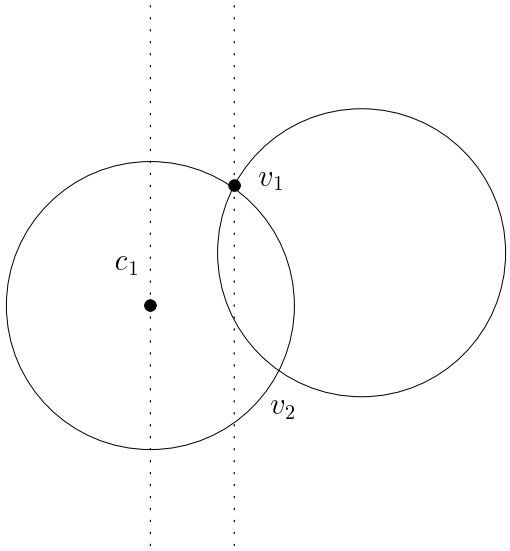

Figure 7: The case $r < |c_1c_2| \leq 3r$

Let S_L denote the subset of the points of S to the left of ℓ. Form the intersection $K_r(S_L)$ of all the disks $B_r(p)$, for $p \in S_L$, and intersect all the other disk boundaries $B_r(q)$, and all the edges of F, with the right boundary Γ_r of $K_r(S_L)$. We obtain a collection of $O(m+n)$ *critical points* along Γ_r, which can be constructed and sorted along Γ_r in $O((m+n)\log(m+n))$ time (this follows from the observation that each disk boundary and each edge of F crosses Γ_r at most twice). We now trace the center c_1 of the left disk along Γ_r, maintaining the set S_2 of points of S not contained in $B_r(c_1)$. As c_1 moves past a critical point, S_2 changes by at most one point that is being added or deleted, or, if the critical point lies on an edge of F, the center c_1 enters or leaves F. Using our data structure, we determine, after each such transition, whether S_2 can be covered by a disk of radius r that is centered at F. We consider as valid only cases where the portion of Γ_r that c_1 currently traces is contained in F; the critical intersection points

of the edges of ∂F with Γ_r help us to keep track of this property.

The cost of this NT2DC decision procedure is $O((m + n \log n) \log(mn))$. Note that in this procedure we assume that the left disk determines r^*. The case where the right disk determines r^* will be handled when we process another orientation for which c_1c_2 is nearly horizontal and c_1 lies to the right of c_2.

3.6 The case where the disks are nearly-tangent

Next assume that the optimal solution satisfies $r^* < |c_1c_2| \leq 3r^*$ and $r^* < r_2$. To detect this case we use a limited amount of parametric searching, applied to the NT2DC procedure that we have just presented. This is done as follows.

We first construct the farthest-neighbor Voronoi diagram $VOR_F(S_L)$ of S_L. For each vertex v of the diagram we compute (in $O(1)$ time) the radius of the smallest disk centered at v and containing S_L. Similarly, for each edge e of the diagram we compute (again, in $O(1)$ time) the radius of the smallest disk centered on e and containing S_L. We obtain a list of $O(n)$ critical radii (as a matter of fact, we keep in the list only critical radii that are smaller than r_2), and we conduct binary search over that list, using the NT2DC decision procedure as a discriminator for the search. We obtain a range I that is delimited by two consecutive critical radii and contains r^*. For any $r \in I$, the combinatorial structure of the right boundary Γ_r of $K_r(S_L)$ is the same, as follows easily from the structure and properties of the farthest-neighbor Voronoi diagram. The cost of this step is $O((m + n \log n) \log n \log(mn))$.

In the next stage of the parametric search, we take each edge of F and each circle $\partial B_{r^*}(p)$, for $p \notin S_L$, and attempt to locate their intersection points with Γ_{r^*} along this curve (i.e., identify the arc of Γ_{r^*} that contains each of these points). Intersecting an edge e of ∂F with Γ_{r^*} amounts to intersecting the line containing e with this curve and determining whether the endpoints of e lie inside or outside $K_{r^*}(S_L)$. Each of these two operations is easy to accomplish using binary search on the breakpoints of Γ_{r^*}. Each such breakpoint is a point on some edge of $VOR_F(S_L)$ at distance r^* from the two sites defining the edge. It is easy to define the critical values of r^* where the output to a query that seeks the position of such a breakpoint relative to a given line or point may change. In a similar manner we can conduct an implicit comparison between a breakpoint of Γ_{r^*} and a query circle of radius r^*. We now execute the generic simulation of all these $m + n$ queries in parallel, applying the parametric searching paradigm at each parallel step. The cost of a single parallel step is

$$O\left((\log(m + n)) \cdot (m + n \log n) \log(mn)\right),$$

and since there are $O(\log n)$ parallel steps, the overall cost of this step is $O((m+n \log n) \log n \log^2(mn))$. We can improve this further, by a factor of $O(\log n)$, by using the technique of Cole [4]. This technique is applicable when the parallel execution can be simulated on a network with bounded fan-out, and this property holds for our algorithm, which is just a collection of binary searches.

At this stage, we have limited the range for r^* further, and for any r in the new range, we know the combinatorial structure of Γ_r, as well as all the critical intersection points of disk boundaries and edges of F along each of the arcs of Γ_r. We still need to sort these points along

each arc, which we do, in a generic simulation mode, in a manner similar to that described in the preceding paragraph. We omit the straightforward details. The cost of this step is asymptotically the same as that of the preceding one (and Cole's improvement is applicable here as well).

At this point we know the exact sequence of critical points along Γ_{r^*}. This allows us to construct explicitly a sequence of $O(m+n)$ bipartitions of S into a left subset S_1 and a right subset S_2, each obtained by placing a disk of radius r^* centered between a pair of consecutive critical points along Γ_{r^*} and by defining the corresponding S_1 (resp. S_2) to be the set of points of S that lie inside (resp. outside) that disk.

Lemma 3.5 *In the specific subcase under consideration, the optimal radius r^* is the upper endpoint of the current range (which possibly consists of a single point) produced by the preceding parametric searching steps.*

Proof: Let D_1 be the disk of the optimal solution whose center c_1 lies on Γ_{r^*}. Since $r^* < r_1$, we know that D_1 is the (unconstrained) smallest enclosing disk of $S_1 = D_1 \cap S$. Hence the boundary ∂D_1 contains either three points of S_1 or two diametrically-opposite points of S_1. By construction, at least one of these points belongs to S_L. Consider the case where exactly one of these points, p, belongs to S_L, which means that c_1 lies in the relative interior of the arc γ of $\partial B_{r^*}(p)$ that appears along Γ_{r^*}. It follows that either there exists another point $q \notin S_L$ such that $\partial B_{r^*}(q)$ is tangent to γ (at c_1), or there exist two points $q, s \notin S_L$ such that $\partial B_{r^*}(q)$ and $\partial B_{r^*}(s)$ intersect γ at a common point (namely, at c_1). However, each of these cases causes a discrete change in the sequence of critical points along Γ_r, as r varies through r^*, and thus r^* will be a critical value in one of the generic comparisons that one of the preceding steps makes. Similar reasoning applies when ∂D_1 contains two points of S_L and one point in the complementary set: In this case c_1 is a breakpoint of Γ_{r^*} and a circle $\partial B_{r^*}(q)$, for some $q \notin S_L$, passes through c_1. Again, this is an event that causes a discrete change in the sequence of critical points and so will also be detected. Finally, the case where ∂D_1 contains only (two or three) points of S_L will be detected during the initial stage of the procedure that uses the farthest-neighbor Voronoi diagram, since in this case r^* is one of the critical values computed at that stage. $\Box$

Finally, we go over the sequence of bipartitions and check, for each bipartition, whether both subsets S_1, S_2 can be covered by a disk of radius r^* and centered in F. This is easy to do, by simply scanning through the sequence, maintaining dynamically the two corresponding subsets S_1, S_2, and performing these checks using our data structure. The cost of this step is only $O((m+n)\log n \log(mn))$. We return the first partition, and the two covering disks. If the processing of the current guess for ℓ has reached this point, the preceding analysis implies that such a solution does exist.

Putting everything together, we have shown:

Theorem 3.6 *The two-center problem with obstacles, for a set S of n points in the plane, and a collection of polygonal obstacles with a total of m edges, can be solved in randomized expected $O(m\log^2(mn) + mn\log^2 n \log(mn))$ time.*

4 An Efficient Approximation Algorithm

In this section we develop a near-linear algorithm that produces an approximation to the optimal 2-center problem with obstacles. That is, given $\varepsilon > 0$, the algorithm computes two congruent disks whose union covers S, whose centers lie in F and whose common radius is at most $(1+\varepsilon)r^*$, where r^* is the optimal radius. The algorithm runs in $O(1/\varepsilon \log(1/\varepsilon)(m \log^2 m + n \log^2 n))$ time.

We begin with the following easy observation: Let Δ denote the diameter of S, and let $p, q \in S$ be two points such that $|pq| = \Delta$. (Clearly, p, q and Δ can be computed in $O(n \log n)$ time.) Suppose first that $r^* < \Delta/5$. In this case, the distance $|c_1c_2|$ between the centers of the two optimal covering disks must be at least $|pq| - 2r^* > 3r^*$. This implies (cf. the analysis in the first case of the algorithm that solves the (exact) 2-center problem with obstacles) that the two optimal disks are disjoint and can be separated by a line whose orientation belongs to some fixed set of constant size.

The first stage of our algorithm finds the optimal solution in case $r^* < \Delta/5$, as follows. Clearly, in this case the two covering disks are well separated, so, as already noted in the previous section, we can construct a set of $O(1)$ lines so that at least one of them separates the two covering disks. Let ℓ be a line in this set, and let S_1, S_2 be the two subsets into which S is partitioned by ℓ. We find the smallest disk enclosing S_1 and centered at F in $O((m+n)\log(mn))$ time, using the algorithm of [9], and similarly for S_2. Asymptotically the overall running time of this stage is the same: $O((m+n)\log(mn))$.

The next stage deals with the case where r^* is much larger than Δ. Let p denote the centroid of S. Note that for any $R > \Delta$, if F and $B_R(p)$ are disjoint, then any disk centered at a point of F and containing any point of S must have radius at least $R - \Delta$. On the other hand, if $F \cap B_R(p) \neq \emptyset$ and c is any point in this intersection then $S \subset B_{R+\Delta}(c)$, implying that $r^* \leq R + \Delta$.

We apply the observations in the preceding paragraph with $R = (1+2/\varepsilon)\Delta$. Clearly, we can detect in $O(m+n)$ time whether $F \cap B_R(p) = \emptyset$. If so, we compute the point $c \in F$ nearest to p (in $O(m)$ time), and return the single disk $B_{|cp|+\Delta}(c)$ as an approximate solution. It is clear that this disk covers S. Moreover, as argued above, any disk centered at a point of F and containing a point of S must have radius at least $|cp| - \Delta$. Hence,

$$\frac{|cp| + \Delta}{r^*} \leq \frac{|cp| + \Delta}{|cp| - \Delta} \leq \frac{(1 + 2/\varepsilon)\Delta + \Delta}{(1 + 2/\varepsilon)\Delta - \Delta} = \frac{(1 + 2/\varepsilon) + 1}{(1 + 2/\varepsilon) - 1} = 1 + \varepsilon,$$

implying that our solution is a $(1+\varepsilon)$-approximation to the optimum.

Hence, in the remaining part of the algorithm (the third stage), we may assume that

$$\frac{\Delta}{5} \leq r^* \leq \left(2 + \frac{2}{\varepsilon}\right)\Delta.$$

Put $r_0 = \Delta/5$ and $\beta = 10 + 10/\varepsilon$. We thus assume that $r^* \in [r_0, \beta r_0]$. Define $r_j = r_0(1+\varepsilon)^{j/2}$, for $j = 0, \ldots, J$, where J is the smallest integer for which $(1+\varepsilon)^{J/2} \geq \beta$; that is,

$$J = 2\left\lceil \frac{\log \beta}{\log(1+\varepsilon)} \right\rceil = O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right).$$

The idea of the final stage of the algorithm is to run a binary search on the list of 'critical radii' $(r_0, r_1, \ldots, r_J)$. At each step of the search, we run an approximating procedure for the *two-disk-covering problem with obstacles*. The exact version of this procedure receives as input S, O and F, as above, and a radius r, and aims to determine whether S can be covered by two disks of radius r, both centered at F. The approximating solution of this problem will be presented in the following subsection. At each search step, we run this procedure with some r_j as the input radius. The approximating procedure can have one of the following two possible (not mutually exclusive) outputs: (a) $r^* > r_j$; (b) $r^* \leq r_{j+1}$; in the latter case, the procedure also outputs two congruent disks of radius at most $(1+\varepsilon)^{1/2} r_j = r_{j+1}$ that are centered at points of F and whose union covers S. Hence, after $O(\log(1/\varepsilon))$ calls to this procedure, we will have obtained a radius $r_j \leq r^*$ and two congruent covering disks of radius $\leq r_{j+2} = (1+\varepsilon) r_j$. In other words, we will have obtained an approximate solution with the desired properties.

4.1 An approximation algorithm for the two-disk-covering problem with obstacles

Let S, O, F and $\varepsilon > 0$ be as above, and let $r > 0$ be an input radius. Recall that the goal of the exact problem is to determine whether there exist two congruent disks of radius r which are centered at points of F and whose union covers S. The goal of the approximating version is to determine whether there exist two congruent disks of radius at most $(1+\varepsilon)^{1/2} r$ which are centered at points of F and whose union covers S. We first establish the following lemma:

Lemma 4.1 *If there exist two congruent disks D_1, D_2 of radius r that are centered at points of F and whose union contains S then there also exist two other disks D_1', D_2' of radius at most $(1+\delta/2)r$ such that*

(a) $D_1' \cup D_2'$ covers S;

(b) The centers of D_1' and D_2' both lie in the Minkowski sum (where $B_{\delta r/2}$ is the ball of radius $\delta r/2$ centered at the origin)

$$F_\delta = F \oplus B_{\delta r/2} = \{x + y \mid x \in F, |y| \leq \delta r/2\};$$

and

(c) Either D_1' and D_2' are disjoint and there exists a line that separates them and has orientation $j\delta/6$, for some integer $0 \leq j \leq 12\pi/\delta$, or D_1' and D_2' intersect and the line connecting the two crossing points of their boundaries has orientation $j\delta/6$ for some j as above.

Proof: Let c_1 and c_2 be the centers of D_1 and D_2, respectively. If $|c_1c_2| > 3r$ then it is clear that $D_1' = D_1$ and $D_2' = D_2$ satisfy (a)–(c) (in fact, they satisfy a stronger property than (c), already used in Subsection 3.3, that there is a line separating these disks whose orientation belongs to a canonical set of constant size, independent of δ). So assume that $|c_1c_2| \leq 3r$. Without loss of generality, assume that the orientation θ of c_1c_2 is between $\pi/2$ and $\pi/2 + \delta/6$. Rotate c_2 about c_1 by the angle $\theta - \pi/2$ in clockwise direction, and let c_2' be the resulting point, which

lies vertically above c_1. We claim that $D_1' = B_{(1+\delta/2)r}(c_1)$ and $D_2' = B_{(1+\delta/2)r}(c_2')$ are two disks with the desired properties. Indeed, we have $|c_2c_2'| < |c_1c_2|\delta/6 \leq 3r\delta/6 = \delta r/2$. This implies that $D_2 \subset D_2'$, from which (a) follows. Property (b) trivially holds for D_1' and is an immediate consequence of the preceding inequality for D_2'. Property (c) (with $j = 0$ in the specific case assumed above) is also immediate. $\Box$

The algorithm is now obvious. We put $\delta = (1+\varepsilon)^{1/2} - 1$. We first compute F_δ. As is well known, F_δ has $O(m)$ complexity and it can be computed in (deterministic) $O(m \log^2 m)$ time [11], or in randomized expected $O(m \log m)$ time [12]. We next iterate over the $O(1/\delta)$ canonical orientations in Lemma 4.1. Let θ be one of them, and assume for convenience that θ is vertical. Sort the points of S in their increasing x-order; suppose that this order is $(p_1, p_2, \ldots, p_n)$. Put $S_i = \{p_1, \ldots, p_i\}$ and $S_i' = \{p_{i+1}, \ldots, p_n\}$, for $i = 0, \ldots, n$, and test whether there exists an i such that each of S_i, S_i' can be covered by a disk of radius $(1+\delta/2)r$ that is centered at a point of F_δ. This can be efficiently carried out using the dynamic data structure of Section 2.

The fixed-radius circle shooting structures of [3] work with the same resource bounds as cited in Theorem 2.1 for the generalized polygons that constitute the complement of F_δ. We briefly justify this claim; for this we assume familiarity of the reader with the paper [3]. We note that the vertical decomposition used in [3] goes through for the generalized polygons after breaking circular arcs into subarcs at points of vertical tangency. Also, the data structure of [3] calls for the computation of the lower envelope of a collection of graphs, each being the boundary of the Minkowski sum of a disk of radius r with either a segment or a circular arc of radius $\delta r/2$. We compute the envelopes separately for each family of objects in $O(k \log k)$ time, where k is the number of objects in the family, and then merge the resulting envelopes in time linear in their complexity, which itself is linear in the number of objects defining the envelope.

If no solution was found for any of the canonical orientations, we conclude that $r^* > r$ and output this inequality. Otherwise, we take each of the resulting disks, call it D, and find the point $q \in F$ nearest to its center c. We replace D by $B_{|cq|+(1+\delta/2)r}(q)$, and note that its radius is at most $(1+\delta)r = (1+\varepsilon)^{1/2}r$. We output the two new disks. (Strictly speaking, if the disks have unequal radii, we replace the smaller one by a concentric disk that is congruent to the larger one.)

For each of the $O(1/\varepsilon)$ canonical orientations and for each of the $O(\log J) = O(\log(1/\varepsilon))$ binary search steps, the decision procedure of the third stage of the algorithm runs in $O((m \log^2 m + n \log n \log(mn)) = O(m \log^2 m + n \log^2 n)$ time, or in randomized expected time $O((m \log m + n \log n \log(mn))$. Putting everything together, we obtain:

Theorem 4.2 *Given S, O and F as above, and a parameter $\varepsilon > 0$, one can construct two congruent disks that are centered at points of F, whose union covers S and whose common radius is at most $1+\varepsilon$ times the optimal radius; the algorithm runs in time $O(1/\varepsilon \log(1/\varepsilon)(m \log^2 m + n \log^2 n))$ or in randomized expected time $O(1/\varepsilon \log(1/\varepsilon)((m + n \log n) \log(mn)))$.*

5 Conclusions

The introduction of obstacles in the p-center problem is natural in the context of facility-location theory as it expresses constraints on where facilities can be placed. We presented two algorithms for solving the 2-center problem with obstacles: an exact algorithm and a near-linear approximation algorithm. These seem to be the first published efficient solutions to this problem.

A major problem that remains open is to devise a near-linear *exact* algorithm for the problem. In fact, any solution with running time $o(mn)$ would be interesting. Notice that the only stage in our solution whose time requirements involve an mn factor is the stage where we look for disks, at least one of which has a center lying on an obstacle edge. The other stages take near-linear time.

As mentioned in the Introduction, our motivation to study this problem comes from robot workcell layout. The robot placement problem has several variants that also merit investigation:

- Industrial robots or other facilities may experience downtime. Place robots or facilities to insure redundancy (i.e., each workpoint covered by at least 2 disks).
- Consider non-circular covers. For example cover all workpoints with cones (e.g., as though positioning cameras or other sensors). Industrial robots have joints limits, so the effective cover may be a disk sector instead of a disk. This adds another dimension to the problem since non-circular covers require specifying orientation.
- Similarly, industrial robots and facilities may be mobile. Consider the placement problem when one or more robots are mounted on linear tracks that allow translation. Decide where to place the tracks avoiding collision with the obstacles. This is an extension of the *segment-center problem* [1] where obstacles are also considered.
- Often industrial robots are limited in how close the end-effector can get to the robot base. This means that instead of covering the workpoints by disks we actually need to cover them by annuli, where the inner radius is fixed.
- If the workspaces of the robots overlap, then robots may collide as they reach the workpoints. This could be avoided during run-time using motion planning, or by covering with *disjoint* disks, which gives rise to a new variant of the p-center problem.

Finally, it would be interesting to devise efficient solutions (exact or approximate) to the p-center problem with obstacles with $p > 2$.

References

[1] P. K. Agarwal, A. Efrat, M. Sharir, and S. Toledo. Computing a segment center for a planar point set. *J. Algorithms*, 15(2):314–323, 1993.

[2] T. Y. Chan. More planar two-center algorithms. *Comput. Geom. Theory Appl.*, 13:189–198, 1999.

[3] S.-W. Cheng, O. Cheong, H. Everett, and R. van Oostrum. Hierarchical vertical decomposition, ray shooting, and circular arc queries in simple polygons. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 227–236, 1999.

[4] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.

[5] R. Cole, J. Salowe, W. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM J. Comput.*, 18(4):792–810, 1989.

[6] J. Craig. Geometric algorithms in Adept RAPID. In P. K. Agarwal, L. Kavraki, and M. Mason, editors, *Third Workshop on Algorithmic Foundations of Robotics*, pages 133–139. A. K. Peters, Ltd, Wellesley, MA, 1998.

[7] D. Eppstein. Dynamic three-dimensional linear programming. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 488–494, 1991.

[8] D. Eppstein. Faster construction of planar two-centers. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 131–138, 1997.

[9] D. Halperin and C. Linhart. The minimum enclosing disk with obstacles. Manuscript, 1999. Java applet: http://www.math.tau.ac.il/CGAL/Projects.html.

[10] P. Hansen, B. Jaumard, and H. Tuy. Global optimization in location. In Z. Drezner, editor, *Facility Location*, pages 43–68. Springer-Verlag, New York, 1995.

[11] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translation al motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, 1986.

[12] J. Matoušek, N. Miller, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 49–58, 1991.

[13] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12:759–776, 1983.

[14] N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM J. Comput.*, 13(1):182–196, 1984.

[15] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[16] M. Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete Comput. Geom.*, 18:125–134, 1997.