

Faster Construction of Planar Two-centers

David Eppstein*

Abstract

Improving on a recent breakthrough of Sharir, we find two minimum-radius circular disks covering a planar point set, in randomized expected time $O(n \log^2 n)$.

1 Introduction

The k -center problem for a point set S is to find 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 [4], which can be viewed less abstractly as one of covering a set of points in the plane by two congruent circular disks, in such a way as to minimize the radius r^* of the disks.

For a long time the best algorithms for this problem had time bounds of the form $O(n^2 \log^c n)$ [1, 5, 12, 11]. In a recent breakthrough, Sharir [16] greatly improved 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 types of partition depending on the relative positions of the two disks D_1 and D_2 . 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. Sharir uses various search techniques to find the appropriate partition efficiently in each of these cases.

Although a very nice theoretical result, Sharir's method needs further refinement to be suitable for prac-

tical implementation. The method is complicated, and Sharir's time bound, while asymptotically close to linear, is high: $O(n \log^9 n)$. (To be fair, as Sharir himself writes, he did not make a serious attempt to improve the performance.) The main result of this paper is to reduce the running time of Sharir's algorithm to (randomized) $O(n \log^2 n)$. We also take some steps towards simplification of the algorithm, by reducing the number of cases and limiting our dependence on *parametric search*. Finally we describe some simple lower bounds on the problem.

2 Parametric Search Considered Harmful

The overall structure of Sharir's algorithm, like that of Agarwal and Sharir [1] and many other recent geometric optimization algorithms, is a *parametric search* [14]. The basic idea of this technique is to automatically translate a decision algorithm (in this case, for determining whether a set of points can be covered by a pair of disks with radius r) into an optimization problem (for finding the optimal radius r^*). The technique simulates the behavior of the decision algorithm as if it were given the optimal radius r^* as input. Each branching point of the decision algorithm is assumed to involve only tests of the sign of low-degree polynomials in the input; each such test can be simulated by computing the roots of these polynomials, and using the original decision algorithm to compare r^* against each of these roots. In this way, each step of the simulation takes time proportional to the decision algorithm itself. To avoid squaring the overall runtime, one often simulates a *parallel* version of the decision algorithm, so that many tests can be simulated at once by binary search in a large set of polynomial roots. The overall result is an algorithm which takes time $O(DT \log P + PT)$, where D is the time for the original decision algorithm, P is the number of processors used in its parallelization, and T is the number of parallel steps. In some circumstances one

*Department of Information and Computer Science, University of California, Irvine, CA 92697-3425, USA; <http://www.ics.uci.edu/~eppstein/>; eppstein@ics.uci.edu. Work supported in part by NSF grant CCR-9258355 and by matching funds from Xerox Corp.

can pipeline the binary searches and avoid the $O(\log P)$ term in this bound [3].

Although parametric search has been very useful in finding improved asymptotics for many problems, particularly in computational geometry, it has not tended to lead to advances in practical algorithms, and has been a contributor towards the gap between theory and practice. There are several reasons for this: the parallel algorithms used in parametric search are usually more complicated and less efficient than their sequential counterparts, the need to simulate a parallel algorithm on a sequential machine leads to added data structures to store the states of many “virtual processors”, the sequence of queries to the decision algorithm made by the simulation is difficult to understand, and finally the $O(DT \log P)$ term in the time bound tends to result in time bounds which, while having a small overall exponent, include many logarithmic factors which dominate the algorithm’s performance for all but the largest values of n . For these reasons researchers have even tried algorithms with more logarithmic factors, using such “theoretical” tools as expander graphs, in an attempt to get away from the parametric search paradigm [12].

In this paper we attack parametric search using the following ideas, which not only improve the asymptotics but also hopefully lead to more practical algorithms. None of these ideas is particularly new, but in combination they improve Sharir’s 2-center algorithm from $O(n \log^9 n)$ to $O(n \log^2 n)$ and greatly reduce the amount of parametric searching present in the algorithm.

Separate cases. Sharir’s algorithm consists of a single large parametric search, simulating a decision algorithm that branches into several cases. Instead, we branch into cases first, performing separate searches for each case. This allows us to tackle each case separately rather than having to deal with the whole algorithm at once. The same idea applies within each case, to algorithms consisting of a sequence of several steps: it is easiest to simulate each of these steps separately rather than as one compound algorithm.

Fewer cases. One of the three cases in Sharir’s algorithm turns out to be unnecessary; it is subsumed in a second case. This does not affect the runtime

since this case can be tested quickly, but eliminating this case simplifies the overall algorithm and reduces the number of places in which parametric search appears.

Simulate a simpler algorithm. The algorithm simulated by the parametric search need not actually solve the decision problem; it needs only to be discontinuous at the optimal value. In one case of the 2-center algorithm we show that an intermediate sequence of events produced as part of the decision algorithm is already discontinuous, so the rest of the algorithm need not be simulated.

Pull preprocessing and postprocessing out of the simulated algorithm. Not all of the algorithm being simulated must actually be simulated; parts of the algorithm that do not depend on the parameter may be run directly instead. It may pay to rearrange the algorithm so that fewer parts require simulation. In one case of the 2-center algorithm, we replace the simulation of a circular hull algorithm by a computation of a Voronoi diagram (preprocessing), a single binary search, and a “gluing step” forming the hull from the search results. Only the binary search needs to be simulated.

Use more powerful data structures. Much of the reason Sharir starts with a decision algorithm comes from his use of dynamic “circumradius decision” data structures, that determine whether the given radius bound is met for a dynamically changing subset of points [10]. By replacing these data structures with a data structure that can keep track of the actual circumradius of a subset of points, we can often turn decision algorithms into optimization algorithms directly. These more powerful data structures will be somewhat slower, but this is made up for by the savings of not going through parametric search. Indeed, this was the main idea of our previous 2-center algorithm [5]. We omit it from this extended abstract, but this idea reduces the time for one of the two cases in our new 2-center algorithm from $O(n \log^2 n)$ to $O(n \log n \log \log n)$; see the technical report version [6] for details.

Randomization. Sharir’s algorithm for the case of the 2-center problem in which the disks have a large overlap involves searching a certain sorted array by applying the circumradius decision data structure to paths in the array. We can apply the decision data structure without parametric search, if we only know a median value among the array entries on the path. Finding this median deterministically seems to be difficult (unless we return to the exact circumradius data structure) but as in randomized quickselect, if we choose an entry randomly we are likely to be near the median. Our resulting algorithm is randomized rather than deterministic, but that shortcoming seems to be only of theoretical interest.

One can also of course do away with parametric search altogether by replacing it with a binary search in the parameter space. However this premature replacement of combinatorics by numerics always loses a logarithmic factor over the decision algorithm, and does not help to find combinatorial information about the eventual solution (such as which two or three points determine the optimal 2-center radius).

3 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 two-part case analysis: either the disks are far apart (the ratio of the distance between centers to r^* is significantly greater than 0), or the disks are close to concentric (the ratio is significantly less than 2). As we do not know a priori which case the input falls into, we perform the algorithms for both cases and return the best two-disk cover returned by either.

If D_1 and D_2 are far apart, we find $O(1)$ 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. Carefully combining this decision algorithm with parametric search yields an $O(n \log^2 n)$ time bound.

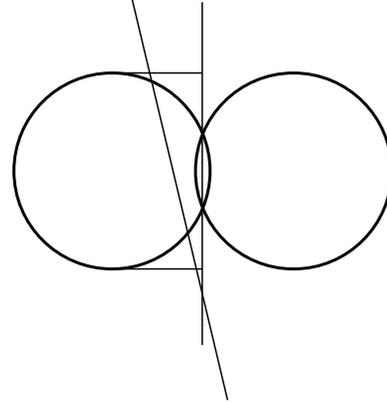


Figure 1. If disks are nearly tangent, a line separates most of D_1 from D_2 .

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 $O(n \log n \log \log n)$. In this abstract we only describe the offline decision problem solution, which is simpler than the combined solution and suffices to achieve our overall $O(n \log^2 n)$ bound.

4 Well separated disks

We first consider the case in which the centers of D_1 and D_2 are separated by a distance of at least ϵr^* . This case (or more specifically, the case in which the disks are nearly tangent) turns out to be the bottleneck of the overall algorithm.

LEMMA 4.1. *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.

Proof. 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 1); 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. \square

COROLLARY 4.1. *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 .*

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 an offline circumradius decision problem data structure [10].

Filling in the details of this outline (as we do below) results in an algorithm for solving the decision problem for this case in time $O(n \log n)$. Sharir [16] 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 a slower offline decision problem data structure.

To use this result to actually find r^* , we apply parametric search to simulate a parallelization of the above decision algorithm, using the decision algorithm again to simulate each step. To speed this search up, we show that only a small portion of the decision algorithm depends on the input parameter r , and that that portion can easily be parallelized.

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.

The offline sequence of insertions and deletions described above can then be found by performing $O(n)$ parallel binary search operations in the circular hull, to determine which vertex of S the moving circle is pivoting around when it crosses the given point, together with one sorting algorithm to determine the sequence of crossings for each separate hull vertex. As we now show, this sequence changes discontinuously at r^* , so we can terminate the simulated algorithm after the construction of this sequence, without simulating the application of the circumradius decision data structure to it.

LEMMA 4.2. *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

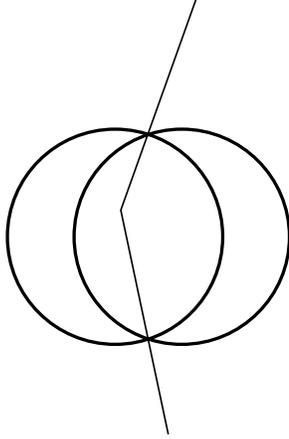


Figure 2. If disks are nearly concentric, input can be partitioned by rays from a point in their intersection.

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 $r = r^*$. \square

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 a collection of binary searches and a sorting algorithm) is suitable for Cole's parametric search speedup [3]. Thus we have the following result:

LEMMA 4.3. *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 $O(n \log^2 n)$.*

5 Nearly concentric disks

We now describe a solution in the remaining case, when the disks we seek are nearly concentric. Like Sharir [16], we reduce the problem to one of searching a certain $n \times n$ matrix, using techniques related to methods of Frederickson and Johnson [7] for selection in sorted matrices.

LEMMA 5.1. *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 5.1. *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 .*

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 . 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 2). 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; without loss of generality 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 $k = \log_2 n + O(1)$.

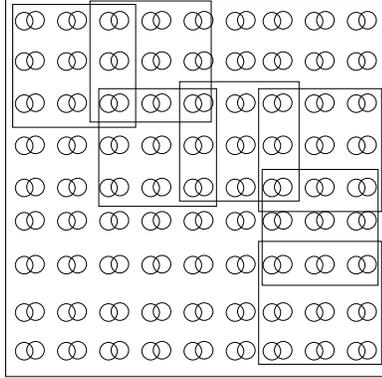


Figure 3. 9×9 matrix with a path of 3×3 squares.

5.1 Search with offline exact circumradius

We now describe how to find the optimal entry in M , assuming a data structure for evaluating entries of M . We will later show how to modify this method to use the circumradius decision problem data structure instead of exact evaluations.

LEMMA 5.2. *After evaluating a single entry of M , we can determine that either the quadrant of M above and right or the quadrant below and left of the evaluated entry does not contain the optimal entry.*

Proof. The points in one of those two quadrants correspond to adding more points to the larger of the two circles, which can never make it smaller. \square

LEMMA 5.3. *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 3 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 of one square, the 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 $2^{k-1-i} + 1$, 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. \square

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

5.2 Search with offline decision problem

The first observation is that any path through matrix M corresponds to an offline sequence of insertions and deletions in a dynamic circumradius problem (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 squares at any stage form a path. Instead, we assume that at stage i we have $c2^i$ squares for some constant $c > 1$. As before, each stage subdivides each square into four, producing $c2^{i+2}$ squares; to continue the process we wish to reduce this number of squares to $c2^{i+1}$. 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 5.4. *Suppose we have $(1 + b)2^i$ squares of size $2^{k-i} + 1$ remaining in M , for some $b > 0$. Then in expected time $O(n(1 + b) \log n)$ we can eliminate $\Omega(b2^i)$ squares from the collection.*

Proof. In this situation, there must be $\Omega(b2^i)$ interior corners of squares. We choose a random entry among these interior corners, and compute its value r with a linear time circumradius algorithm. We then connect the interior square corner entries into a path and use the offline decision algorithm of Hershberger and Suri to test for each 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 the expected number of rounds needed is $O(1)$. \square

LEMMA 5.5. *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 5.3, 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. \square

Combining this method with the exact evaluation method sketched earlier produces a further speedup of this case to $O(n \log n \log \log n)$ time; we defer details to the technical report version of this paper [6].

6 Conclusions

The algorithms for the two cases described above together yield the following result:

THEOREM 6.1. *We can find r^* (and a two-disk cover with radius r^*) in expected time $O(n \log^2 n)$.*

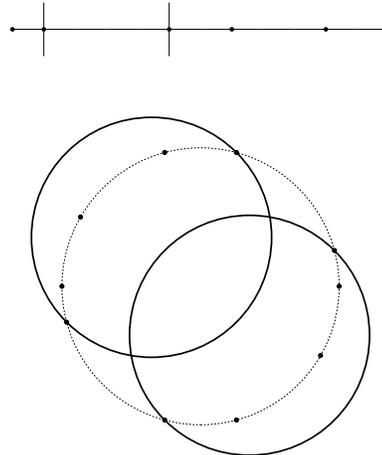


Figure 4. Max gap problem and corresponding two-center solution.

The following lower bound was observed by Welzl (personal communication).

THEOREM 6.2. *Any deterministic algorithm for the two-center problem requires $\Omega(n \log n)$ time in the algebraic decision tree model of computation.*

Proof. We reduce the max gap problem to the two-center problem. In the max gap problem, we are given $n + 1$ values x_0, x_1, \dots, x_n , not in sorted order, and we wish to determine the largest distance between adjacent values that would exist if the values were sorted. Without loss of generality we can assume that $x_0 = 0$, $x_n = 1$, and $0 \leq x_i \leq 1$ for all i .

To transform this problem to the two-center problem, we map each value x_i to two opposite points on the unit circle, $(\cos(\pi x_i), \sin(\pi x_i))$ and $(-\cos(\pi x_i), -\sin(\pi x_i))$. Each gap between adjacent values is thereby transformed into a symmetric pair of arcs on the circle, with arc length proportional to the gap size (Figure 4.)

Any subset of points on the circle has circumradius at most one, with circumradius less than one if and only if the subset includes points forming an arc of length at most π . Therefore, the optimal two-disk cover of these points will be formed by separating them on the pair of arcs corresponding to the longest gap of the original values. The radius of the disk cover is determined by the length of this gap, so this reduction applies as well to the decision versions of the two problems.

Since max gap is known to have an $\Omega(n \log n)$ lower bound [2, 13, 15], the same bound applies to the two-center problem. \square

A recent paper [9] extends some deterministic algebraic decision tree lower bounds to the randomized algebraic decision tree model, but it is not clear whether these results apply to max gap or to the two-center problem. Note also that these bounds are quite sensitive to the assumed model of computation, indeed, if one allows the use of a (non-algebraic) integer truncation operation, max gap has a deterministic linear time solution [8, 15].

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. There is only one bottleneck in our current algorithm: the case of nearly tangent disks. (The case of overlapping disks can be sped up to $O(n \log n \log \log n)$, and it turns out that if the disk centers are separated by more than $(2 + \epsilon)r^*$ the optimal disk pair can be found in linear time.) We achieved faster time bounds in other parts of the algorithm by avoiding parametric search; perhaps it can be avoided in the bottleneck case as well. Alternately, perhaps we can find a deterministic two-center algorithm with performance matching our randomized algorithm. Again, there is only one bottleneck: the randomized selection used in 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] M. Ben-Or. Lower bounds for algebraic computation trees. *15th ACM Symp. Theory of Computing*, 1983, pp. 80–86.
- [3] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. Assoc. Comput. Mach.*, vol. 34, 1987, pp. 200–208.
- [4] Z. Drezner. The planar two-center and two-median problems. *Transportation Science* 18 (1984) 351–361.
- [5] 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.
- [6] D. Eppstein. Faster construction of planar two-centers. Tech. Rep. 96-12, Dept. of Information and Computer Science, Univ. of California, Irvine, 1996. Available online at <http://www.ics.uci.edu/Document/UCI:ICS-TR-96-12>.
- [7] 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.
- [8] T. Gonzalez. Algorithms on sets and related problems. Tech. Rep., Dept. of Computer Science, Univ. of Oklahoma, 1975.
- [9] D. Grigoriev, M. Karpinski, F. Meyer auf der Heide, and R. Smolensky. A lower bound for randomized algebraic decision trees. *28th ACM Symp. Theory of Computing*, 1996, pp. 612–619.
- [10] J. Hershberger and S. Suri. Offline maintenance of planar configurations. *2nd ACM-SIAM Symp. Discrete Algorithms*, 1991, 32–41.
- [11] J. Jaromczyk and M. Kowaluk. An efficient algorithm for the Euclidean two-center problem. *10th ACM Symp. Computational Geometry*, 1994, pp. 303–311.
- [12] M. Katz and M. Sharir. An expander-based approach to geometric optimization. *9th ACM Symp. Computational Geometry*, 1993, pp. 198–207.
- [13] D. T. Lee and Y. F. Wu. Geometric complexity of some location problems. *Algorithmica*, vol. 1, 1986, pp. 193–211.
- [14] N. Megiddo. Applying parallel computation algorithms in the design of sequential algorithms. *J. ACM* 30, 1983, pp. 852–865.
- [15] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.
- [16] M. Sharir. A near-linear algorithm for the planar 2-center problem. *12th ACM Symp. Computational Geometry*, 1996, pp. 106–112.