Faster Construction of Planar Two-centers

David Eppstein*

Department of Information and Computer Science
University of California, Irvine, CA 92717
http://www.ics.uci.edu/∼eppstein/

Tech. Report 96-12

July 5, 1996

Abstract

Improving on a recent breakthrough of Sharir, we show how to find two circular disks of minimum radius covering a set of points in the Euclidean plane, in randomized expected time $O(n \log^2 n)$.

*Work supported in part by NSF grant CCR-9258355 and by matching funds from Xerox Corp.

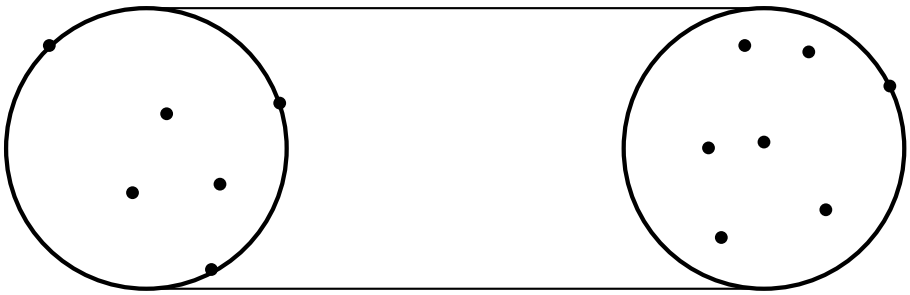

Figure 1. Set covered by two disks of same radius, with points on outer boundaries.

1 Introduction

The k-center problem in a metric space consists of, given a set S of some n points in the space, finding k points (called centers, usually not required to be a subset of S) such that the maximum distance from any point in S to the nearest center is minimized. A case of particular interest is the *planar two-center* problem [7], which can be viewed less abstractly as one of covering a set of points in the plane by two circular disks, in such a way as to minimize the radius r^* of the larger of the two disks.

By increasing the size of the smaller of the two disks, we can assume without loss of generality that the solution to the planar two-center problem consists of two disks D_1 and D_2 of equal radius r^*, situated in such a way that each disk has an input point on the boundary of the convex hull of the two disks, and such that D_1 is the circumcircle of some set of two or three points (such a configuration is shown in Figure 1).

For a long time the best algorithms for this problem have had time bounds of the form $O(n^2 \log^c n)$. The basic idea behind many of these algorithms was that one can find a line separating the points in D_1 from the points in D_2 (if the disks overlap, this line can be chosen to connect the points where the two disks' boundaries cross). Agarwal and Sharir [1] use a *parametric search* through the $O(n^2)$ combinatorially distinct partitions of the points by a line to solve the two-center problem. The general idea of this method is to use a data structure for a *decision problem* (does a given point set have circumradius less than some threshhold) to test whether r^* greater or less than some given value r. By simulating this algorithm generically, on the unknown value $r = r^*$, using the decision algorithm itself to determine the answer to any test involving r^*, one gets a solution to the two-center

problem in time $O(n^2 \log^3 n)$. Katz and Sharir [13] achieved a similar result by a method involving expander graphs.

In previous work [8], the author replaced this indirect parametric search with a more direct idea: instead of a data structure for the decision problem, use a data structure for the exact circumradius to determine the optimal two disks corresponding to any partition of the input. Because it is solving a harder problem, the exact circumradius data structure is somewhat slower than the corresponding decision problem data structure, but this slowdown is made up for by the fact that only one sweep through the possible partitions is required, instead of the multiple sweeps performed in a parametric search. Using this idea, the author achieved a time bound of $O(n^2 \log^2 n \log\log n)$.

Jaromczyk and Kowaluk [12] were the first to break away from the idea of searching all linear partitions of the input points. Their method breaks the problem into several cases, and performs a different type of search for each case. Their result is an $O(n^2 \log n)$ bound.

In a recent breakthrough, Sharir [15] used a similar sort of case analysis to greatly improve all of these algorithms, giving a two-center algorithm with running time $O(n \log^c n)$. The basic idea is to search for different partitions depending on the relative positions of the disks D_1 and D_2 defined above. If the two are far apart, one can easily separate them by a line. If they are nearly tangent to each other, Sharir separates the points into two sets, those contained in D_1 and all remaining points. And finally, if D_1 and D_2 are close to concentric, so that they have a large area of overlap, Sharir finds some point x in that overlap and partitions the points by two rays from x through the two points where the circles cross. In each case Sharir finds the relevant partition by a parametric search, like that of Agarwal and Sharir [1], based on a data structure for the circumradius decision problem. The actual time bound given in Sharir's paper is $O(n \log^9 n)$, but Sharir writes "We have not made a serious attempt to improve the performance" and some improvement is clearly possible. However it seems that, by following this approach, one will still end up with an algorithm with several logarithmic factors in it, that is quite complicated and impractical for all but the largest values of n.

In this paper we apply the same idea to Sharir's method that our earlier paper did to Agarwal and Sharir's: replace the decision problem data structure and parametric search by a direct search using an exact circumradius data structure. With this idea, we show that the nearly concentric case can be solved in randomized expected time $O(n \log n \log\log n)$, and the other cases can be solved in time $O(n \log^2 n)$. With this improvement, the total time for our version of Sharir's algorithm is $O(n \log^2 n)$.

2 Overview

Our algorithm closely follows that of Sharir. We outline the method here, and explain each of the steps in detail later. The overall structure is a three-part case analysis: either the disks are far apart (the ratio of the distance between centers to r^* is significantly greater than 2), the disks are close to tangent (the ratio is near 2), or the disks are close to concentric (the ratio is significantly less than 2). Each of these cases is handled differently; the basic ideas in each case are very similar to those of Sharir, but we handle them more carefully and reduce as much as possible the portion of the algorithm that must be handled with parametric search, to reduce the total time. As we do not know a priori which case the input falls into, we perform the algorithms for each of the following cases and return the best two-disk cover returned by any of them.

If D_1 and D_2 are far apart, we find a set of $O(1)$ lines such that one of them separates D_1 from D_2. By computing the circumradii on each side of each such partition, we find r^* for this case in time $O(n)$.

If D_1 and D_2 are nearly tangent, we can still find a set of $O(1)$ lines such that one of them nearly separates D_1 from D_2. We use these lines to find a set S that is entirely contained in D_1 and includes a point on the boundary of D_1. To test a given radius r, we then find the circular hull of S, and swing a circle of radius r around this hull to find a sequence of partitions of the point set, and apply an offline dynamic decision problem data structure. Combining this decision algorithm with parametric search yields an $O(n \log^2 n)$ time bound.

If D_1 and D_2 are close to overlapping, we can find a set of $O(1)$ points such that one of them (call it x) is contained in both D_1 and D_2. We then sort the input points radially around x, and consider partitions determined by two rays through x. The angles of the rays can be considered as indices into a two-dimensional array, and by using sorted array selection methods of Frederickson and Johnson we can find the optimum pair of angles after a sequence of $O(n)$ circumradius evaluations. Each evaluation can be answered by a modified version of our previous paper's exact circumradius data structure, or one can perform many evaluations at once using an offline decision problem data structure. By combining both methods of evaluation, we get an algorithm for this case with running time $O(n \log n \log \log n)$.

3 Circumradius data structures

We now review what is known about computation of the circumradius of a point set (i.e., the 1-center problem). This is well known to be solvable using linear-programming type methods, in linear time [14]. However we are interested in *dynamic* circumradius problems, in which as points are inserted to our set or deleted from it, we must maintain the circumradius of the changing set. In all our applications, the set we will be maintaining is a subset of our original input, and in general the sequences of insertions and deletions of the dynamic algorithm will be *offline* (that is, the entire sequence is known in advance before any circumradius computation need be performed). All the time bounds we cite will be measured in terms of the total number of updates (insertions and deletion), since rather than starting with some particular set of points one can imagine starting from the empty set and inserting points one by one.

Hershberger and Suri [11] provide an offline dynamic algorithm (in the sense described above) for a closely related problem: given a value r, maintain a dynamic point set and determine after each change whether its circumradius is less than or greater than r. We call this the *circumradius decision problem*.

Lemma 1 (Hershberger and Suri). *An offline sequence of n updates in the circumradius decision problem can be solved in time $O(n \log n)$.*

Sharir [15] gives a different algorithm for this same offline decision problem, with the somewhat worse running time $O(n \log^2 n)$; however his method is easier to parallelize, which makes it easier to use in parametric search methods. We will not need a parallel version of the offline decision problem, so we use the faster method of Hershberger and Suri.

We next consider maintaining not just the answer to a decision problem about the circumradius, but the circumradius itself. In a previous paper, the author [8] solved the offline version of the exact circumradius problem. We will modify this algorithm later, so we describe it in a little more detail here.

The basic idea (common to many offline algorithms) is to use a *segment tree* on a set of intervals corresponding to the insertion and deletion times of each point. This results in a recursive partition of the points into $O(n)$ subsets, each point belonging to $O(\log n)$ subsets, such that The set at each time during the offline sequence is representable as the disjoint union of a collection of $O(\log n)$ subsets, found by following a path to the root in the segment tree.

Our data structure then lifts each subset to a three-dimensional set using the lifting transformation $(x, y) \mapsto (x, y, x^2 + y^2)$, finds the convex hull of each lifted subset, and constructs a Dobkin-Kirkpatrick recursive decomposition of each hull. Once we have constructed this data structure, it remains to determine the circumradius of the points in the set after each update. We do this by combining the appropriate subsets of points:

Lemma 2 (Eppstein). *Given k sets of $O(n)$ points, represented by the Dobkin-Kirkpatrick hierarchies of their liftings, we can find the circumradius of their union in time $O(k^3 \log^3 n)$.*

This gives an algorithm with polylogarithmic time per update, but the exponent in the polylog is large. By applying a randomized recursion of Clarkson [5] this can be reduced to $O(k \log n + \log k \log^3 n)$. In the application of this to offline dynamic circumradius, $k = O(\log n)$ and the overall time is polylogarithmic (with or without the randomized recursion). Our previous paper further reduces the time per update to $O(\log^2 n \log\log n)$ by using the structure of the segment tree: Each collection of k sets of points is formed by adding one subset to a collection of $k - 1$ sets of points. If the minimal enclosing disk of those $k - 1$ sets also encloses the kth set (as can be tested quickly using the Dobkin-Kirkpatrick hierarchy) there is nothing more to optimize. Otherwise, one of the three points determining the minimal enclosing disks must come from the kth set itself and this can be used to improve the running time.

For our application of these ideas to the 2-center problem, we will not have an offline sequence of updates. However we will again be able to form a recursive partition of the points into subsets, such that any circumradius query we are interested in can be answered by combining $O(\log n)$ subsets. Similar speedups to the ones described above can be used to make these queries fast; however for our purposes it will suffice that they are polylogarithmic.

The same method applies in a more general *semi-online* setting, in which only partial information about the future sequence of updates is known. For completeness, we mention that the fully dynamic circumradius problem can be solved in worst case time $O(n^\epsilon)$ per update [2], and in expected time $O(1)$ per update for a certain class of input distributions [9]; however we will not use these results here.

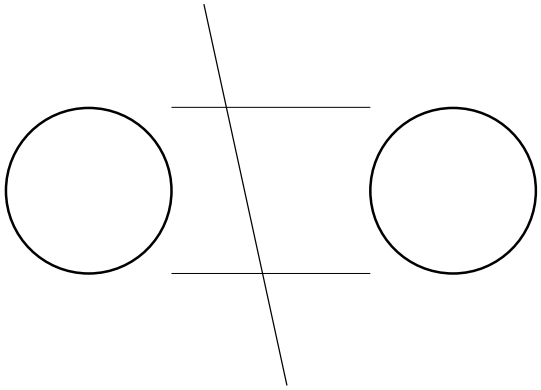

Figure 2. Far apart disks can be separated by a line.

4 Well separated disks

We now start our discussion of our two-center algorithm with the case in which the disks D_1 and D_2 are far apart. Strictly speaking this case is unnecessary as it is subsumed in the next one, however it provides a simple warm-up to the problem.

Lemma 3. *Suppose that the distance between the centers of D_1 and D_2 is at least $(2+\epsilon)r^*$, for some constant $\epsilon > 0$. Then the set of lines separating the circles forms a set of measure a constant fraction of the measure of the set of all lines crossing the minimal enclosing disk of the points.*

We briefly sketch the proof here: if the disk centers are at distance $d \geq (2+\epsilon)r^*$, then one can draw two line segments on parallel lines tangent to the disks, of length $d - 2r^* = \Omega(d)$ each, such that any line crossing the two segments separates the disks (Figure 2). The measure of the set of lines crossing the two segments is $\Omega(d)$ (since they have length $\Omega(d)$ and are spaced at most $O(d)$ units apart). But the points have circumradius at most $d + 2r^* = O(d)$ so the measure of the lines crossing their minimal enclosing disk is $O(1)$.

As a consequence of this result, one can find a set of $O(1)$ lines (where the constant here depends on ϵ) such that, if the disks D_1 and D_2 fall into this case, one of the lines separates them. We can then simply compute the circumradii of each pair of sets partitioned by one of these lines, in $O(n)$ time each. If the disks are well separated, one of these pairs of circumradii gives the optimal value r^*.

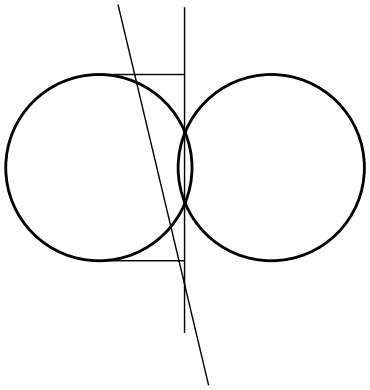

Figure 3. If disks are nearly tangent, a line separates most of D_1 from D_2.

Lemma 4. *Suppose that the distance between the centers of D_1 and D_2 is at least $(2+\epsilon)r^*$, for some constant $\epsilon > 0$. Then we can find r^* (and a two-disk cover with radius r^*) in time $O(n)$.*

5 Nearly tangent disks

We next consider the case in which the centers of D_1 and D_2 are separated by a distance close to $2r^*$, so the disks are close to tangent. This case turns out to be the bottleneck of the overall algorithm.

Lemma 5. *Suppose that the distance d between the centers of D_1 and D_2 is at least ϵr^*, for some constant $\epsilon > 0$. Then the set of lines containing the portion of the convex hull boundary of D_1 and D_2 belonging to D_1, and not containing any point of $D_2 - D_1$, forms a set of measure a constant fraction of the measure of the set of all lines crossing the minimal enclosing disk of the points.*

The proof is similar to that for Lemma 3. We draw two line segments, on the parallel lines tangent to D_1 and D_2, passing from the points of tangency on D_1 to the perpendicular bisector of the two disk centers (Figure 3); all lines crossing both segments have the properties described in the lemma, and (since the line segments have length $\Omega(d)$ and are within distance $O(d)$ of each other, and the point set as a whole has circumradius $O(d)$) this set of lines has a constant fraction of the total measure of the lines passing through the minimal enclosing disk of the points.

Corollary 1. *Suppose that the distance d between the centers of D_1 and D_2 is at least ϵr^*, for some constant $\epsilon > 0$. We can find a family of $O(1)$ subsets of the points, such that for some member S of the family, S is entirely contained in D_1 and has at least one point on the boundary of D_1.*

As before, we do not know which of the sets in the family is the one S described above, but we can perform the following algorithm for all the members of this family, and return the best two-disk cover found by these separate runs.

Suppose we know the set S, and we want to test whether $r < r^*$ for some given radius r. We can test this as follows: Compute the circular hull of S with radius r (that is, the intersection of all radius-r disks containing S); this is a convex figure with radius-r circular-arc sides. We consider a process in which a circle tangent to this hull and containing it swings around with different angles of tangency, in the process passing through all points of the hull. As the circle passes around the hull, its boundary may cross over some of the other points of our input (but not of course over any points of S); we form an offline sequence of insertions and deletions corresponding to these crossings, to keep track of the set inside the circle. Then $r \geq r^*$ if and only if one can expand D_1 and D_2 to form disks D_1' and D_2' of radius r, with D_1' containing all of S and having a point of S on its boundary, and D_2' containing all those points not in D_1'; but then our moving circle must coincide at some point with D_1', and we can test for the existence of an appropriate D_2' using Hershberger and Suri's offline decision problem data structure.

We summarize the results on this case so far:

Lemma 6. *Suppose that the distance d between the centers of D_1 and D_2 is at least ϵr^*, for some constant $\epsilon > 0$. Then in $O(n \log n)$ time we can test whether $r \geq r^*$ for any given value r.*

Sharir [15] describes a decision algorithm essentially identical to the one above, but gives a slower $O(n \log^2 n)$ bound for the same problem, only because he uses his slower offline decision problem data structure in place of that of Hershberger and Suri.

To use this result to actually find r^*, we apply *parametric search*. This method has been applied described many times in computational geometry, so we only briefly summarize it here. It involves simulating the decision algorithm, performing each step the way it would be performed if $r = r^*$. At each conditional branch of the decision algorithm, the step to be performed

next can typically be determined not from r itself but from the sign of some low-degree polynomial in r. To compute this sign for the unknown value $r = r^*$ we find the roots of the polynomial and use the decision algorithm itself to determine which of them are above and below r^*. Since the decision algorithm's behavior changes for r above and below r^*, the simulation of it must end up testing $r = r^*$ to determine the correct behavior. Therefore we can determine r^* by looking at all values tested in this simulation, and choosing the smallest of those for which the result of the test was that it was at least r^*.

This method as described above would roughly square the running time of the decision algorithm, leaving us no better than known results on the two-center problem. However the algorithm being simulated can be sped up in several ways. First, it may be a parallel algorithm rather than a sequential one; then at each time step many roots of polynomials can be simultaneously compared to r^* via binary search, giving an overall time equal to the work of the parallel algorithm plus $O(T \log n)$ calls to the decision algorithm, where T is the parallel time. Second, only the parts of the algorithm that depend on r^* need to be parallelized or simulated; the rest can be done once and for all outside the parametric search. Third, a technique of Cole [6] allows us in many cases (particularly those involving sorting) to call the decision algorithm only once per parallel time step of the simulated algorithm, rather than logarithmically many times. And fourth, the simulated algorithm need not actually solve the decision problem; it need merely be any algorithm the behavior of which undergoes a discrete change at $r = r^*$. We apply all of these speedup techniques to this problem.

Lemma 7. *The sequence of offline updates produced by the decision algorithm outlined above undergoes a discrete change at $r = r^*$.*

Proof: By assumption, D_1 is the circumcenter of some two or three points, one of which is on the circular hull of S. If it is the diameter circle of two points, or if it passes through two points of S, then at $r = r^*$ the sequence changes from including the remaining point to not including it. If D_1 is the circumcenter of a triangle with two vertices outside S, the order in which those two vertices are crossed by the moving circle changes at $r = r^*$. □

Lemma 8. *The sequence of vertices on the circular hull of S with respect to radius r can be found by performing $O(n \log n)$ steps not depending on r together with one parallel step involving $O(n)$ independent computations.*

Proof: Each boundary segment of the circular hull corresponds to a circle of radius r tangent to two vertices and entirely containing the rest of S. Such a circle has its center on an edge of the farthest point Voronoi diagram of S. We compute this diagram in time $O(n \log n)$, and compute for each edge of the diagram the interval of radii for which some circle with its center on that edge contains S and is tangent to two points. The parallel step then involves simply comparing the $O(n)$ interval endpoints with r itself. This gives us the identities of the endpoints of boundary segments of the circular hull, from which the hull itself is easily constructed. □

Lemma 9. *The offline sequence of insertions and deletions described above can be found by performing $O(n)$ parallel binary search operations in the circular hull, together with one sorting algorithm.*

Proof: The binary searches determine which vertex of S the moving circle is pivoting around when it crosses the given point. From this information, the time the crossing occurs can be computed in constant time by a low-degree polynomial in r, and we need merely sort these times. □

The steps described above constitute a parallel algorithm for computing the offline sequence of updates of the decision problem; this sequence changes discretely at $r = r^*$, and the parallel algorithm (consisting of one completely parallel step, a collection of binary searches, and a sorting algorithm) is suitable for Cole's parametric search speedup. Thus we have the following result:

Lemma 10. *Suppose that the distance d between the centers of D_1 and D_2 is at least ϵr^*, for some constant $\epsilon > 0$. Then we can find r^* (and a two-disk cover with radius r^*) in time $O(n \log^2 n)$.*

6 Nearly concentric disks

We now describe a solution in the case when the disks are nearly concentric. Like Sharir [15], we reduce the problem to one of searching a certain $n \times n$ matrix. However at this point we differ: Sharir applies a search technique similar to methods of Aggarwal et al. [3] for finding row minima in matrices satisfying a certain monotonicity condition, where the method we use is more closely related to methods of Frederickson and Johnson [10] for selection in sorted matrices.

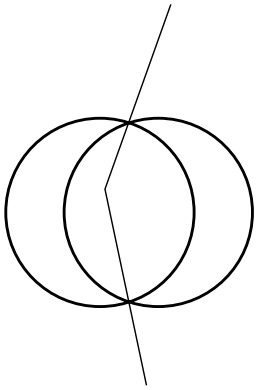

Figure 4. If disks are nearly concentric, input can be partitioned by rays from a point in their intersection.

Lemma 11. *Suppose that the distance d between the centers of D_1 and D_2 is at least $(2-\epsilon)r^*$, for some constant $\epsilon > 0$. Then the set of points contained in $D_1 \cap D_2$, and forming an angle bounded away from zero with the two crossing points of D_1 and D_2, forms a set of measure a constant fraction of the measure of the set of points in the minimal enclosing disk of the points.*

Corollary 2. *Suppose that the distance d between the centers of D_1 and D_2 is at least $(2-\epsilon)r^*$, for some constant $\epsilon > 0$. Then we can find a set of $O(1)$ points such that at least one of them is contained in $D_1 \cap D_2$ and forms an angle bounded away from zero with the two crossing points of D_1 and D_2.*

Obviously we don't know which of these points is the one in $D_1 \cap D_2$, so we try them all as before. From now on let x be a known point assumed to be in $D_1 \cap D_2$. (It is not necessary for x to be one of the input points). We consider partitions of the input points into two sets, bounded by a pair of rays from x. Clearly, if those rays pass through the crossings of D_1 with D_2, they bound regions in which all points contained in either circle are entirely contained in one of the two circles (Figure 4). Because the angle from x to the two crossing points is by assumption bounded from zero, we can find $O(1)$ lines through x, one of which separates the two rays from each other; again we do not know which so we try all possibilities. From now on we assume one of these lines is fixed, and without loss of generality assume that it is horizontal.

Thus to find r^* we need merely find the partition of the points by two rays, one going upwards from x and one downwards from x, that minimizes the circumradii of the two sets.

We sort the input points above and below x radially around x, and number the rays upwards of x and downwards of x by the position at which they partition this sorted sequence. Thus the two-ray partitions we are interested in are made to correspond to pairs of integers, which can be interpreted as positions in a matrix M with the numbers of rows and columns totaling n. Each entry of the matrix can be thought of as giving a pair of numbers, the circumradii of the point sets on either side of the corresponding partition. We are interested in finding the entry minimizing the larger of the two numbers. For simplicity, we pad the matrix so that it is square with side length equal to $2^k + 1$ for some $k = \log_2 n + O(1)$.

6.1 Search with offline exact circumradius

We first describe a method for evaluating any specific entry in M.

Lemma 12. *We can evaluate any entry in M in time $O(\log^c n)$ for some constant c.*

Proof: Form a recursive partition of the points above and below x, in which each subset is partitioned in two equal subsets according to the radial sorted order around x. Each point is in $O(\log n)$ sets, and any contiguous subsequence of the sorted order can then be formed as the disjoint union of $O(\log n)$ sets. We can then lift the points in each set, form the convex hull and recursive decomposition, and combine sets exactly as in our offline exact circumradius algorithm, to achieve the claimed result. □

The exact value of c will depend on the details of the implementation (whether we allow randomization, and whether we take advantage of the order in which we evaluate entries of M); with some care it is possible in this case to evaluate matrix entries in randomized expected time $O(\log^2 n \log\log n)$ each. However for our results it only matters that $c = O(1)$.

We now describe how to use matrix evaluations to find the optimal entry in M.

Lemma 13. *After we evaluate a single entry of M, we can determine either that the quadrant of M above and right of the entry does not contain the optimal entry, or that the quadrant below and left does not contain the optimal entry.*

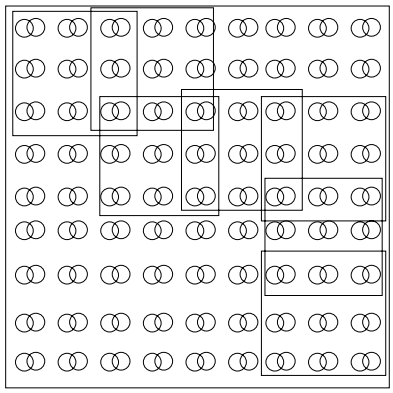

Figure 5. 9×9 matrix with a path of 3×3 squares.

Proof: One of the two circumradii at the evaluated entry is at least as large as the other, and the points in one of those two quadrants correspond to adding more points to that large circle, which can never make it smaller. □

Lemma 14. *Once we have found x and computed the radial sorted order around x, we can find the optimum circumradius pair in $O(n)$ matrix entry evaluations and $O(n)$ additional time.*

Proof: Recall that the side length of M is assumed to be $2^k + 1$ where $k = \log n + O(1)$. We perform a sequence of $k = O(\log n)$ stages. In each stage the remaining parts of the matrix in which the optimal entry may lie form a path of $O(2^i)$ squares, each of size $O(2^{k-i}+1)$, running from the upper left corner of the matrix to the lower right corner. Two successive squares in the path overlap either by one row or one column, and each successive square is either right of or below its predecessor. Figure 5 depicts such a path schematically, with $k = 3$ and $i = 2$; the matrix entries are represented symbolically by pairs of circles. (Initially, we have a path consisting of one single square, namely the whole matrix itself).

In each stage, we then subdivide these squares into four times as many squares, of half the size each. (More precisely the smaller squares have side length $2^{k-1-1} + 1$, and overlap in the centers of the larger squares.) This collection of smaller squares no longer forms a path, as some square corners are interior to the collection (all four small squares around them are present in the collection). We then simply evaluate each of these interior points and

eliminate one of the four neighboring squares. The remaining squares again form a path, and we are ready to repeat the process.

After $O(\log n)$ iterations, involving probes at $O(n)$ matrix entries, our path consists of $O(n)$ squares of size $O(1)$ and we can simply evaluate all remaining entries. □

As a consequence we can solve this case in time $O(n \log^c n)$. We next see how to use the offline decision problem algorithm (in place of the exact circumradius data structure used here) to speed this up.

6.2 Search with offline decision problem

We can use the same basic matrix search technique, combined with an offline decision problem, to achieve a somewhat better result. This portion of our algorithm is quite similar to the method used by Sharir [15].

The first observation is that any path through matrix M corresponds to an offline sequence of insertions and deletions in a dynamic circumradius problem (this is obvious, since each horizontal or vertical step corresponds to moving a point from one partition to the other either above or below x). In particular, the sequence of entries evaluated at each stage of the algorithm above can be connected to form a path of length $O(n)$, so we can use evaluate the same entries using Hershberger and Suri's offline decision algorithm and achieve a total time of $O(n \log^2 n)$ to determine whether $r \geq r^*$. (This differs from Sharir's $O(n \log^3 n)$ for this subproblem only because Sharir uses a slower offline decision algorithm.)

How do we make this into a search algorithm? The key idea is to relax the requirement that the collection of squares we have at any stage forms a path. Instead, we assume that at stage i we have $c2^i$ squares for some constant $c > 1$. As before, we subdivide each square into four, so now we have $2c(2^{i+1})$ squares; to continue the process we wish to reduce this number of squares back to c. If we could evaluate all interior corners of the collection of squares, this would be easy, but we can only use a decision algorithm.

Lemma 15. *Suppose we have a collection of $(1+b)2^i$ squares of size $2^{k-i}+1$ remaining in M, where b is some positive constant. Then in expected time $O((n + b2^i) \log n)$ we can eliminate $\Omega(b2^i)$ squares from the collection.*

Proof: In this situation, there most be $\Omega(b2^i)$ interior corners of squares. We choose a random entry among these interior corners, and compute its value (either using the data structure described earlier, or using a linear time circumradius algorithm). Call this value r We then use the offline decision

algorithm of Hershberger and Suri to test, for each interior square corner entry, whether one or both of the corresponding circles has radius at least r. If so, we can eliminate one or both of the neighboring squares. The expected number of eliminated squares is a constant fraction of the number of interior corners, so after a constant expected number of rounds of this procedure we will have eliminated $\Omega(b2^i)$ squares. □

Lemma 16. *Suppose that the distance d between the centers of D_1 and D_2 is at most $(2-\epsilon)r^*$, for some constant $\epsilon > 0$. Then we can find r^* (and a two-disk cover with radius r^*) in randomized expected time $O(n \log^2 n)$.*

Proof: We go through a similar sequence of stages to those in Lemma 14, maintaining the invariant that at stage i there are $O(2^i)$ squares of size $2^{k-i}+1$. After $O(\log n)$ stages, each taking time $O(n \log n)$, we will have reduced the problem to one of evaluating $O(n)$ remaining entries of M. A similar strategy of selecting one entry, evaluating it, and performing the decision algorithm to test its value against the remaining entries, leads from this point to the optimal value in an expected $O(\log n)$ stages. □

6.3 Hybrid search

Although the $O(n \log^2 n)$ time above matches that for the previous case of the two-center problem, we now sketch an improvement. This shows that this nearly concentric case is not the bottleneck of the overall algorithm, which may be important if the other slower parts of the algorithm can be improved. Further, if one is interested in practical implementation, it pays to speed up every part of a program.

The idea is simple: the method using the exact circumradius data structure is slow because it evaluates many matrix entries, but the bound on its time does not really involve the number of stages in the matrix searching procedure. The method using the decision algorithm is slow because it performs many stages, but the bound on its time does not really involve the number of entries evaluated. By switching between the two algorithms we can combine the advantages of both.

Specifically, suppose we have a data structure as described in Lemma 12 for evaluating any matrix entry in time $O(\log^c)$. We then use the first method, involving this matrix evaluation data structure, only for the first $\log n - O(\log \log n)$ stages, for which the total number of entries evaluated is $O(n \log^{-c} n)$. The time for this portion of the algorithm is $O(n)$. We next switch to the other method, using an offline decision algorithm,

for the remaining $O(\log\log n)$ stages of the matrix search procedure. This takes expected time $O(n \log n \log\log n)$, after which there remain $O(n)$ matrix entries to evaluate. We continue with the offline decision algorithm procedure for $O(\log\log n)$ more stages, in each of which we test a randomly chosen entry from the remaining set; again this takes expected time $O(n \log n \log\log n)$, and results in a set of only $O(n \log^{-c} n)$ matrix entries to evaluate. Finally, we evaluate these with our data structure, in time $O(n)$.

Lemma 17. *Suppose that the distance d between the centers of D_1 and D_2 is at most $(2-\epsilon)r^*$, for some constant $\epsilon > 0$. Then we can find r^* (and a two-disk cover with radius r^*) in randomized expected time $O(n \log n \log\log n)$.*

7 Conclusions

Combining Lemmas 4, 10, and 17 gives the following result:

Theorem 1. *We can find r^* (and a two-disk cover with radius r^*) in randomized expected time $O(n \log^2 n)$.*

It may be interesting to consider how this can be improved. Bounds for the two-center problem have come a long way but it is not clear that we have reached the end of possible improvement. We are not aware of any nontrivial lower bounds for the problem, so conceivably $O(n)$ may be possible. There is only one bottleneck in our current time bound: the parametric search technique used in the case of nearly tangent disks. We achieved faster time bounds in other parts of the algorithm by avoiding parametric search; perhaps it can be avoided here as well. Alternately, one may wish to find a deterministic two-center algorithm with performance matching our randomized algorithm. Again, there is only one bottleneck: the randomized selection used in the offline decision procedure version of our algorithm for the case of nearly concentric disks. If we could improve our deterministic matrix entry evaluation procedure to $O(\log^2 n)$, we could avoid this portion of the algorithm; alternately perhaps the selection procedure could be derandomized. It remains interesting as well to consider improvements for k-center problems with $k > 2$.

References

[1] P. K. Agarwal and M. Sharir. Planar geometric location problems and maintaining the width of a planar set. *2nd ACM-SIAM Symp. Discrete Algorithms*, 1991, pp. 449–458.

[2] P. K. Agarwal, D. Eppstein, and J. Matoušek. Dynamic algorithms for half-space reporting, proximity problems, and geometric minimum spanning trees. *33rd IEEE Symp. Foundations of Comp. Sci.*, 1992, pp. 80–89.

[3] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix searching algorithm. *Algorithmica*, vol. 2, 1987, pp. 209–233.

[4] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *30th IEEE Symp. Found. Comput. Sci.*, 1989, pp. 586–591; *SIAM J. Comput.*, to appear.

[5] K. L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. *29th IEEE Symp. Found. Comput. Sci.*, 1988, 452–456.

[6] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. Assoc. Comput. Mach.*, vol. 34, 1987, pp. 200–208.

[7] Z. Drezner. The planar two-center and two-median problems. *Transportation Science* 18 (1984) 351–361.

[8] D. Eppstein. Dynamic three-dimensional linear programming. *32nd IEEE Symp. Foundations of Comp. Sci.*, 1991, pp. 488–494; *ORSA J. Computing*, vol. 4, 1992, pp. 360–368 (special issue on computational geometry).

[9] D. Eppstein. Average case analysis of dynamic geometric optimization. *5th ACM-SIAM Symp. Discrete Algorithms*, 1994, 77–86. *Comp. Geom. Theory & Applications*, to appear.

[10] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in $X + Y$ and matrices with sorted columns. *J. Comput. Sys. Sci.*, vol. 24, 1982, pp. 197–208.

[11] J. Hershberger and S. Suri. Offline maintenance of planar configurations. *2nd ACM-SIAM Symp. Discrete Algorithms*, 1991, 32–41.

[12] J. Jaromczyk and M. Kowaluk. An efficient algorithm for the Euclidean two-center problem. *10th ACM Symp. Computational Geometry*, 1994, pp. 303–311.

[13] M. Katz and M. Sharir. An expander-based approach to geometric optimization. *9th ACM Symp. Computational Geometry*, 1993, pp. 198–207.

[14] N. Megiddo. Linear-time algorithms for linear programming in $\mathbf{R}^3$ and related problems. *SIAM J. Comput.*, vol. 12, 1983, pp. 759–776.

[15] M. Sharir. A near-linear algorithm for the planar 2-center problem. *12th ACM Symp. Computational Geometry*, 1996, to appear.