Selecting Distances in the Plane*

Pankaj K. Agarwal,[†] Boris Aronov,[‡]
Micha Sharir,[§] and Subhash Suri[¶]

Abstract

We describe a randomized algorithm for computing the k^{th} smallest distance in a set of n points in the plane, based on the parametric search technique of Megiddo [Me1]. The expected running time of our algorithm is $O(n^{4/3}\log^{8/3} n)$. A deterministic version of our procedure runs in time $O(n^{3/2}\log^{5/2} n)$. Both versions improve the previously best known upper bound of $O(n^{9/5}\log^{4/5} n)$ by Chazelle [Ch]. A simple $O(n \log n)$ time algorithm for computing an approximation of the median distance is also presented.

1. Introduction

Given a set S of n points in the plane, let $d_1 \leq \cdots \leq d_{\binom{n}{2}}$ be the distances determined by pairs of points in S. For a given positive integer $k \leq \binom{n}{2}$, we want to determine the value of d_k and find a pair that realizes d_k. Explicitly computing all the distances and applying the linear-time selection algorithm of Blum et al. [BFPRT] (see also [SPP]) solves the problem in time $O(n^2)$. The challenge is to produce the answer in subquadratic time. Such an algorithm, based on Yao's batching technique [Ya], has been given by Chazelle [Ch]. His approach actually works in any dimension, and computes the k^{th} smallest distance in a set of n points in $\mathbb{R}^d$ in time $O(n^{2-\beta(d)}\log^{1-\beta(d)} n)$, where $\beta(d) = 1/(2^d+1)$. Thus, for $d = 2$, the running time is $O(n^{9/5}\log^{4/5} n)$. Salowe [Sa] has shown that, in L_1 or L_∞ metric in the plane, the above problem can be solved in time $O(n\log^2 n)$. He also extended Chazelle's algorithm to general L_p metric, in which case the algorithm (for points in the plane) runs in time $O(n^{2-\gamma(p)}\log^{1-\gamma(p)} n)$, where $\gamma(p) = 1/2^{2p-1}$ if p is even and $\gamma(p) = 1/2^{2p+1}$ if p is odd.

The main result of our paper is a randomized algorithm for computing the k^{th} smallest Euclidean distance in a set of n points in the plane with expected running time $O(n^{4/3}\log^{8/3} n)$. A deterministic version of the algorithm runs in (worst-case) time $O(n^{3/2}\log^{5/2} n)$. In addition, our algorithms can be extended to general L_p metric, with a small increase in time complexity—the randomized algorithm takes $O(n^{4/3}\log^3 n)$ time, and the deterministic one takes $O(n^{3/2}\log^3 n)$ time, independent of p.

These algorithms are purely combinatorial and produce the exact value of the k^{th} smallest distance and a pair of points realizing it. If one is willing to compromise on these issues, simpler solutions are available. For example, in Section 5, we present a deterministic $O(n\log n)$ algorithm for finding an approximate median distance—it identifies a pair of points with the property that at least some fixed fraction of pairs are separated by a larger distance and some other fixed fraction by a smaller distance. We also describe a simpler version of our main algorithm, which computes the k^{th} smallest distance to b bits of accuracy in deterministic time $O(bn^{3/2}\log^{1/2} n)$, or in randomized expected time $O(bn^{4/3}\log^{2/3} n)$.

*Work by the first three authors has been partly supported by Office of Naval Research Grant N00014-87-K-0129, by National Science Foundation Grant DCR-83-20085, and by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science Foundation Science and Technology Center—NSF-STC88-09648. Work by the second author has also been partially supported by National Security Agency Grant MDA 904-89-H-2030. Work by the third author has also been supported by National Science Foundation Grant CCR-89-01484, and by grants from the U.S.-Israeli Binational Science Foundation, and the Fund for Basic Research administered by the Israeli Academy of Sciences.

[†]DIMACS, Rutgers University, Piscataway, NJ 08855, USA and Computer Science Department, Duke University, Durham, NC 27706, USA

[‡]DIMACS, Rutgers University, Piscataway, NJ 08855, USA

[§]Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel

[¶]Bellcore, Morristown, NJ 07960, USA

1.1 Parametric Search

Our selection algorithms are based on Megiddo's parametric search technique [Me1]. The basic idea behind his method is as follows: Suppose we have a problem $\mathcal{P}(d)$ that receives as input n data items and a real parameter d. We want to find a value d^* of d at which the output of $\mathcal{P}(d)$ satisfies certain properties. Suppose we have an efficient sequential algorithm A_s for solving $\mathcal{P}(d)$ at any given d, and that, as a by-product, the algorithm A_s can also determine whether the given d is equal to, less than, or greater than the desired value d^*. Assume moreover that the flow of execution of A_s depends on comparisons, each of which involves testing the sign of a low-degree polynomial in d and in the input items.

Megiddo's technique then runs the algorithm A_s "generically", without specifying the value of d, with the intention of simulating its execution at the unknown d^*. Each time a comparison is to be made, the few roots of the associated polynomial are computed, and we run A_s "off line" at each of them, thereby determining the location of d^* among these roots, and thus also the sign of the polynomial at d^*, i.e., the outcome of the comparison at d^*. Then execution of the generic A_s can be resumed. As we proceed through this execution, each comparison that we resolve further constrains the range where d^* can lie, and we thus obtain a sequence of progressively smaller intervals, each known to contain d^*, until we reach the end of the generic A_s with a final interval I. It follows that the outcome of A_s will be combinatorially the same when we run it on any $d \in I$, including d^*. If I is a singleton, we have found d^*; otherwise it is often straightforward to determine d^*, depending on the nature of the problem—in many cases any value $d^* \in I$ would do.

The cost of this implicit search is usually dominated by C_sT_s, where C_s is the maximum number of comparisons executed by A_s and T_s is the running time of A_s. Since this bound is generally too high, Megiddo suggests to replace the generic A_s by a parallel algorithm, A_p. If A_p uses Π processors and runs in T_p parallel steps, then each such step involves at most Π *independent* comparisons, that is, each can be carried out without having to know the outcome of the others. We can then compute the roots of all Π polynomials associated with these comparisons, and run a binary search to find the location of d^* among them (using, say, the serial algorithm A_s at each binary search step). This requires $O(\Pi + T_s \log \Pi)$ time per parallel step, for a total of $O(\Pi T_p + T_s T_p \log \Pi)$ time, which often results in a saving of nearly an order of magnitude in the running time. An improvement of this technique by Cole [Co] can further reduce the running time in certain cases by another logarithmic factor.

1.2 Ranking Problem

In our case the parametric search problem is finding d_k, the k^{th} smallest distance of S (this is the desired d^*), and the problem $\mathcal{P}(d)$, for a given d, is the *ranking problem*, which is defined as follows:

Given a set S of n points in the plane and a real number $d > 0$, determine the number $\mathcal{N}^-(d)$ of pairs of points in S whose distance is smaller than d, and the number $\mathcal{N}^+(d)$ of pairs of points in S whose distance is at most d.

Note that, having computed $\mathcal{N}^-(d)$ and $\mathcal{N}^+(d)$, we can compare d with d_k as follows:

- $d < d_k$ if $\mathcal{N}^+(d) < k$;
- $d = d_k$ if $\mathcal{N}^-(d) < k \leq \mathcal{N}^+(d)$;
- $d > d_k$ if $\mathcal{N}^-(d) \geq k$.

Thus our problem fits well into Megiddo's parametric search paradigm, and the technique described above can indeed be applied. This requires the design of efficient serial and parallel algorithms for the ranking problem.

Although for our purpose it is sufficient to use Valiant's model [Va] of parallel computation, we will derive algorithms that are also efficient under the weaker concurrent-read concurrent-write (CRCW) PRAM model. In this model several processors are allowed to read simultaneously from the same memory location as well as to write simultaneously to the same location, but the latter is allowed only if they attempt to write the same value; see [Vi] for a survey of results concerning PRAM models.

We present two techniques for solving the ranking problem. The first, described in Section 2, is deterministic and yields a sequential algorithm with running time $O(n^{3/2} \log^{1/2} n)$, and a parallel algorithm with $O(\log n)$ parallel steps, using $O(n^{3/2})$ processors. The second approach, presented in Section 3, is a randomized variant of the first. It yields an improved serial algorithm with expected running time $O(n^{4/3} \log^{2/3} n)$. A parallel implementation of this algorithm requiring expected $O(\log n)$ parallel steps and $O(n^{4/3})$ processors is also obtained.

In Section 4 we combine these algorithms with Megiddo's technique to obtain efficient solutions to the k^{th} smallest distance problem—a deterministic $O(n^{3/2} \log^{5/2} n)$ algorithm, and a randomized algorithm with expected running time $O(n^{4/3} \log^{8/3} n)$. Here we assume a model of computation in which the roots of a polynomial of fixed degree can be computed in constant time.

In Section 5 we describe a simple, deterministic $O(n \log n)$ algorithm for computing an approximate median distance, that is, we return a distance whose rank is between $c_1\binom{n}{2}$ and $c_2\binom{n}{2}$, for two fixed fractions c_1, c_2, with $0 < c_1 < 1/2 < c_2 < 1$.

In Section 6 we discuss our results, present several extensions and generalizations of our algorithms, and state several related open problems.

2 A Simple Ranking Algorithm

Our goal is to produce an algorithm which, given a set S of n points in the plane and a positive real number d, computes the number $\mathcal{N}^-(d)$ of pairs of points whose distance is smaller than d, and the number $\mathcal{N}^+(d)$ of pairs of points whose distance is at most d. It is not difficult to see that it is sufficient to solve efficiently the following problem:

> *Given a set $\mathcal{D}$ of n open congruent discs of radius $d > 0$ and a set $\mathcal{P}$ of m points, compute the number of pairs $(p, D) \in \mathcal{P} \times \mathcal{D}$ such that p lies in D, and the number of pairs (p, D) such that p lies in the closure $\bar{D}$ of D.*

An obvious approach to solving this problem is to test, for each pair (p, D), whether p lies in D or in $\bar{D}$, which requires $\Omega(mn)$ time. The main result of this section is a more efficient procedure for this problem: it runs in time $O(n(m \log m)^{1/2} + m \log m)$; a parallel version takes $O(\log m + \log n)$ time using $O(n\sqrt{m}+m)$ processors. Later in Section 3, we further improve the time bounds by allowing randomization. The outline of our first algorithm is as follows.

1. Construct the arrangement $\mathcal{A}(\mathcal{D})$ of the n circles bounding the discs of $\mathcal{D}$. $\mathcal{A}(\mathcal{D})$ partitions the plane into $O(n^2)$ vertices, edges, and faces (see [Ed] for details concerning arrangements of lines and [CL, EGPPSS] for arrangements of circles in the plane).
2. Refine $\mathcal{A}(\mathcal{D})$ by extending two vertical segments up and down from each vertex of $\mathcal{A}(\mathcal{D})$ and from the two points of vertical tangency of each of the given discs, until they intersect a disc boundary. If there is no such intersection, the vertical segment is extended to infinity (see figure 1). It is easily checked that the resulting planar map, referred to as the *vertical decomposition* of $\mathcal{A}(\mathcal{D})$ and denoted by $\mathcal{A}^*(\mathcal{D})$, still consists of $O(n^2)$ regions ("trapezoids"), each in general bounded by two vertical segments and two circular arcs, though some trapezoids may be degenerate. The resulting trapezoids, edges, and vertices form a true partition of the plane, i.e., trapezoids are considered open and edges—relatively open.

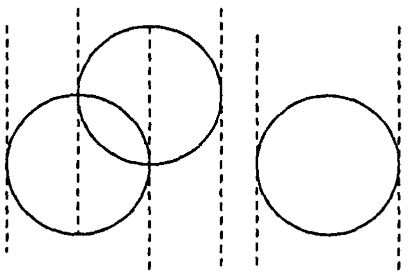

Figure 1: Vertical decomposition of an arrangement of discs

3. For each trapezoid τ (resp. edge e, vertex v) of $\mathcal{A}^*(\mathcal{D})$, compute the number c_τ (resp. c_e, c_v) of open discs of $\mathcal{D}$ covering it. In addition, compute for each vertex v of $\mathcal{A}(\mathcal{D})$ the number c_v^* of circles passing through v.
4. Preprocess $\mathcal{A}^*(\mathcal{D})$ for efficient point location.
5. For each point $p \in P$, determine the feature $\phi(p)$ of $\mathcal{A}^*(\mathcal{D})$ containing p—this is either a trapezoid $\tau(p)$, an edge $e(p)$, or a vertex $v(p)$.
6. The desired number $\mathcal{N}^-(d)$ of containments between points and open discs is $\sum_{p \in \mathcal{P}} c_{\phi(p)}$. The number $\mathcal{N}^+(d)$ of containments between points and closed discs is
$$\mathcal{N}^-(d) + \delta + \sum_{p \in V} c^*_{v(p)},$$
where δ is the number of points of P lying on edges of $\mathcal{A}(\mathcal{D})$, and $V \subseteq \mathcal{P}$ is the set of points that lie at vertices of $\mathcal{A}(\mathcal{D})$.

2.1 A Sequential Algorithm

We first present a sequential algorithm implementing the above procedure in time $O(n^2 + m \log n)$, slower than promised above; the better time bound will be obtained in Section 2.3 by an easy modification.

In time $O(n^2)$, we compute $\mathcal{A}(\mathcal{D})$ using the incremental approach of Chazelle and Lee [CL]. We then extend vertical segments from points of vertical tangency of each circle. For each circle C this can be easily done in $O(n)$ time by computing the intersections of each vertical tangent of C with all other circles and then choosing the ones that lie immediately above and below the point of tangency. Let $\mathcal{M}$ denote the resulting planar map; all its faces are simply connected and are monotone in the x-direction. We can therefore compute $\mathcal{A}^*(\mathcal{D})$ from $\mathcal{M}$ as follows.

For each face f of $\mathcal{M}$, walk along its upper and lower boundaries in lock-step and, for each vertex v on the lower (resp. upper) boundary, find the edge of the upper (resp. lower) boundary lying immediately above (resp. below) it; this process amounts to merging the vertices on the upper and lower boundaries of f in x-order. The total time spent in computing $\mathcal{A}^*(\mathcal{D})$ is $O(n^2 + \sum_{f \in \mathcal{M}} |f|) = O(n^2)$, where $|f|$ denotes the number of edges bounding f. Note that if two adjacent trapezoids τ, τ' of $\mathcal{A}^*(\mathcal{D})$ share an edge, which is a portion of the boundary of a disc D, then the sets of discs covering τ, τ' respectively differ by the disc D, so $c_{\tau'} = c_\tau \pm 1$; if τ, τ' share a vertical boundary segment, then $c_{\tau'} = c_\tau$. Thus the quantities c_τ, for all trapezoids τ of $\mathcal{A}^*(\mathcal{D})$, can be easily computed in $O(n^2)$ time. The quantities c_e, c_v, and c_v^*, for edges e and vertices v of $\mathcal{A}^*(\mathcal{D})$ can be computed in a similar fashion. Observe that $\sum_{v \in \mathcal{A}^*(\mathcal{D})} c_v^* = O(n^2)$, so we can even afford to maintain explicitly the sets of discs passing through each vertex of $\mathcal{A}(\mathcal{D})$.

Finally, $\mathcal{A}^*(\mathcal{D})$ is a monotone planar subdivision, therefore it can be preprocessed, in time $O(n^2)$, into a data structure of size $O(n^2)$ so that a point location query can be answered in time $O(\log n)$ [EGS]. We then locate the points of $\mathcal{P}$ in $\mathcal{A}^*(\mathcal{D})$, and sum up the associated quantities $c_{\phi(p)}$, $c^*_{v(p)}$. By putting all these steps together, we conclude that the numbers $\mathcal{N}^-(d)$, $\mathcal{N}^+(d)$ can be determined in time $O(n^2 + m \log n)$.

2.2 A Parallel Algorithm

We next derive a parallel version of the algorithm—again, not quite as efficient as promised. Although, in order to apply Megiddo's technique, it suffices to assume Valiant's model of parallelism for the ranking procedure, we present an efficient algorithm under the weaker CRCW PRAM model. Recall that we allow simultaneous writes by many processors to the same memory location only if they all write the same information. The arrangement $\mathcal{A}^*(\mathcal{D})$ can be constructed in this model in $O(\log n)$ time using $O(n^2)$ processors, as follows.

We calculate the points of intersection of every pair of circles, and then sort around each circle the points lying on it, thereby obtaining the edges of $\mathcal{A}(\mathcal{D})$. In addition, the edges incident to each vertex of $\mathcal{A}(\mathcal{D})$ are sorted around the vertex. We next compute the vertical segments emanating from points of vertical tangency. Each such edge is computed in $O(\log n)$ time using $O(n/\log n)$ processors—simply calculate all intersections of the vertical tangent with the remaining circles and choose the ones nearest to the point of tangency, using the parallel algorithm of [SV] for finding the minimum. Finally, each new endpoint of these segments is located in the appropriate circle and the edge containing it is split into two subedges—this is a trivial parallel step, requiring $O(\log n)$ time and $O(n)$ processors. This produces the planar map $\mathcal{M}$, which is then refined to obtain $\mathcal{A}^*(\mathcal{D})$, as described earlier. We merge the upper and lower boundaries of a face f by applying a variant of a parallel merge procedure that runs in $O(\log|f|)$ parallel time using $O(|f|)$ processors [SV]. It follows easily that the procedure just outlined computes $\mathcal{A}^*(\mathcal{D})$ in $O(\log n)$ parallel time, using $O(n^2)$ processors.

We now compute the quantities c_τ, c_e, c_v, $c_v^\star$ for all features of $\mathcal{A}^*(\mathcal{D})$. Let the *adjacency graph* of $\mathcal{A}^*(\mathcal{D})$ be the planar graph dual to $\mathcal{A}^*(\mathcal{D})$, i.e., its nodes are the trapezoids of $\mathcal{A}^*(\mathcal{D})$ and two vertices are joined by an arc if and only if the corresponding trapezoids share an edge of $\mathcal{A}^*(\mathcal{D})$. As mentioned earlier, for two adjacent trape-

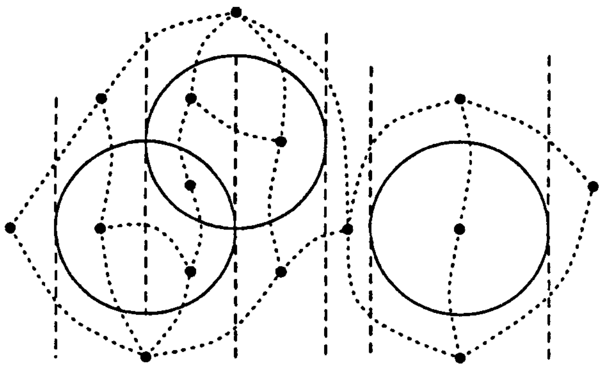

Figure 2: $\mathcal{A}^*(\mathcal{D})$ and its dual graph

zoids τ_1 and τ_2, $|c_{\tau_1} - c_{\tau_2}| \leq 1$ and the exact relationship between them is determined uniquely by the nature of the edge separating τ_1 from τ_2 and the location of τ_1, τ_2 relative to that edge. We compute an arbitrary spanning tree of the adjacency graph, rooted at some unbounded trapezoid τ_0 of $\mathcal{A}^*(\mathcal{D})$, and convert it into an Eulerian path π by traversing each edge of the tree twice, once in each direction [TVs]. To a node of π corresponding to a trapezoid $\tau \neq \tau_0$ we assign the weight $w_\tau = c_\tau - c_{pred(\tau)}$, where $pred(\tau)$ is the predecessor of τ in π. For the root τ_0, we put $w_{\tau_0} = c_{\tau_0} = 0$. Since c_τ is simply the sum $\sum w_{\tau'}$ over all nodes $\tau' \in \pi$ between τ_0 and τ, all these quantities can be computed in $O(\log n)$ parallel time and $O(n^2)$ processors, using the parallel prefix-sum algorithm of [TVs]. The quantities c_e, c_v and $c_v^\star$ can be computed in a similar fashion—the easy details are omitted.

Finally, since $\mathcal{A}^*(\mathcal{D})$ is a monotone planar subdivision, the parallel algorithm of Tamassia and Vitter [TVt] can preprocess $\mathcal{A}^*(\mathcal{D})$, in time $O(\log n)$ using $O(n^2)$ processors, so that point location queries can be answered in $O(\log n)$ time. By performing m queries, one for each point $p \in \mathcal{P}$, we obtain the quantities $c_{\phi(p)}$, which we can then sum in parallel time $O(\log m)$. In a similar manner we can obtain the sum $\sum c^\star_{v(p)}$ and count the number of points on edges of $\mathcal{A}(()\mathcal{D})$. Therefore, we now have a parallel (CRCW) algorithm for computing the number of point-disc containments between a set $\mathcal{P}$ of m points and a set $\mathcal{D}$ of n congruent discs (open or closed). The algorithm runs in parallel time $O(\log n + \log m)$ and uses $O(n^2 + m)$ processors.

2.3 An Easy Improvement

Using a well-known balancing technique, the performance of both the sequential and the parallel versions of the algorithm can be improved as follows. In the sequential case, partition $\mathcal{D}$ arbitrarily into $\lceil n/(m \log m)^{1/2} \rceil$ subsets each of size at most $(m \log m)^{1/2}$, and apply the above sequential algorithm to each subset of $\mathcal{D}$ separately; for each subset of discs, the entire point-set $\mathcal{P}$ is considered. The running time becomes

$$O(\left\lceil \frac{n}{\sqrt{m \log m}} \right\rceil m \log m) = O(n(m \log m)^{1/2} + m \log m).$$

Following a similar approach, but partitioning $\mathcal{D}$ into $\lceil \frac{n}{m^{1/2}} \rceil$ subsets of size at most $\sqrt{m}$, we produce a parallel version that still requires $O(\log m + \log n)$ time, but uses only $O(n\sqrt{m} + m)$ processors. In summary, we have proved the following theorem.

Theorem 2.1 *Given a set $\mathcal{P}$ of m points and a set $\mathcal{D}$ of n congruent open discs in the plane, the number of pairs of the form (p, D), with $D \in \mathcal{D}$ and $p \in \mathcal{P} \cap D$, and the number of pairs of the form (p, D), with $D \in \mathcal{D}$ and $p \in \mathcal{P} \cap \bar{D}$, can both be computed in time $O(n(m \log m)^{1/2} + m \log m)$ by a sequential algorithm, or in time $O(\log m + \log n)$ by a parallel algorithm (in the CRCW model) using $O(n\sqrt{m} + m)$ processors.*

Recalling our original motivation for the ranking problem, the following corollary is an immediate consequence of the previous theorem.

Corollary 2.2 *Given a set of n points in the plane and a positive real number d, we can compute the numbers $\mathcal{N}^-(d)$, $\mathcal{N}^+(d)$ of pairs of points with distance smaller than d or at most d, respectively, in time $O(n^{3/2}\sqrt{\log n})$ by a sequential algorithm, or in time $O(\log n)$ by a CRCW-parallel algorithm using $O(n^{3/2})$ processors.*

Remark: In the above algorithms, the set $\mathcal{D}$ can be replaced by any collection of objects of simple shape (e.g. objects whose boundary is an algebraic curve of some fixed degree). In this case, however, we have to assume that certain operations involving these objects, like computing the intersection points of a pair of object boundaries, finding the points of vertical tangency on the boundary of an object, finding the intersections of an object with a vertical line, etc., can each be performed in $O(1)$ time. One must also be able to sort a set of points along the boundary of an object efficiently. Most of the steps in the above algorithm extend to this general case. The only step that requires modification is the construction of $\mathcal{A}(\mathcal{D})$ in the sequential case. To carry out this construction, we replace the algorithm of Chazelle and Lee [CL] by the line-sweep procedure of Bentley and Ottmann [BO]. The running time of the latter algorithm is slightly worse: $O(n^2 \log n)$. This causes a similar small increase in the running time of our point-object containment algorithm. We leave details to the reader.

In view of the above remark, we can solve the ranking problem in any L_p-metric by drawing an L_p-unit disc around each point $q \in \mathcal{P}$, that is the region

$$\{(x,y) \in \mathbb{R}^2 : |x - x(q)|^p + |y - y(q)|^p \leq 1\},$$

and counting the number of point-disc containments. Thus, we obtain

Corollary 2.3 *Given a set of n points in the plane and a real positive number d, we can compute the numbers $\mathcal{N}^-(d)$, $\mathcal{N}^+(d)$ of pairs of points with L_p-distance smaller than d or at most d, respectively, in time $O(n^{3/2}\log n)$ by a sequential algorithm, or in time $O(\log n)$ by a parallel algorithm using $O(n^{3/2})$ processors.*

3 Randomized Ranking Algorithms

In this section we significantly improve both the sequential and the parallel versions of the ranking algorithm by allowing randomization.

Recall that, given a set $\mathcal{D}$ of n congruent discs and a set $\mathcal{P}$ of m points, we wish to compute the number of point-disc containments for both the open and the closed discs. We start by randomly selecting a set $\mathcal{R}$ of r discs from $\mathcal{D}$; all subsets of size r are equally likely in our probability distribution. We will fix the parameter r later—it is not assumed to be a constant. We form the arrangement $\mathcal{A}(\mathcal{R})$ of the circles bounding the sample discs and construct the corresponding vertical decomposition $\mathcal{A}^*(\mathcal{R})$; recall that $\mathcal{A}^*(\mathcal{R})$ is a planar partition consisting of $O(r^2)$ "trapezoids". In computing $\mathcal{A}^*(\mathcal{R})$, we also record for each face of $\mathcal{A}(\mathcal{R})$, the list of trapezoids covered by it, and for each trapezoid of $\mathcal{A}^*(\mathcal{R})$, the face of $\mathcal{A}(\mathcal{R})$ containing it. In addition, for each trapezoid τ, we compute: (1) c_τ, the number of discs in $\mathcal{D}$ whose interiors completely cover it, (2) $\mathcal{D}_\tau$, the set of discs in $\mathcal{D}$ whose boundaries cut τ, and (3) $\mathcal{P}_\tau$, the set of points in $\mathcal{P}$ that fall into τ. In an analogous manner, for each edge e of $\mathcal{A}^*(\mathcal{R})$, we compute c_e, the number of discs in $\mathcal{D}$ whose interiors cover e, $\mathcal{D}_e$, the set of discs whose boundaries cut e, and $\mathcal{P}_e$, the set of points in $\mathcal{P}$ lying in e. Finally, for each vertex v of $\mathcal{A}^*(\mathcal{R})$ that belongs to $\mathcal{P}$, we compute c_v, the number of open discs containing v, and c_v^*, the number of disc boundaries passing through v.

Let $n_\tau = |\mathcal{D}_\tau|$, $m_\tau = |\mathcal{P}_\tau|$, $n_e = |\mathcal{D}_e|$, and $m_e = |\mathcal{P}_e|$. Denoting by $\mathcal{N}^-(\mathcal{P},\mathcal{D})$ (resp. $\mathcal{N}^+(\mathcal{P},\mathcal{D})$) the number of point-disc containments for the open (resp. closed) discs, we have

$$\begin{aligned}\mathcal{N}^-(\mathcal{P},\mathcal{D}) &= \sum_{\tau\in\mathcal{A}^*(\mathcal{R})} (\mathcal{N}^-(\mathcal{P}_\tau,\mathcal{D}_\tau) + c_\tau m_\tau) + \\ &\quad \sum_{e\in\mathcal{A}^*(\mathcal{R})} (\mathcal{N}^-(\mathcal{P}_e,\mathcal{D}_e) + c_e m_e) + \\ &\quad \sum_{v\in V} c_v, \qquad (1)\\ \mathcal{N}^+(\mathcal{P},\mathcal{D}) &= \sum_{\tau\in\mathcal{A}^*(\mathcal{R})} (\mathcal{N}^+(\mathcal{P}_\tau,\mathcal{D}_\tau) + c_\tau m_\tau) + \\ &\quad \sum_{e\in\mathcal{A}^*(\mathcal{R})} (\mathcal{N}^+(\mathcal{P}_e,\mathcal{D}_e) + c_e m_e) + \\ &\quad \sum_{v\in V} (c_v + c_v^*), \qquad (2)\end{aligned}$$

where $V \subseteq \mathcal{P}$ is the set of points lying on vertices of $\mathcal{A}^*(\mathcal{R})$. In summary, our procedure for computing $\mathcal{N}^-(\mathcal{P},\mathcal{D})$ and $\mathcal{N}^+(\mathcal{P},\mathcal{D})$ has the following high-level form:

1. Choose a random subset $\mathcal{R} \subseteq \mathcal{D}$ of r discs.
2. Compute the vertical decomposition $\mathcal{A}^*(\mathcal{R})$ of the arrangement of circles bounding the discs in $\mathcal{R}$.
3. Compute $\mathcal{P}_\tau$, $\mathcal{D}_\tau$, and c_τ for each trapezoid τ of $\mathcal{A}^*(\mathcal{R})$; compute $\mathcal{P}_e$, $\mathcal{D}_e$, and c_e for each edge e of $\mathcal{A}^*(\mathcal{R})$; and compute c_v, c_v^* for each vertex v of $\mathcal{A}^*(\mathcal{R})$ that belongs to $\mathcal{P}$.
4. For each trapezoid τ and each edge e of $\mathcal{A}^*(\mathcal{R})$ calculate the quantities $\mathcal{N}^-(\mathcal{P}_\tau, D_\tau)$, $\mathcal{N}^+(\mathcal{P}_\tau, D_\tau)$, $\mathcal{N}^-(\mathcal{P}_e, D_e)$, and $\mathcal{N}^+(\mathcal{P}_e, D_e)$ using an appropriate variant of the algorithm of Section 2.
5. Obtain $\mathcal{N}^-(\mathcal{P},\mathcal{D})$ and $\mathcal{N}^+(\mathcal{P},\mathcal{D})$ from equations (1), (2).

The performance of this algorithm depends mainly on two factors: efficiency of calculating $\mathcal{D}_\tau$, $\mathcal{P}_\tau$, c_τ etc., and a good choice of r to balance the cost of dividing the original problem into several smaller problems (steps 2–3) against the cost of solving these subproblems (step 4).

We now describe in detail how to compute efficiently $\mathcal{P}_\tau$, $\mathcal{D}_\tau$, and c_τ for each trapezoid τ, $\mathcal{P}_e$, $\mathcal{D}_e$ and c_e for each edge e, and c_v, c_v^* for each point $v \in V$.

3.1 Divide Step

We determine $\mathcal{P}_\tau$ and $\mathcal{P}_e$ by the standard point-location method: compute $\mathcal{A}^*(\mathcal{R})$, preprocess it for point location, and identify the trapezoid or the edge (if the point lies on the boundary of a trapezoid) containing each point of $\mathcal{P}$. This is easily accomplished in $O(r^2 + m \log r)$ sequential time, or $O(\log r)$ parallel time using $O(r^2 + m)$ processors (cf. Section 2).

Computation of the set $\mathcal{D}_\tau$ for each trapezoid τ is slightly more involved. We define the *horizon* of a circle C in $\mathcal{A}(\mathcal{R})$ to be the collection of faces of $\mathcal{A}(\mathcal{R})$ met by C. The size (i.e., the total number of edges) of the horizon of C is $O(r\alpha(r))$, where $\alpha(\cdot)$ is the functional inverse of the Ackermann's function [HS, EGPPSS]. Thus the number of trapezoids of $\mathcal{A}^*(\mathcal{R})$ meeting C is also bounded by $O(r\alpha(r))$.

Our sequential algorithm traces each circle C through $\mathcal{A}^*(\mathcal{R})$ to find all trapezoids it cuts. We begin by locating the leftmost point l of C in $\mathcal{A}^*(\mathcal{R})$, and thus implicitly in $\mathcal{A}(\mathcal{R})$, and then follow the procedure of Chazelle and Lee [CL] to identify faces of $\mathcal{A}(\mathcal{R})$ that meet C; observe that we are reporting the faces of $\mathcal{A}(\mathcal{R})$, not of $\mathcal{A}^*(\mathcal{R})$. This takes $O(r)$ time per circle. (Note that unlike Chazelle and Lee we do not actually insert C into the arrangement.) The procedure, when repeated for all circles C bounding discs in $\mathcal{D}$, runs in $O(n(r + \log r)) = O(nr)$ time. Now, let f be a face of $\mathcal{A}(\mathcal{R})$ met by C. For each trapezoid $\tau \in f$, we test whether τ intersects C. Since the total number of trapezoids in the faces of $\mathcal{A}(\mathcal{R})$ meeting C is $O(r\alpha(r))$, $\mathcal{D}_\tau$, for all trapezoids $\tau \in \mathcal{A}^*(\mathcal{R})$, can be computed in total time $O(nr\alpha(r))$.

As for a parallel procedure for computing the sets $\mathcal{D}_\tau$, we proceed as follows: For each circle C we compute its intersections with every sample circle in $\mathcal{R}$, sort them along C, and then locate each of them on the edges of $\mathcal{A}(\mathcal{R})$. Now we can easily identify the faces of $\mathcal{A}(\mathcal{R})$ met by C. All this can be done in $O(\log r)$ time using $O(r)$ processors per circle. Next, for each trapezoid τ lying in a face met by C, we determine whether τ intersects C. As the number of trapezoids in the faces of $\mathcal{A}(\mathcal{R})$ met by C is $O(r\alpha(r))$, we can use $O(r)$ processors to decide, in $O(\alpha(r))$ time, which trapezoids of the faces of $\mathcal{A}(\mathcal{R})$ met by C actually intersect C. The book-keeping associated with distributing processors among the trapezoids may require additional $O(\log n)$ time. Applying the same procedure to all circles of $\mathcal{D} - \mathcal{R}$, we can determine $\mathcal{D}_\tau$ in $O(\log n)$ time using $O(nr)$ processors. To summarize, the sets $\mathcal{D}_\tau$ of circles cutting τ, for all τ, can be computed in $O(nr\alpha(r))$ sequential time or $O(\log n)$ parallel time using $O(nr)$ processors. The sets $\mathcal{D}_e$, for all edges e of $\mathcal{A}^*(\mathcal{R})$, can be computed, in $O(nr\alpha(r))$ sequential time, or $O(\log n)$ parallel time using $O(nr)$ processors, by testing for intersection with e all the discs cutting one of the two trapezoids adjacent to e.

Consider next the task of computing the number of open discs covering each trapezoid. The following approach is suitable for both sequential and parallel versions of the algorithm. We construct the adjacency graph on the trapezoids of $\mathcal{A}^*(\mathcal{R})$, as defined in the preceding section. It is possible for the degree of a node in the adjacency graph to be arbitrarily large. For example, there may exist an arc of a sample circle which has a single trapezoid τ on one side, while a large number of trapezoids are adjacent to it on the opposite side. As a result, τ will be connected to all trapezoids lying across the arc from it. To deal with the possibility of such a situation, we modify the adjacency graph as follows.

In general, a trapezoid is bounded by two arcs of a circle on the top and bottom and by two vertical segments on the sides. Given two adjacent trapezoids τ, τ' lying on opposite sides of a circle C, let γ, γ' be the closed arcs of C lying on the boundary of τ, τ', respectively (for the purpose of this definition a circle is cut into lower and upper half at the points of vertical tangency). If γ is fully contained in the relative interior of γ' or vice versa, the edge connecting τ to τ' is removed from the adjacency graph. Apply a similar procedure to pairs of trapezoids sharing a vertical edge. It is easily checked that the resulting "thinned-down" adjacency graph is still connected and no node has degree higher than 8.

Next, obtain an arbitrary spanning tree of the modified adjacency graph. By replacing each edge of the tree by a pair of oppositely directed edges, we construct an Eulerian path π of length $O(r^2)$ (see figure 3). Let $\mathcal{C}_\tau$ denote the

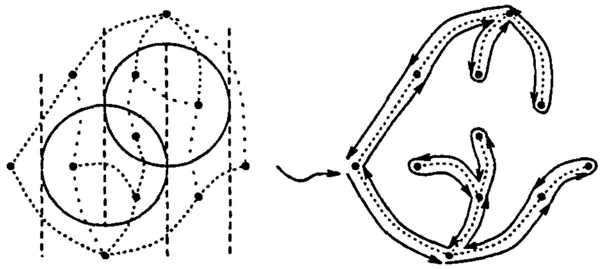
Figure 3: Eulerian path π built from a spanning tree.

collection of discs covering the trapezoid $\tau \in \mathcal{A}^*(\mathcal{R})$, thus $c_\tau = |\mathcal{C}_\tau|$. Let τ, τ' be two trapezoids corresponding to two adjacent nodes of π. It is easily seen that the symmetric difference of the sets $\mathcal{C}_\tau$ and $\mathcal{C}_{\tau'}$ is a subset of $\mathcal{D}_\tau \cup \mathcal{D}_{\tau'}$, and that, consequently, $w_\tau = c_\tau - c_{\tau'}$ can be computed in $O(n_{\tau'} + n_\tau)$ sequential time, by determining which of the circles in $\mathcal{D}_\tau \cup \mathcal{D}_{\tau'}$ cover τ but not τ', or vice versa. Thus the total sequential time spent in computing w_τ, for all nodes of π, is $O(\sum_\tau n_\tau)$, as π does not visit any node more than 8 times. Once w_τ is known, c_τ, for all $\tau \in \mathcal{A}^*(\mathcal{R})$, are computed in $O(r^2)$ time by a prefix sum algorithm. The total sequential time spent in computing c_τ is therefore $O(r^2 + \sum_\tau n_\tau) = O(nr\alpha(r))$.

As for the parallel version, it is easily seen that all the above steps can be performed in $O(\log n)$ time using $O(r^2 + \sum_\tau n_\tau)$ processors (see [TVs], [FL], [CV]).

Finally we compute the sets $\mathcal{D}_e$ and quantities c_e, c_v, and c_v^*, for all edges e and vertices v of $\mathcal{A}^*(R)$, using similar techniques. Their computation can be done within the same sequential or parallel bounds stated above. We leave it for the reader to fill in the easy missing details.

3.2 Time Complexity Analysis

We now return to the overall algorithm. The non-recursive terms in the equations (1) and (2), such as $\sum_\tau c_\tau m_\tau$, can now be readily computed. In addition, we have to compute and sum up the quantities $\mathcal{N}^-(\mathcal{P}_\tau, \mathcal{D}_\tau)$, $\mathcal{N}^+(\mathcal{P}_\tau, \mathcal{D}_\tau)$, for each trapezoid τ. Each of these quantities can be computed using the algorithm of Section 2, with the following twist. With every instance $(\mathcal{P}, \mathcal{D})$ of the ranking problem we can associate a "dual" instance $(\mathcal{D}^*, \mathcal{P}^*)$, where $\mathcal{D}^*$ is the set of centers of the discs in $\mathcal{D}$ and $\mathcal{P}^*$ is the collection of discs of radius d drawn around the points of $\mathcal{P}$. Clearly, the number of point-disc contain-

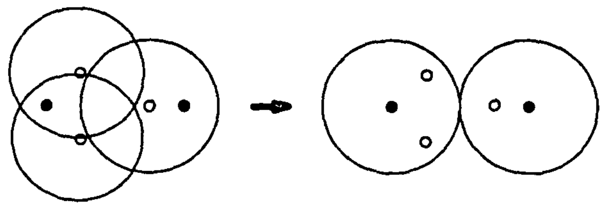

Figure 4: Dualizing the problem

ments (both for open and for closed discs) in an instance of the problem and in its dual instance are the same, as containment of a point p in a disc centered at q is equivalent to q lying in a congruent disc centered at p. For each τ, we transform the instance (P_τ, D_τ) into its dual $(\mathcal{D}^*_\tau, \mathcal{P}^*_\tau)$, and apply the procedures of Section 2 to the dual instance. This will take $O(m_\tau(n_\tau \log n_\tau)^{1/2} + n_\tau \log n_\tau)$ sequential time, or $O(\log n_\tau + \log m_\tau)$ parallel time using $O(m_\tau n_\tau^{1/2} + n_\tau)$ processors.

Finally, for an edge $e \in \mathcal{A}^*(\mathcal{R})$, $\mathcal{N}^-(\mathcal{P}_e, \mathcal{D}_e)$ and $\mathcal{N}^+(\mathcal{P}_e, \mathcal{D}_e)$ are computed as follows: Sort the intersection points of e and the circles bounding the discs of $\mathcal{D}_e$ along e; this gives a partition of e into at most $2n_e + 1$ subarcs. For every point p in a subarc, $\mathcal{N}^-(\{p\}, \mathcal{D}_e)$, $\mathcal{N}^+(\{p\}, \mathcal{D}_e)$ remain the same, and can be computed in $O(n_e)$ time over all subarcs. Now $\mathcal{N}^-(\mathcal{P}_e, \mathcal{D}_e)$, $\mathcal{N}^+(\mathcal{P}_e, \mathcal{D}_e)$ can be easily calculated in time $O(n_e + m_e \log n_e)$ by locating the points of $\mathcal{P}_e$ on e. Any circle cutting edge e, bounding trapezoid τ, cuts τ, and no circle cutting τ cuts the boundary of τ more than 8 times, so $\sum_e n_e \le 8 \sum_\tau n_\tau$. Obviously $\sum m_e \le m$. Thus, the total time spent in computing $\mathcal{N}^-(\mathcal{P}_e, \mathcal{D}_e)$, $\mathcal{N}^+(\mathcal{P}_e, \mathcal{D}_e)$ over all edges of $\mathcal{A}^*(\mathcal{R})$ is $O(\sum_\tau n_\tau \log n_\tau + m \log n)$. As to the parallel version, the above steps can be easily performed in time $O(\log n + \log m)$ using $O(\sum_\tau n_\tau + m)$ processors.

To summarize, the sequential time required to execute steps (2) and (3) of our algorithm is bounded by

$$O(r^2) + O(m \log r) + O(n r \alpha(r)) + O(\textstyle\sum_\tau n_\tau \log n_\tau) + O(m \log n) = O(n r \alpha(r) + \textstyle\sum_\tau n_\tau \log n_\tau + m \log n),$$

so the time performance $T(m, n)$ of the sequential algorithm satisfies the following equation.

$$T(m,n) = \sum_\tau O(m_\tau (n_\tau \log n_\tau)^{1/2} + n_\tau \log n_\tau) + O\left(n r \alpha(r) + m \log n\right).$$

Here the number of trapezoids τ is bounded by $O(r^2)$, and $\sum m_\tau \le m$. To obtain a good bound on $T(m, n)$ from the above equation, we need to control the size of each n_τ. This is where we bring into play the randomized nature of the algorithm. Applying the analysis of Clarkson and Shor [CS, Theorem 3.6], the expected value of $\sum_\tau n_\tau \log n_\tau$ is $O(\frac{n}{r} \log \frac{n}{r})$ times the expected number of trapezoids in $\mathcal{A}^*(\mathcal{R})$, which is $O(r^2)$. Secondly, for each point $p \in \mathcal{P}$, consider the quantity $(n_{\tau(p)} \log n_{\tau(p)})^{1/2}$, where $\tau(p)$ is the trapezoid of $\mathcal{A}^*(\mathcal{R})$ containing p. As a corollary of Theorem 3.6 of [CS], the expected value of this quantity is $O(\sqrt{\frac{n}{r} \log \frac{n}{r}})$. In this argument we have assumed that all points of $\mathcal{P}$ fall inside trapezoids of $\mathcal{A}^*(\mathcal{R})$; however the analysis is easily extended to handle points that fall on edges or vertices of the sample arrangement, observing that these points need not be processed at all in the calculation of the quantities $\mathcal{N}^-(\mathcal{P}_\tau, \mathcal{D}_\tau)$, $\mathcal{N}^+(\mathcal{P}_\tau, \mathcal{D}_\tau)$. We omit the easy details.

Summing over all points in $\mathcal{P}$, we therefore obtain

$$\begin{aligned} E\left[\sum_\tau m_\tau n_\tau^{1/2} \log^{1/2} n_\tau\right] &= E\left[\sum_{p \in \mathcal{P}} n_{\tau(p)}^{1/2} \log^{1/2} n_{\tau(p)}\right] \\ &= \sum_{p \in \mathcal{P}} E\left[n_{\tau(p)}^{1/2} \log^{1/2} n_{\tau(p)}\right] \\ &= O\left(\frac{mn^{1/2}}{r^{1/2}} \log^{1/2}(\frac{n}{r})\right). \end{aligned}$$

The above observations imply the following bound on the expected value of $T(m, n)$:

$$O\left(\frac{mn^{1/2}}{r^{1/2}} \log^{1/2}(\frac{n}{r}) + nr(\log \frac{n}{r} + \alpha(r)) + m \log n\right).$$

Choosing $r = \min\left\{\left\lceil \frac{m^{2/3}}{n^{1/3} \log^{1/3}(n^2/m)} \right\rceil, n\right\}$, we obtain

$$E[T(m,n)] = O\left(m^{2/3} n^{2/3} \log^{2/3}(\frac{n^2}{m}) + m \log n\right).$$

As for the parallel version of the algorithm, we can enhance its performance by the following "quality control" step: After we have computed the quantities m_τ, n_τ, we compute the sum $\sum_{\tau \in \mathcal{A}^*(\mathcal{R})} (m_\tau n_\tau^{1/2} + n_\tau)$, and if it does not exceed $2Cm^{2/3} n^{2/3}$, for some constant C to be chosen below, the algorithm continues, otherwise we restart it from step 1 with a new random sample (cf. page 5). As we will see later, this condition ensures that the number of processors is always $O(m^{2/3} n^{2/3} + m + n)$.

If we do not restart the algorithm, its running time is always $O(\log m + \log n)$, and, by an analysis similar to the above, the number of processors it requires is given by

$$\Pi(m,n) = O\left(\sum_\tau m_\tau n_\tau^{1/2} + \sum_\tau n_\tau + m\right).$$

Using the results of [CS], we can bound the expected value of this expression by

$$O\left(\frac{mn^{1/2}}{r^{1/2}} + nr + m\right). \tag{3}$$

Substituting $r = \min\left\{\lceil m^{2/3}/n^{1/3}\rceil, n\right\}$ yields

$$E[\Pi(m,n)] = C \cdot (m^{2/3}n^{2/3} + m + n), \tag{4}$$

where $C > 0$ is an absolute constant.

On the other hand, if we restart the algorithm whenever $\sum m_\tau n_\tau^{1/2} > 2Cm^{2/3}n^{2/3}$, where C is chosen as above, that is whenever the number of processors required exceeds $2C(m^{2/3}n^{2/3}+m+n)$, then, by (4), the probability that the algorithm is restarted t times is less than $1/2^t$. Therefore the expected running time of the parallel algorithm is $O(\log m + \log n)$. Summarizing our results and interpreting them in the context of the ranking problem, we obtain:

Theorem 3.1 *Given a set S of n points in the plane and a positive number d, the numbers $\mathcal{N}^-(d)$, $\mathcal{N}^+(d)$ of pairs of points of S closer than d apart and at most d apart, respectively, can be computed by a sequential randomized algorithm that requires $O(n^{4/3}\log^{2/3} n)$ expected time, or by a randomized parallel algorithm (in the CRCW PRAM model) that runs in $O(\log n)$ expected parallel time and uses $O(n^{4/3})$ processors.*

Remarks:

(i) If the algorithm is allowed to allocate processors as needed, we do not have to restart the algorithm, in which case the worst-case running time is $O(\log n)$, and the expected number of processors is $O(n^{4/3})$. Observe that the number of processors is always bounded by $O(n^2)$.

(ii) Unlike the algorithm of the previous section, the algorithms described here cannot be easily modified to count the number of point-disc containments for discs of different radii, because to compute $\mathcal{N}^-(\mathcal{P}_\tau, \mathcal{D}_\tau)$, $\mathcal{N}^+(\mathcal{P}_\tau, \mathcal{D}_\tau)$ we solve the dual problems $(\mathcal{D}^*_\tau, \mathcal{P}^*_\tau)$, and this duality does not work for arbitrary size discs. However, the duality goes through if $\mathcal{D}$ is a set of translates of some object, and therefore the algorithms can be extended to solve the ranking problem in general L_p metric—sequentially in $O(n^{4/3}\log n)$ expected time, or in expected time $O(\log n)$ using $O(n^{4/3})$ processors.

4 Computing the k$^{\text{th}}$ Distance

In this section we describe an algorithm to compute the k^{th} smallest distance, denoted d_k, determined by the pairs of points in S, for a given $0 < k \le \binom{n}{2}$. The problem can be reformulated as that of finding the distance d^* satisfying $\mathcal{N}^-(d^*) < k \le \mathcal{N}^+(d^*)$. Hence this problem can be solved by applying Megiddo's parametric search technique to the ranking problem, in the manner outlined in the introduction.

Recall that to this end we need to run one of the parallel algorithms of Section 3 for solving the ranking problem for d^*, without knowing the value of this parameter. Observe that these algorithms operate in stages, each consisting of a set of comparisons followed by some auxiliary computations. The auxiliary computations depend only on the outcome of the comparisons; they need not manipulate explicitly the parameter d, and can therefore be carried out without having to know the value of d. More specifically, all program's decisions can be stated in terms of replies to the following types of questions:

1. Is a given point inside (on the boundary of, outside) a given disc?
2. Do two discs intersect (touch)?
3. Are two given points distinct?
4. Given two points, what is the relative order of their x and y coordinates?
5. Given three points on a circle, what is their relative order around the circle?
6. Given three circles meeting in a point, in what cyclic order do they emanate from it?
7. Are two given points closer than (exactly, further than) d apart?

In all such questions, discs and circles come from the set $\mathcal{D}$. Points, on the other hand, are the input points from $\mathcal{P}$, the leftmost and rightmost points of circles, or points of intersection of two circles or of a circle and a vertical line tangent to another circle. A point that is not one of the input points is identified by the circle(s) or circle and line defining it—its coordinates are not constants but simple algebraic functions of d. It is not difficult to see that all of the above queries can indeed be answered by examining the sign of a fixed degree *characteristic polynomial* in d and the coordinates of a constant number of points.[1] For example, testing disjointness of two circles of radius d, centered at (x_1, y_1) and (x_2, y_2), respectively, is equivalent to checking that

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 - 4d^2 > 0.$$

One can easily verify that all decisions made by the ranking algorithms can be reduced to such sign tests. For example, to compute the arrangement $\mathcal{A}(\mathcal{D})$ we first need to identify all its vertices, i.e., all pairs of discs whose boundaries intersect; then we need to sort the vertices around each circle and the edges around each vertex, and so on. All these operations can be reduced to test of the type (1)-(7), as is readily verified.

Returning to the issue of running the generic ranking algorithm, let $p(d)$ be the characteristic polynomial of a

[1] Since some point coordinates may be algebraic functions of d, the "characteristic polynomial" is not necessarily a polynomial, but a more general algebraic function of d of some relatively simple form; it can be handled by similar means.

comparison, and let $z_1 < \cdots < z_u$ be the real roots of $p(d)$.Since each comparison is a test of the sign of $p(d)$, the result of the comparisons does not change in the open interval (z_{i-1}, z_i). Our goal is to find the interval (z_{k-1}, z_k) containing $d^\star$, or determine that $d^\star$ coincides with one of these roots. In the latter case we can stop immediately, for we have found $d^\star$. In the former case we can determine the sign of p at $d^\star$, and thus resolve the corresponding comparison without yet knowing the exact value of $d^\star$.

As described in the introduction, in order to resolve many comparisons at the same time, we run the parallel algorithm given in Section 3. Since we are only going to simulate the parallel algorithm, we need not restart it even if it requires more than $2Cn^{4/3}$ processors. This guarantees that the algorithm will have $O(\log n)$ parallel stages; recall that the number of processors used is always $O(n^2)$.

Now we describe the overall algorithm in more detail. Consider the comparisons performed at the j^{th} stage of the randomized parallel algorithm for the ranking problem. Let $p_1, \ldots, p_s$ be the characteristic polynomials of these comparisons and let $z_1 < \cdots < z_{t-1}$, for $t = O(n^2)$, be the distinct roots of these polynomials. For the sake of convenience let us add two extra points $z_0 = -\infty$ and $z_t = +\infty$ to the above set of points. Our goal is to find β such that either $d_k = z_\beta$ or $d_k \in (z_\beta, z_{\beta+1})$. We proceed as follows: Compute $\mathcal{N}^-(z_{\lfloor t/2 \rfloor})$, $\mathcal{N}^+(z_{\lfloor t/2 \rfloor})$, using the sequential algorithm of Section 3. If $\mathcal{N}^-(z_{\lfloor t/2 \rfloor}) < k$ and $\mathcal{N}^+(z_{\lfloor t/2 \rfloor}) \geq k$, then $z_{\lfloor t/2 \rfloor} = d_k$ and the algorithm can stop right away. If $\mathcal{N}^-(z_{\lfloor t/2 \rfloor}) \geq k$ then we know that $d_k < z_{\lfloor t/2 \rfloor}$, and if $\mathcal{N}^+(z_{\lfloor t/2 \rfloor}) < k$ then we know that $d_k > z_{\lfloor t/2 \rfloor}$. In either of these latter cases we can continue the binary search within the appropriate half of the sequence of roots. After $O(\log n)$ such steps we will thus have found a β_j such that either $d_k = z_{\beta_j}$ or $d_k \in (z_{\beta_j}, z_{\beta_j+1})$. Since no p_i has a root in this open interval I_j, we can easily determine the sign of each polynomial p_i at $d^\star$, and resolve all comparisons of the j^{th} parallel stage. If $z_{\beta_j} = d_k$, we can stop, as described above, otherwise we go to the next stage.

Note that when we resolve comparisons for the $(j+1)^{st}$ stage, we can throw away all the roots of the characteristic polynomials of $(j+1)^{st}$ stage that do not lie in the interval $(z_{\beta_j}, z_{\beta_j+1})$, because we already know that $d_k \in I_j$. For the remaining roots we repeat the same process as in the j^{th} stage. Thus we always ensure that $I_{j+1} \subseteq I_j$. In practice this observation is likely to result in considerable savings in later stages of the parallel algorithm, but we do not know how to estimate these savings in the worst case.

It is easily checked that when we finally stop, we must have found the exact value of d_k—that is, the interval containing it must have shrunk to a point. Indeed, if an open interval I did remain, then for any point $d \in I$ the ranking algorithm would have computed the same pair of numbers $(\mathcal{N}^-, \mathcal{N}^+)$, with $\mathcal{N}^- < k$ and $\mathcal{N}^* \geq k$, which is clearly impossible for more than one value of d.

Next, we analyze the running time of the above algorithm. The overall procedure performs two types of operations: (i) resolving comparisons, and (ii) doing auxiliary computations. Since the expected number of processors is $O(n^{4/3})$, and there are $O(\log n)$ stages, the expected total time required by auxiliary computations including sorting the roots of the characteristic polynomials is $O(n^{4/3} \log n)$.As to the time spent in resolving comparisons, observe that the number of processors, and therefore the number of comparisons to be resolved at each stage is always $O(n^2)$, which implies that the sequential algorithm is invoked $O(\log n)$ times per stage. Since the expected running time of the sequential algorithm is $O(n^{4/3} \log^{2/3} n)$, the total expected time spent in resolving comparisons over all $O(\log n)$ stages is $O(n^{4/3} \log^{8/3} n)$. This establishes our main result.

Theorem 4.1 *Given a set S of n points in the plane, we can compute the k^{th} smallest distance determined by pairs of points in S by a randomized algorithm whose expected running time is $O(n^{4/3} \log^{8/3} n)$.*

If, in the above procedure, we use the deterministic algorithms of Section 2, instead of the randomized ones, we obtain

Theorem 4.2 *Given a set S of n points in the plane, we can deterministically compute the k^{th} smallest distance determined by pairs of points in S in time $O(n^{3/2} \log^{5/2} n)$.*

Since the algorithms described in Section 2 and 3 can be extended to the general L_p metric, we have

Corollary 4.3 *Given a set S of n points in the plane, we can compute the k^{th} smallest distance, in any L_p-metric, determined by pairs of points in S, by a randomized algorithm in expected time $O(n^{4/3} \log^3 n)$, or by a deterministic algorithm in time $O(n^{3/2} \log^3 n)$.*

Remark: The algorithms described in this section operate under the assumption that computing the roots of a polynomial of a fixed degree can be accomplished in $O(1)$ time. Both randomized procedures described above are expected to make $O(n^{4/3} \log n)$ root-finding queries, while their deterministic counterparts ask $O(n^{3/2} \log n)$ questions of this type.

5 Approximating the Median Distance

In this section we describe a simple construction that yields, in $O(n \log n)$ time, an approximation of the median distance of S.

For simplicity, we assume that no three points are collinear and no four points co-circular. By the "ham-sandwich" theorem [Ed], there exist two lines ℓ_1, ℓ_2 such that each of the four quadrants Q_1, Q_2, Q_3, Q_4 determined by them contains at most $\lfloor n/4 \rfloor$ points of S. Let O be the intersection point of ℓ_1 and ℓ_2. Let D be a closed disk centered at O and containing $\lfloor n/6 \rfloor$ points of S. The boundary circle C of D and lines ℓ_1, ℓ_2 split S into eight subsets—four lying inside C and four outside (see figure 5).

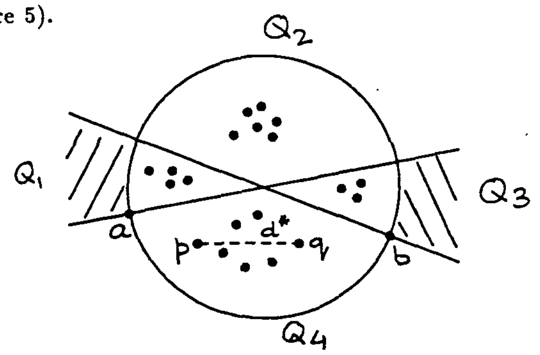

Figure 5: Approximate median

Compute the diameter of each of the inner subsets and let d^* be the largest of the four. Observe that there is a large number of pairs of points in S separated by no more than d^*, as no two points lying in the same inner subset of S are further than d^* apart. Let n_i (resp. m_i) be the number of points in the inner (resp. outer) subset of $S \cap Q_i$. Since $\sum n_i = \lfloor n/6 \rfloor$, the number of distances not exceeding d^* is at least

$$\binom{n_1}{2} + \binom{n_2}{2} + \binom{n_3}{2} + \binom{n_4}{2} \geq 4\binom{n/24}{2} > c_1\binom{n}{2}$$

for any $c_1 < \frac{1}{144}$ and sufficiently large n.

Next, we bound the rank of d^* from above. Without loss of generality assume that the angle spanned by Q_1 is not bigger than the angle spanned by Q_2. Let a, b be the endpoints of the arc $Q_2 \cap C$. It is easily seen that $d^* \leq |ab|$. On the other hand the pairs of points lying in the opposite outer subsets of Q_1, Q_3 are guaranteed to be farther than $|ab|$ apart. Therefore, the number of pairs of points at a distance greater than d^* is at least $m_1 \cdot m_3 \geq n^2/48$, as $m_1, m_3 \leq \frac{n}{4}$ and $m_1 + m_3 \geq \frac{n}{2} - \frac{n}{6} = \frac{n}{3}$.

Finally, we bound the time complexity of our construction. The 4-partition of S is determined in $O(n)$ time using Megiddo's algorithm [Me2], and D can be easily computed using a linear selection algorithm [BFPRT]. Computing the diameter of a set of at most n points takes $O(n \log n)$ time [Ed]. Thus we have shown:

Theorem 5.1 *In time $O(n \log n)$, it is possible to identify a pair of points $p, q \in S$ with the property that there exist absolute constants c_1 and c_2 such that $0 < c_1 < \frac{1}{2} < c_2 < 1$ and the rank of $|pq|$ is between $c_1\binom{n}{2}$ and $c_2\binom{n}{2}$.*

6 Conclusion

In this paper we presented efficient randomized as well as deterministic algorithms for computing the k^{th} smallest distance in a given set of n points in the plane. We also gave efficient algorithms for the ranking problem, which we regard as interesting in its own right. Our techniques can be generalized in several ways. We mention some of them here—see the full version for details.

1. Our techniques can be extended to yield subquadratic algorithms for the ranking problem in $\mathbb{R}^3$.
2. If it is not required to identify the pair of points defining the k^{th} smallest distance d_k, one can use an approximation algorithm to compute the value of d_k with the accuracy of b bits by splitting the interval $[d_1, d_{\binom{n}{2}}]$ repeatedly into two equal parts and identifying the interval containing d_k. This requires b applications of the ranking algorithm and thus takes $O(bn^{3/2}\log^{1/2} n)$ deterministic time or $O(bn^{4/3}\log^{2/3} n)$ randomized expected time.
3. We can also solve the "bichromatic" version of the problem—find the k^{th} smallest distance between m red points and n blue points.
4. The ranking algorithms can be easily extended to report how many points lie in each disc, or how many discs contain each point.

Finally we conclude with some open problems:

(i) Although we have improved the running time of the previously best known algorithms for computing the k^{th} smallest distance in a given set of n points, we have no reason to believe that our algorithms are optimal or even close to optimal. This raises the usual question of whether the running time of these algorithms can be further improved.

(ii) Our algorithm for computing the k^{th} smallest distance is quite complicated. It would be interesting to construct a simpler algorithm with the same running time.

(iii) Another open question is to compute the k^{th} smallest distance in $\mathbb{R}^3$. One of the subproblems that needs to be tackled in order for our algorithm to generalize to three dimensions is preprocessing an arrangement of congruent spheres in parallel for fast point-location queries.

References

[BO] J.L. Bentley and T. Ottmann, Algorithms for reporting and counting geometric intersections, *IEEE Trans. on Computers* C–28 (1979), 643–647.

[BFPRT] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan, Time bounds for selection, *J. Computer and Systems Sciences* 7 (1973), 448–461.

[Ch] B. Chazelle, New techniques for computing order statistics in Euclidean space, *Proceedings 1st Annual Symposium on Computational Geometry*, 1985, pp. 125–134.

[CL] B. Chazelle and D.T. Lee, On a circle placement problem, *Computing* 1 (1986), 1–16.

[CEGSW] K. Clarkson, H. Edelsbrunner, L. Guibas, M. Sharir and E. Welzl, Combinatorial complexity bounds for arrangements of curves and surfaces, *Discrete and Computational Geometry* 5 (1990), 99–160.

[CS] K. Clarkson and P. Shor, Applications of random sampling in computational geometry II, *Discrete and Computational Geometry* 4 (1989), 387–421.

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

[CV] R. Cole and U. Vishkin, Deterministic coin tossing with applications to optimal parallel list ranking, *Information and Control* 70 (1986), 32–53.

[Ed] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[EGPPSS] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, and M. Sharir, Arrangements of curves in the plane: Combinatorics, topology and algorithms, *Proceedings* 15th *International Colloquium on Automata, Languages and Programming*, 1988, pp. 214–229.

[EGS] H. Edelsbrunner, L.J. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM J. Computing* 15 (1986), 317–340.

[FL] M. Fischer and L. Ladner, Parallel prefix computation, *J. ACM* 27 (1980), 831-838.

[HS] S. Hart and M. Sharir, Nonlinearity of Davenport-Schinzel sequences and of generalized path-compression schemes, *Combinatorica* 6 (1986), 151–177.

[Ma] J. Matoušek, Cutting hyperplane arrangements, *Proceedings* 6th *Annual Symposium on Computational Geometry*, 1990.

[Me1] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM* 30 (1983), 852–865.

[Me2] N. Megiddo, Partition with two lines in the plane, *J. Algorithms* 6 (1985), 430-433.

[PS] F. Preparata and M. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, Heidelberg, 1985.

[Sa] J. Salowe, Selection problems in computational geometry, PhD Thesis, Department of Computer Science, Rutgers University, New Brunswick, New Jersey, 1987.

[SPP] A. Schönhage, M. Paterson, and N. Pipinger, Finding the median, *J. Computer and Systems Sciences* 13 (1976), 184-199.

[SV] Y. Shiloach and U. Vishkin, Finding the maximum, merging and sorting in a parallel computation model, *J. Algorithms* 2 (1981), 88-102.

[TVs] R. Tarjan and U. Vishkin, An efficient parallel biconnectivity algorithm, *SIAM J. Computing* 14, (1985), 862-874.

[TVt] R. Tamassia and J.S. Vitter, Optimal parallel algorithms for transitive closure and point location in planar structures, Dept. Comp. Sci. Tech. Rept. CS-89-20, Brown University, March 1989.

[Va] L. Valiant, Parallelism in comparison problems, *SIAM J. Computing* 4 (1975), 348-345.

[Vi] U. Vishkin, Synchronous parallel computation—a survey, Dept. Comp. Sci. Tech. Rept. 71, New York University, New York, 1983.

[Ya] A. Yao, On constructing minimum spanning tree in k-dimensional space and related problems, *SIAM J. Computing* 11 (1982), 721-736.