More Planar Two-Center Algorithms

Timothy M. Chan*

August 19, 1997

Abstract

This paper considers the *planar Euclidean two-center problem*: given a planar n-point set S, find two congruent circular disks of the smallest radius covering S. The main result is a deterministic algorithm with running time $O(n \log^2 n \log^2 \log n)$, improving the previous $O(n \log^9 n)$ bound of Sharir and almost matching the randomized $O(n \log^2 n)$ bound of Eppstein. If a point in the intersection of the two disks is given, then we can solve the problem in $O(n \log n)$ time with high probability.

Keywords: two-center, randomization, parametric search

1 Introduction

Consider the following "facility location" problem: given a set S of n "demand" points in $\mathbb{R}^d$ and a number p, find a set T of p "supply"points in $\mathbb{R}^d$ minimizing $\max_{s \in S} \min_{t \in T} d(s,t)$, where $d(s,t)$ denotes the Euclidean distance between s and t. Geometrically, the problem is equivalent to finding p congruent disks of the smallest radius covering S and is referred to as the *(Euclidean) p-center problem*. The 1-center problem can be solved in worst-case $O(n)$ time for any fixed dimension d [5, 10, 19]; simple randomized $O(n)$-time methods are also known [6, 24].

The next "easiest" case, the 2-center problem in two dimensions, is the subject of several recent papers in the computational geometry literature. Hershberger and Suri [14] considered the weaker problem of deciding whether S can be covered by two disks of radius r for a given r. They showed that this decision problem can be solved in $O(n^2 \log n)$ time (a small improvement was subsequently noted by Hershberger [13]). Agarwal and Sharir [1] used this result in conjunction with the powerful *parametric-search* paradigm, invented by Megiddo [18], to obtain an $O(n^2 \log^3 n)$-time algorithm for the two-center problem. Later, Katz and Sharir [17] showed how parametric search can be avoided with the use of expanders; the running time remained the same. Using a randomized approach, Eppstein [11] gave an algorithm with expected running time $O(n^2 \log^2 n \log \log n)$. Afterwards, Jaromczyk and Kowaluk [16] gave a deterministic $O(n^2 \log n)$-time algorithm based on new geometric insights of the problem. In the appendix, we describe another $O(n^2 \log n)$ solution obtained directly from the decision algorithm of Hershberger and Suri and a simple application of randomization.

In a major breakthrough, Sharir [22] showed that the planar 2-center problem can actually be solved in near-linear time. The time bound of his algorithm is $O(n \log^9 n)$ and can be improved.

*Dept. of Math. and Computer Science, University of Miami, Coral Gables, FL 33124–4250, `tchan@cs.miami.edu`

Indeed, shortly after the announcement of Sharir's result, Eppstein [12] gave such an improvement with a randomized algorithm running in $O(n \log^2 n)$ expected time. This paper describes further refinements of Sharir and Eppstein's deterministic and randomized algorithms.

The strategy behind these near-linear-time methods is to divide the problem into two cases: (i) when the centers of the two optimal disks are well-separated, and (ii) when the centers of the two optimal disks are close together. Let $r_{\rm opt}$ denote the radius of the optimal disks, δ be the distance between the centers of the disks, and let $c \in (0,2)$ be a fixed constant.

Case 1: $\delta \geq cr_{\rm opt}$. Sharir showed that in this instance, deciding whether S can be covered by two disks of radius r for a given r can be done in $O(n \log^2 n)$ time. By applying parametric search, one can find $r_{\rm opt}$ in $O(n \,\mathrm{polylog}\, n)$ time. Eppstein noted that a $\log n$ factor can be removed in the decision problem if one uses an "offline" data structure of Hershberger and Suri [15]. In combination with an improved parametric search scheme, this leads to a deterministic $O(n \log^2 n)$-time algorithm to find $r_{\rm opt}$.

Case 2: $\delta < cr_{\rm opt}$. Here, the intersection of the two disks contains a disk of radius $(1 - c/2)r_{\rm opt}$, while the smallest disk enclosing S has radius at most $2r_{\rm opt}$. One can generate a constant number of points so that at least one of them lies in the intersection of the two disks. The problem then reduces to a constant number of the following problem, where $o \in \mathbb{R}^2$ is a fixed point, which we may assume to be the origin:

> **The Restricted 2-Center Problem.** Find two congruent disks D_1, D_2 of the smallest radius such that $S \subseteq D_1 \cup D_2$ and $o \in D_1 \cap D_2$.

Sharir considered a decision version of the above problem and gave an $O(n \log^3 n)$-time algorithm; the exact version is then solved by parametric search in $O(n \log^9 n)$ time. Eppstein solved the above problem directly, using randomization, in $O(n \log n \log \log n)$ expected time.

We will consider *Case 2* in this paper. We show that the restricted 2-center problem can be solved in $O(n \log n)$ time with high probability, thus improving on Eppstein's randomized method by a $\log \log n$ factor. Our method is also simpler than Eppstein's. Furthermore, it prepares us for the description of our deterministic algorithm, which uses parametric search and runs in time $O(n \log^2 n \log^2 \log n)$. To obtain this running time, we need an efficient parallel algorithm for a certain two-dimensional convex programming problem; such an algorithm is provided by a recent paper of Chan [3]. In addition, a number of tricks are employed to further speed up the parametric search.

With Eppstein's worst-case $O(n \log^2 n)$ bound for *Case 1*, we can now solve the planar 2-center problem in $O(n \log^2 n)$ time with high probability, or in $O(n \log^2 n \log^2 \log n)$ time deterministically.

2 Preliminaries

We begin with notation and simple observations, most of which were noted in earlier papers [12, 22].

On the circumradius. Given an arbitrary planar point set T, let $\rho(T)$ denote the radius of the smallest disk enclosing $T \cup \{o\}$. For any point $p = (a, b)$, let $h(p)$ denote the halfspace obtained from the standard lifting map:

$$\{(x, y, z) : z \geq -2ax - 2by + a^2 + b^2\}.$$

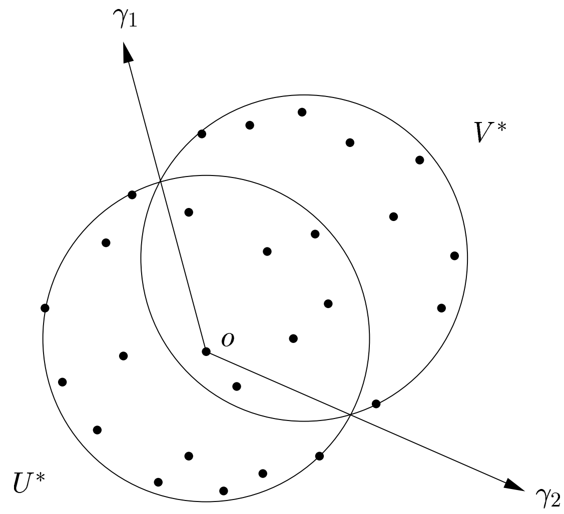

Figure 1: The restricted 2-center problem.

Then the square of $\rho(T)$ is the minimum of $x^2 + y^2 + z$ over all points (x, y, z) in the polyhedron $\mathcal{P}(T) = \bigcap_{p \in T \cup \{o\}} h(p)$. Let $B_r(p)$ denote the closed disk with center p and radius $r > 0$. We have $\rho(T) < r$ iff the intersection $\mathcal{I}_r(T) = \bigcap_{p \in T \cup \{o\}} B_r(p)$ has a nonempty interior.

On the restricted 2-center problem. Now, let S be the given planar point set. Consider an optimal pair of disks D_1 and D_2, of radius r^*, for the restricted 2-center problem. Consider the two intersection points of ∂D_1 and ∂D_2, and let γ_1 and γ_2 be rays through these points emanating from the origin o. We know that the rays γ_1 and γ_2 are separated by at least one of the two coordinate axes, since the angle between them lies in $[\pi/2, 3\pi/2]$. Without loss of generality, we may assume that they are separated by the x-axis.

Let S^+ (S^-) be the set of points of S above (below) the x-axis. We may assume that S^+ and S^- are both sets containing exactly n points. Sort the points of S^+ and S^- radially around the origin in counterclockwise order. Let the sorted order be $\langle p_1, \ldots, p_n \rangle$ for S^+ and $\langle q_1, \ldots, q_n \rangle$ for S^-. For each $i = 0, \ldots, n$ and $j = 0, \ldots, n$, define

$$U_{ij} = \{p_{i+1}, \ldots, p_n\} \cup \{q_1, \ldots, q_j\}, \quad \text{and} \quad V_{ij} = \{p_1, \ldots, p_i\} \cup \{q_{j+1}, \ldots, q_n\}.$$

Observe that $r^* = \max\{\rho(U^*), \rho(V^*)\}$ where $\{U^*, V^*\}$ is a partition of S formed by the two rays γ_1 and γ_2 (see Figure 1). Furthermore, $\{U^*, V^*\} = \{U_{ij}, V_{ij}\}$ for some i and j. Thus,

$$r^* = \min_{0 \le i,j \le n} \max\{A[i,j], B[i,j]\},$$

where we define $A[i,j] = \rho(U_{ij})$ and $B[i,j] = \rho(V_{ij})$. Our goal of finding the optimal radius r^* then reduces to a search problem on the matrices A and B. Having found r^*, we can easily identify the optimal disks D_1 and D_2 afterwards.

On the matrix search problem. Note that the matrices A and B satisfy the following monotonicity properties: for each $i = 0, \ldots, n$ and $j = 0, \ldots, n$,

$$\begin{array}{ccccccc} A[i,0] & \leq & A[i,1] & \leq & \cdots & \leq & A[i,n], \\ A[0,j] & \geq & A[1,j] & \geq & \cdots & \geq & A[n,j], \\ B[i,0] & \geq & B[i,1] & \geq & \cdots & \geq & B[i,n], \\ B[0,j] & \leq & B[1,j] & \leq & \cdots & \leq & B[n,j]; \end{array}$$

For convenience, we extend the matrices as follows: $A[i,-1] = B[i,n+1] = B[-1,j] = 0$ and $A[i,n+1] = A[-1,j] = B[i,-1] = \infty$ for each $i = 0, \ldots, n$ and $j = -1, \ldots, n$.

For each $i = 0, \ldots, n$, we let

$$r^{(i)} = \min_{0 \leq j \leq n} \max\{A[i,j], B[i,j]\}$$

be the "i-th row minimum," so that $r^* = \min_{0 \leq i \leq n} r^{(i)}$. The following inequality—a direct consequence of the monotonicity properties along a row—will be of use later: for each $j = -1, \ldots, n$,

$$r^{(i)} \geq \min\{A[i,j+1], B[i,j]\} \tag{1}$$

3 A Randomized Algorithm

Before we give our randomized algorithm, we first describe data structures that we need. The first, noted by Eppstein [12], is used for evaluating entries of the matrices A and B.

Lemma 3.1 *With $O(n \log n)$ preprocessing time, we can compute $A[i,j]$ and $B[i,j]$ in $O(\log^6 n)$ time for any given i, j.*

Proof: Using a segment tree on the lists $\langle p_1, \ldots, p_n \rangle$ and $\langle q_1, \ldots, q_n \rangle$, we can construct a collection of canonical subsets of total size $O(n \log n)$ such that every set of the form $\{p_{i+1}, \ldots, p_n\}$ or $\{q_1, \ldots, q_j\}$ can be written as a union of $O(\log n)$ of these canonical subsets. For each canonical subset $T \subseteq S$, construct a hierarchical representation [9] of the polyhedron $\mathcal{P}(T)$; this can be done in $O(n \log n)$ time by constructing the polyhedra in a bottom-up fashion and using Chazelle's linear-time polyhedra intersection algorithm [4].

Given i, j, we write U_{ij} as a union of $k = O(\log n)$ canonical subsets $T_1, \ldots, T_k$. Now, the square of $A[i,j] = \rho(U_{ij})$ is the minimum of $x^2 + y^2 + z$ over all $(x, y, z) \in \mathcal{P}(U_{ij}) = \mathcal{P}(T_1) \cap \cdots \cap \mathcal{P}(T_k)$. Eppstein [11] showed that minimizing a convex function over the intersection of k polyhedra can be found in $O(k^3 \log^3 n)$ time using hierarchical representations. So, $A[i,j]$ can be computed in $O(\log^6 n)$ time. We can compute $B[i,j]$ in a similar manner. $\square$

Remark: The $O(\log^6 n)$ query time can probably be improved; see Eppstein [11] or Chan [3] for ideas. However, at present, these approaches do not yield a deterministic bound near $O(\log^2 n)$ or a randomized bound near $O(\log n)$, so this data structure alone is not sufficient to derive our results.

Next we note that a data structure with logarithmic query time can be obtained for a weaker problem: comparing a matrix entry with a fixed number r.

Lemma 3.2 *Fix $r > 0$. With $O(n \log n)$ preprocessing time, we can decide whether $A[i,j] < r$ and whether $B[i,j] < r$ in $O(\log n)$ time for any given i, j.*

Proof: Preparata [20] gave an online algorithm for constructing planar convex hulls in $O(n \log n)$ time. It is a straightforward exercise to extend the online algorithm to other similar configurations in the plane, such as intersections of halfplanes or intersections of congruent disks. As a consequence, in $O(n \log n)$ time we can compute the intersection $\mathcal{I}_r(\{p_{i+1}, \ldots, p_n\})$ for each i from n down to 0. Note that $\mathcal{I}_r(\{p_{i+1}, \ldots, p_n\})$ is a convex "arc-gon" and can be represented as a sequence of arcs. Persistent search trees [21] allow us to retrieve a binary-searchable version of each such sequence in logarithmic time (note that the total structural change of the intersection is at most linear). In a similar manner, we can also compute the intersection $\mathcal{I}_r(\{q_1, \ldots, q_j\})$ for each j from 0 up to n.

Given i, j, we can now decide whether $\mathcal{I}_r(U_{ij}) = \mathcal{I}_r(\{p_{i+1}, \ldots, p_n\}) \cap \mathcal{I}_r(\{q_1, \ldots, q_j\})$ has a nonempty interior in $O(\log n)$ time, since we can apply standard logarithmic-time methods for convex polygons to detect whether two convex arc-gons intersect. Hence, whether $A[i,j] < r$ can be decided in $O(\log n)$ time, and the related question for the matrix B can be answered in the same way. □

Remark: If amortized time bounds are sufficient, then persistent search trees can be avoided. We will only apply the lemma to entries along a monotone path of the matrix, so we just need a "transcript" recording the changes made during the online algorithm, and the ability to play the transcript backwards.

Now we solve the decision problem—comparing r^* with a given number r—using Lemma 3.2 and a straightforward linear search. A similar search technique is noted recently by Devillers and Katz [8]. Previously, Sharir [22] described a more complicated and less efficient search algorithm, using an iterative scheme with $O(\log n)$ phases.

Theorem 3.3 *Given $r > 0$, we can decide whether $r^* < r$ in $O(n \log n)$ time. Furthermore, we can return the set of indices $\{i : r^{(i)} < r\}$ within the same time bound.*

Proof: The following algorithm prints the desired set of indices (nothing is printed iff $r^* \geq r$):

1. $j \leftarrow -1$
2. for $i \leftarrow 0$ up to n do
3. while $A[i, j+1] < r$ do $j \leftarrow j+1$
4. print i iff $B[i,j] < r$

As A is monotone increasing along each row and monotone decreasing along each column, we have the invariant $A[i,j] < r$. After step 3, the index j is the largest such that $A[i,j] < r$. If $B[i,j] < r$, then $r^{(i)} < r$; otherwise $r^{(i)} \geq r$ by (1). Thus, the algorithm is correct. Since $O(n)$ entries of A and B are compared with r, the theorem follows from Lemma 3.2. □

The above theorem yields a simple algorithm to compute r^* by random sampling. Previously, following the approach of Sharir [22], Eppstein [12] described a procedure to find r^* in $O(n \log n \log \log n)$ expected time using a complicated "hybrid search."

Theorem 3.4 *We can compute r^* by an algorithm that runs in $O(n \log n)$ time with probability $1 - 2^{-\Omega(n/\log^{12} n)}$.*

Proof: Given an index i, we can evaluate $r^{(i)}$ as follows:

1. find largest $j \in \{-1, \ldots, n\}$ with $A[i,j] < B[i,j]$
2. return $\min\{A[i,j+1], B[i,j]\}$

Clearly, $r^{(i)} \leq A[i,j+1]$ and $r^{(i)} \leq B[i,j]$; the correctness of the algorithm is then immediate from (1). Since $A - B$ is monotone increasing along each row, step 1 can be done by binary search using $O(\log n)$ evaluations of the matrices A and B. By Lemma 3.1, the above algorithm finds $r^{(i)}$ in $O(\log^7 n)$ time after preprocessing.

Now, pick $m = \lfloor n/\log^6 n \rfloor$ indices i uniformly at random from $\{0, \ldots, n\}$ and evaluate $r^{(i)}$; this step requires $O(n \log n)$ time. Let r be the minimum of these at most m numbers. Find all other indices i with $r^{(i)} < r$ in $O(n \log n)$ time by Theorem 3.3 and evaluate $r^{(i)}$ for each such i. Then r^* is the minimum of r and these numbers.

The total running time of this method is bounded by $O(n \log n)$, if the number of indices i with $r^{(i)} < r$ is bounded by m. Define the *rank* of an index i to be the position of $r^{(i)}$ in a sorted ordering of the multiset $\{r^{(i)} : 0 \leq i \leq n\}$. The probability that $|\{i : r^{(i)} < r\}| > m$ is no greater than the probability of picking m indices uniformly at random from $\{0, \ldots, n\}$, each having rank $> m$; this probability is $(1 - m/(n+1))^m = 2^{-\Omega(n/\log^{12} n)}$. $\square$

4 A Deterministic Algorithm

It does not appear that the algorithm in Theorem 3.4 (or Eppstein's algorithm) can be efficiently derandomized. We note, however, that the algorithm in Theorem 3.3 for deciding whether $r^* < r$ does not use randomization. To find r^* deterministically, we will use this decision algorithm in combination with the well-known *parametric-search* technique [7, 18]. In what follows, we assume that the reader is familiar with this technique; for instance, see earlier papers on the 2-center problem [1, 12, 22] or the survey by Agarwal and Sharir [2].

In order to apply parametric search, we need an efficient parallel version of the decision algorithm; we do not have to parallelize preprocessing steps that do not depend on the parameter r, and we can use Valiant's comparison model of computation [23]. Unfortunately, the algorithm in Theorem 3.3 is inherently sequentially. Moreover, the algorithm uses the data structure in Lemma 3.2, and the preprocessing of this data structure (which heavily depends on r) is inherently sequential as well. We first rectify the latter problem by giving an alternative data structure for comparing entries of A and B for a given r. The following gives a parallel algorithm for comparing $O(n)$ matrix entries with r simultaneously, using a recent technique of Chan for two-dimensional convex programming [3].

Lemma 4.1 *We can preprocess S in $O(n \log n)$ time so that given $r > 0$ and $O(n)$ pairs of indices, we can decide whether $A[i,j] < r$ and whether $B[i,j] < r$ for every pair (i,j) in $O(\log n \log\log n)$ parallel steps using $O(n \log n)$ processors.*

Proof: The preprocessing of S is done as in the proof of Lemma 3.1: build canonical subsets of total size $O(n \log n)$ and construct the polyhedron $\mathcal{P}(T)$ for each canonical subset T. We may assume that the facets of the polyhedra have been triangulated. This preprocessing requires $O(n \log n)$ time and is done independently of r.

Now, given r, we construct the intersection $\mathcal{I}_r(T)$ for each canonical subset T by intersecting the facets of $\mathcal{P}(T)$ with the paraboloid $\{(x,y,z) : x^2+y^2+z=r^2\}$ and projecting them vertically to the xy-plane. This step can be carried out easily in parallel logarithmic time with $O(n \log n)$ processors. We can store the sequence of arcs of each $\mathcal{I}_r(T)$ in an array.

Given a pair (i,j), we write U_{ij} as a union of $k = O(\log n)$ canonical subsets $T_1, \ldots, T_k$. Deciding whether $A[i,j] < r$ reduces to deciding whether the intersection $\mathcal{I}_r(U_{ij}) = \mathcal{I}_r(T_1) \cap \cdots \cap \mathcal{I}_r(T_k)$ has a nonempty interior. In a recent paper [3], we give algorithms for detecting a common intersection of k convex n-gons; the approach applies to the arc-gons $\{\mathcal{I}_r(T_i)\}_{i=1}^{k}$ as well. One of the algorithms, which has sequential running time $O(k \log k \log n)$, can be parallelized:

This algorithm proceeds in $O(\log k)$ iterations and attempts to find the leftmost point v^* in the intersection. In each iteration, we perform a constant number of "oracle calls" to determine which side of certain lines v^* lies on. Such an oracle is answered by intersecting each arc-gon with the given lines and requires $O(k)$ independent binary searches, each taking $O(\log n)$ time. All $O(\log k)$ oracle calls can be implemented in $O(\log k \log n)$ time with $O(k)$ processors. In addition, the algorithm performs $O(k)$ work per iteration, which we can afford to do sequentially as k is small. We refer the reader to the paper [3] for further details. It can be checked that for $k = O(\log n)$, the algorithm takes $O(\log n \log\log n)$ parallel steps using $O(\log n)$ processors.

To compare $O(n)$ entries of A with r, we assign $O(\log n)$ processors to each entry and apply the above parallel algorithm. Comparing entries of B with r can be done similarly. □

Next we give a new decision algorithm. It only makes $O(\log\log n)$ calls to the algorithm in the above lemma, and hence can be efficiently parallelized.

Theorem 4.2 *We can preprocess S in $O(n \log n)$ time so that given $r > 0$, we can decide whether $r^* < r$ in $O(\log n \log^2 \log n)$ parallel steps using $O(n \log n)$ processors.*

Proof: Let $m = \lfloor n/\log^6 n \rfloor$ as before, and let $j_p = p\lfloor n/m \rfloor$ for $p = 1, \ldots, m-1$. Set $j_0 = -1$ and $j_m = n+1$. For each j_p, find the largest index i_p with $A[i_p, j_p] > B[i_p, j_p]$. (Note that since $A - B$ is monotone increasing along each row, we have $-1 = i_0 \le i_1 \le \cdots \le i_m = n$.) Since $A-B$ is monotone decreasing along each column, we can find each index i_p by binary search using $O(\log n)$ evaluations of the matrices A and B. By Lemma 3.1, the search takes $O(\log^7 n)$ time after preprocessing. This computation requires $O(n \log n)$ time total and is done independently of r.

Now, given r, we decide whether $r^{(i)} < r$ for each $i = 0, \ldots, n$ as follows:

1. let $p \in \{0, \ldots, m-1\}$ be such that $i_p < i \le i_{p+1}$
2. if $A[i, j_p] \ge r$ then return "no"
3. find largest $j \in \{j_p, \ldots, j_{p+1}\}$ with $A[i,j] < r$
4. return "yes" iff $B[i,j] < r$

As $i_p < i \le i_{p+1}$, we have $A[i, j_p] \le B[i, j_p]$ and $A[i, j_{p+1}] > B[i, j_{p+1}]$. If $A[i, j_p] \ge r$, then $r^{(i)} \ge r$ by (1) and "no" is returned in step 2. If $A[i, j_{p+1}] < r$, then $r^{(i)} < r$ and "yes" is returned in step 4. Otherwise, the largest j with $A[i,j] < r$ lies between j_p and j_{p+1} and is found in step 3. As in the proof of Theorem 3.3, we have $B[i,j] < r$ iff $r^{(i)} < r$. The correctness of the algorithm thus follows.

Since A is monotone increasing along each row, step 3 can be performed by binary search using $O(\log(j_{p+1} - j_p)) = O(\log\log n)$ comparisons of matrix entries with r. Therefore, we can determine all indices $\{i : r^{(i)} < r\}$ in $O(\log\log n)$ rounds of comparisons of $O(n)$ matrix entries with r, which can be resolved by Lemma 4.1. $\square$

We are now in position to apply parametric search.

Theorem 4.3 *We can compute r^* in $O(n\log^2 n\log^2\log n)$ time.*

Proof: Given a decision algorithm with sequential running time T_S and a decision algorithm with parallel running time T_P using P processors, the parametric-search technique [18] finds the optimal solution r^* in $O(PT_P + T_ST_P\log P)$ time by simulating the parallel algorithm on r^* (using the sequential algorithm to resolve comparisons with r^*). In our application, $T_S = O(n\log n)$ by Theorem 3.3, and $T_P = O(\log n\log^2\log n)$ and $P = O(n\log n)$ by Theorem 4.2. So, an $O(n\log^3 n\log^2\log n)$ time bound is obtained.

Cole [7] described an improved parametric-search technique that achieves running time $O(PT_P + T_S(T_P + \log P))$, assuming that the parallel decision algorithm satisfies a "bounded fan-in/fan-out" requirement. Our parallel algorithm can easily be made to satisfy the requirement. The overall running time is reduced to $O(n\log^2 n\log^2\log n)$. $\square$

Remark: Our application of parametric search is actually not too complex, as our parallel algorithm consists mainly of $O(\log^2\log n)$ stages of independent binary searches. In particular, sorting networks are not needed here (but are needed in Eppstein's application [12] of parametric search for *Case 1*).

5 Conclusions

With the previous work of Sharir and Eppstein, the results in this paper imply the following: there is a randomized algorithm that solves the (unrestricted) planar 2-center problem in $O(n\log^2 n)$ time with high probability; furthermore, there is a deterministic algorithm with performance almost matching the randomized algorithm, settling a question posed by Eppstein [12], if one ignores the small $\log^2\log n$ factor. Our algorithms make use of refinements of various techniques, including matrix searching, parametric searching, and data structures for low-dimensional convex programming.

Our results raise the next question: is there an $o(n\log^2 n)$ algorithm (deterministic or randomized) for the planar 2-center problem? To answer this, we need a faster algorithm in the case of well-separated centers (*Case 1*).

References

[1] P. K. Agarwal and M. Sharir. Planar geometric location problems. *Algorithmica*, 11:185–195, 1994.

[2] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. Tech. Report CS-1996-19, Dept. of Computer Science, Duke Univ., Durham, 1996.

[3] T. M. Chan. Deterministic algorithms for 2-d convex programming and 3-d online linear programming. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 464–472, 1997. Full version at http://www.cs.miami.edu/~tchan/cp.ps.gz.

[4] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21:671–696, 1992.

[5] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21:579–597, 1996.

[6] K. L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42:488–499, 1995.

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

[8] O. Devillers and M. J. Katz. Optimal line bipartitions of point sets. In *Proc. 7th Int. Sympos. Algorithms and Computation*, Lect. Notes in Comput. Sci., vol. 1178, Springer-Verlag, pages 45–54, 1996. *Int. J. Comput. Geom. Appl.*, to appear.

[9] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra: A unified approach. In *Proc. 17th Int. Colloq. Automata, Languages, and Programming*, Lect. Notes in Comput. Sci., vol. 443, Springer-Verlag, pages 400–413, 1990.

[10] M. E. Dyer. On a multidimensional search technique and its application to the Euclidean one-centre problem. *SIAM J. Comput.*, 15:725–738, 1986.

[11] D. Eppstein. Dynamic three-dimensional linear programming. *ORSA J. Comput.*, 4:360–368, 1992.

[12] D. Eppstein. Faster construction of planar two-centers. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 131–138, 1997. Also as Tech. Report 96-12, Dept. of Information and Computer Science, Univ. of California, Irvine, 1996.

[13] J. Hershberger. A faster algorithm for the two-center decision problem. *Inform. Process. Lett.*, 47:23–29, 1993.

[14] J. Hershberger and S. Suri. Finding tailored partitions. *J. Algorithms*, 12:431–463, 1991.

[15] J. Hershberger and S. Suri. Off-line maintenance of planar configurations. *J. Algorithms*, 21:453–475, 1996.

[16] J. Jaromczyk and M. Kowaluk. An efficient algorithm for the Euclidean two-center problem. In *Proc. 10th ACM Sympos. Comput. Geom.*, pages 303–311, 1994.

[17] M. Katz and M. Sharir. An expander-based approach to geometric optimization. In *Proc. 9th ACM Sympos. Comput. Geom.*, pages 198–207, 1993.

[18] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30:852–865, 1983.

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

[20] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[21] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29:669–679, 1986.

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

[23] L. Valiant. Parallelism in comparison problems. *SIAM J. Comput.*, 4:348–355, 1975.

[24] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science* (H. Maurer, ed.), Lect. Notes in Comput. Sci., vol. 555, Springer-Verlag, pages 359–370, 1991.

Appendix

In this appendix, we outline an $O(n^2 \log n)$ randomized algorithm for the (unrestricted) planar 2-center problem using a data structure of Hershberger and Suri. The advantage of this algorithm lies in its simplicity. Furthermore, as it does not use any of the special geometry of the 2-center problem, with appropriate data structures the approach can perhaps be generalized to related problems such as the construction of weighted 2-centers.

We find it convenient to redefine some of the notation previously used. Let $\rho(T)$ to be the radius of the smallest disk enclosing T, and let $\mathcal{I}_r(T) = \bigcap_{p \in T} B_r(p)$. Let the given point set be $S = \{p_1, \ldots, p_n\}$. Consider an optimal pair of disks D_1 and D_2 of radius r^* of the unrestricted problem. Consider the two intersection points of ∂D_1 and ∂D_2 and let ℓ be the line through the two points. Let U_{ij} be the set of points in S that is below the line through p_i and p_j (including p_i and p_j); let V_{ij} be the set of points in S strictly above this line.

Observe that $r^* = \max\{\rho(U^*), \rho(V^*)\}$ where $\{U^*, V^*\}$ is the partition of S formed by the line ℓ. Furthermore, $\{U^*, V^*\} = \{U_{ij}, V_{ij}\}$ for some i and j. Thus,

$$r^* = \min_{1 \le i,j \le n} \max\{\rho(U_{ij}), \rho(V_{ij})\}.$$

Now, let

$$r^{(i)} = \min_{1 \le j \le n} \max\{\rho(U_{ij}), \rho(V_{ij})\}.$$

By using a linear-time method for the 1-center problem, we can evaluate each $r^{(i)}$ in $O(n^2)$ time. We note that whether $r^{(i)} < r$ can be decided in $O(n \log n)$ time: This problem reduces to finding a line through p_i so that in the resulting partition $\{U, V\}$ of S, both $\mathcal{I}_r(U)$ and $\mathcal{I}_r(V)$ have nonempty interiors. We can generate all $O(n)$ such partitions using a rotational line sweep around p_i, and we can use the offline data structure of Hershberger and Suri [15] to maintain the intersections $\mathcal{I}_r(U)$ and $\mathcal{I}_r(V)$.

As a result, the following simple algorithm finds the minimum $r^* = \min_{1 \le i \le n} r^{(i)}$: set $r = \infty$; for each $i = 1, \ldots, n$ in random order, test whether $r^{(i)} < r$ in $O(n \log n)$ time, and if so, evaluate $r^{(i)}$ in $O(n^2)$ time and reset $r = r^{(i)}$. It is clear that this procedure computes r^*. Let t be the number of indices i for which $r^{(i)}$ is evaluated by this procedure. A standard exercise in algorithm analysis reveals that the expected value of t is bounded by the n-th harmonic number $H_n = O(\log n)$. It follows that the algorithm runs in $O(n^2 \log n)$ expected time.