On enclosing k points by a circle

Jiří Matoušek
Department of Applied Mathematics
Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic

Abstract

We consider the problem of finding, for a given n-point set P in the plane and an integer $k \leq n$, a smallest circle enclosing at least k points of P. We present randomized algorithms with $O(nk)$ space and $O(n \log n + nk)$ expected running time, resp. $O(n)$ space and $O(n \log n + nk \log k)$ time. This improves on previous results by logarithmic factors, and our algorithms are simpler and easier to implement.

1 Introduction

One can solve the problem stated in the abstract by computing the so-called kth order Voronoi diagram for the set P. Methods are known for doing this in time close to $O(nk)$, which is the complexity of the diagram. The best published methods need time of the order $n^{1+\varepsilon}k$, $\varepsilon > 0$ an arbitrarily small positive constant, they are fairly complicated and not very promising for a practically efficient implementation. Recently Efrat *et al.* [ESZ93] gave an alternative algorithm, with time and space complexity $O(nk \log^2 n)$ and $O(nk)$, respectively. They also describe a variant with $O(nk \log^2 n \log(n/k))$ running time requiring only $O(n \log n)$ space. After a preliminary version of this note was prepared, the author learned (by the kindness of A. Efrat) about two other recent works on the considered problem. Eppstein and Erickson [EE93] solve the problem in $O(n \log n + kn \log k)$ time with $O(n \log n + kn + k^2 \log k)$ space, and Datta *et al.* [DLSS93] give an algorithm with the same running time and space improved to $O(n+k^2 \log k)$. The algorithm given below is still slightly better and hopefully simpler. On the other hand, the latter authors also consider various generalizations and also a dynamic version of the problem, none of which will be treated in the current note.

The Efrat *et al.*'s algorithm is based on a technique called *parametric search* due to Megiddo [Meg83]. Parametric search, while very elegant and powerful in theory, usually yields algorithms which, at least on the first sight, look difficult to implement and not very efficient in practice (however, we are not aware of any experience with an actual implementation of a nontrivial instance of parametric search). Several authors worked on replacing parametric search by other techniques at specific instances, see [DMN92], [Mat91], [CEGS92], [KS93]. In particular, in [Mat91] the author suggested a general approach, replacing parametric search by a suitable randomized search.

The present paper can be viewed as a contribution to developing this methodology, demonstrated on the above described specific problem. Starting with a similar approach as Efrat *et al.*, we analyze the problem in some more detail and obtain a randomized algorithm without parametric search. We have the following results:

Theorem 1 *A smallest circle enclosing at least k points among n given points can be found in expected time $O(n \log n + kn)$, using $O(nk)$ space. With $O(n)$ space, expected time $O(n \log n + nk \log k)$ can be achieved.*

This improves the asymptotical complexity of the above mentioned algorithms somewhat, and our algorithms are quite simple, in particular the one with linear space. Let us remark that some ingredients of our technique can most likely be also applied to improve and/or simplify some of the more general results of Datta et al. [DLSS93].

With the current approach, it seems unlikely to get a running time substantially below $O(nk)$ for the considered problem. Determining the actual complexity remains a challenging open problem.

Let us begin the exposition of the algorithm by few definitions. For a point p in the plane, let $B(p,r)$ denote the closed disk with center p and radius r, and let $C(p,r)$ be the bounding circle of $B(p,r)$. Let P, the given point set, be fixed. For a point $x \in \mathbb{R}^2$, let $\mathrm{depth}(r,x)$ be the number of points $p \in P$ for which the disk $B(p,r)$ contains x, and let $\mathrm{depth}(r)$ be the maximum of $\mathrm{depth}(r,x)$ over all points $x \in \mathbb{R}^2$. Let us say that points p, p' are r-incident if their distance is at most $2r$, that is, if the circles $C(p,r)$ and $C(p',r)$ intersect.

The overall structure of our algorithm is similar to that of Efrat *et al.*. They start by observing that the problem is equivalent to computing

$$r^* = \min\{r \geq 0;\ \mathrm{depth}(r) \geq k\}\,,$$

plus a point x with $\mathrm{depth}(r^*,x) \geq k$. Both algorithms work in two phases. The first phase finds a $r_0 \geq r^*$ which approximates r^* well enough in a suitable sense, and the second phase then determines r^* exactly.

2 The first phase

In the first phase we compute a number r_0 with

$$k \leq \mathrm{depth}(r_0) = O(k)\,, \tag{1}$$

using $O(n)$ space and $O(n \log n)$ deterministic time[1]. We can moreover obtain the following additional information: For every point $p \in P$ we get a subset $I(p) \subseteq P$, $|I(p)| = O(k)$, containing all the points r_0-incident with p. The sets $I(p)$ are implicitly represented in $O(n)$ space in such a way that the elements of each $I(p)$ can be listed in $O(k)$ time.

The main observation which allows us to do the computations quickly is the following:

Lemma 2 *For any $A \geq 1$, $r > 0$ we have*

$$\mathrm{depth}(r) \leq \mathrm{depth}(Ar) \leq (A+1)^2 \mathrm{depth}(r)\,.$$

Proof: The first inequality is clear. For the second inequality, write $k = \mathrm{depth}(Ar)$ and consider a point x with $\mathrm{depth}(Ar,x) = k$. The disk $B(x,Ar)$ contains k points of P, and thus $B(x,(A+1)r)$ contains k disks of radius r centered at points of P. The sum of areas of

[1] In Efrat *et al.*'s algorithm the requirement on r_0 is somewhat weaker, they compute r_0 such that $\mathrm{depth}(r_0) \geq k$ and the arrangement of all circles $C(p,r_0)$ for all $p \in P$ has $O(nk)$ vertices. On the other hand, the first phase of the algorithm of [DLSS93] is similar to ours.

these disks is at least $k\pi r^2$, while the area of $B(x,(A+1)r)$ is $\pi(A+1)^2r^2$. Hence there is a point covered by at least $k/(A+1)^2$ of the smaller disks. □

We will approximate depth(r) by the maximum depth of a point from a suitable grid. For a point $p=(x,y)$ and a number $r>0$, define a point $\mathrm{grid}_r(p)=(\lfloor x/r\rfloor r, \lfloor y/r\rfloor r)$, and write $\mathrm{grid}_r(P)$ for $\{\mathrm{grid}_r(p);\ p\in P\}$. Let gdepth($r$) denote the maximum number of points of P mapped to a single point by the mapping $\mathrm{grid}_{r/\sqrt{2}}$.

Lemma 3

$$\mathrm{gdepth}(r)\le \mathrm{depth}(r)=O(\mathrm{gdepth}(r))\,.$$

Proof: The first bound is immediate as any point p has distance at most r from $\mathrm{grid}_{r/\sqrt{2}}(p)$. For the second estimate, the proof is very similar to that of Lemma 2: a r-disk around a point x of depth k contains k points of P, and these are mapped by $\mathrm{grid}_{r/\sqrt{2}}$ into $B(x,2r)$. On the other hand, this disk contains only $O(1)$ distinct points of $\mathrm{grid}_{r/\sqrt{2}}(P)$, by a volume argument, so there must be a grid point to which $\Omega(k)$ points of P are mapped. □

We begin the first phase of the algorithm by sorting the points of P by the x-coordinates, and also by the y-coordinates. Then, for a given r, we can easily compute the set $\mathrm{grid}_r(P)$ and, for every its point, a list of points of P mapped to it (so in particular we can determine gdepth(r)). We can find this information in $O(n)$ time as follows: Using the sorted order in x-coordinate, we can also sort the points of $\mathrm{grid}_r(P)$. Then we label the points of P by integers between 1 and at most n in such a way that points p,p' get the same label iff the x-coordinates of $\mathrm{grid}_r(p)$ and $\mathrm{grid}_r(p')$ coincide. We compute a similar labeling for the y-coordinate. We sort the points of P lexicographically by the pairs of labels; points mapped to the same point by grid_r appear consecutively in the sorted list.

By Lemma 3, it suffices to find r_0 with $k\le \mathrm{gdepth}(r_0)=O(k)$. When searching for such an r_0, we can restrict ourselves to the set of all pairwise distances among points of P, and we can search with accuracy up to a factor of 2, say. The set of all distances is still unnecessarily complicated. We let Δ denote the set of all differences of consecutive x-coordinates of points of P and all differences of consecutive y-coordinates. It is easily seen that each pairwise distance in P is in range $\delta/2, 2n\delta$ for some $\delta\in\Delta$. We thus first locate the position of r_0 among the elements of Δ; suppose that we find r_0 is between δ_1 and δ_2. Then we search among numbers of the form $\delta_1 2^m$ and $\delta_2 2^m$, with $-1\le m\le \log_2 n+1$. We can produce and sort Δ in $O(n\log n)$ time, and then perform the binary search in $O(\log n)$ steps, using the above described $O(n)$ time testing of gdepth(r) to direct the search. Thus, r_0 can be found in $O(n\log n)$ time[2].

It remains to compute the auxiliary information. To this end, we assign to each point of P the 4 grid points surrounding it instead of a single grid point as before. We let $\mathrm{Grid}_r(p)=\{\mathrm{grid}_r(p)+(i,j);\ i,j=0,1\}$. For every point occurring in some $\mathrm{Grid}_{2r_0}(p)$ $(p\in P)$, we compute a list of all points of P having it in its 4-tuple. Using Lemmas 2,3, we get that each such list has $O(k)$ points only. It is easy to check that any two r_0-incident points occur simultaneously in at least one of these lists. Thus, in order to find the set $I(p)$ as described in the beginning of this section, it suffices to merge the lists of the 4 points in $\mathrm{Grid}_{2r_0}(p)$.

[2] In practice we could perform a binary search for r_0 by simple interval halving, in the interval between the smallest and the largest distance of points of P. In the infinite precision mode of computation (Real RAM) we cannot afford to do this, since the ratio of the maximum and minimum distances can be arbitrarily large.

3 The second phase

For a point $p \in P$, let r_p^* be the minimum r such that the circle $C(p, r)$ contains some point x with $\mathrm{depth}(r, x) \geq k$. Clearly we have $r^* = \min\{r_p^*;\ p \in P\}$. First we deal with the following subproblem:

> Given r_0 satisfying (1), a point $p \in P$ and a set $I(p)$ as described in the beginning of the previous section, find r_p^* or conclude that $r_p^* > r_0$.

We observe that for given r, we can test if $r < r_p^*$, in $O(k \log k)$ time, as follows. We compute the intersections of all circles of the form $C(q, r)$ with $q \in I(p)$ with the circle $C(p, r)$, we sort these intersections along $C(p, r)$, we determine the depth of one arbitrarily chosen point $x \in C(p, r)$ and then we walk around $C(p, r)$, maintaining the current depth incrementally. In this way, we find the deepest point on $C(p, r)$, and this tells us whether $r < r_p^*$. With some extra care, we can also distinguish the cases $r = r_p^*$ and $r > r_p^*$.

We will search for r_p^* in the interval $(0, r_0]$. We could apply parametric search to the previously described procedure for comparing given r with the (unknown) r_p^*, similarly as Efrat *et al.* do, but we can use a much simpler algorithm. Obviously we only have to look at the values of r where the system of arcs defined on $C(p, r)$ by the other circles changes combinatorially; call such combinatorial changes the *events*. There are two types of events. Events of one type (1-events) are caused by one other circle $C(q, r)$, and they occur for the r where $C(p, r)$ and $C(q, r)$ touch. Events of the other type (2-events) occur when three circles $C(p, r), C(q_1, r)$ and $C(q_2, r)$ have a common triple intersection.

The total number of 1-events is $O(k)$, and we can list them explicitly and determine the position of r_p^* among them in $O(k \log^2 k)$ time by binary search. The total number of 2-events is $O(k^2)$, and we can search r_p^* among them in $O(k^2 + k \log^2 k) = O(k^2)$ time (we need not sort them, we repeatedly select median from the remaining events during the binary search). This simple method will be good enough in one variant of our algorithm.

We now describe a better method, which finds r_p^* in $O(k \log^2 k)$ expected time. First we note that if we have narrowed the search to an interval (r_1, r_2) containing at most k events, then we can determine r_p^* in $O(k \log k)$ additional time, by a sweep technique. To get a clearer geometric picture, we note that as r decreases from r_2 to r_1, the intersections of $C(p, r)$ with some $C(q, r)$ move along the axis of the segment pq, thus their positions fill two segments (or one segment if $C(p, r_1) \cap C(q, r_1) = \emptyset$). All 2-events thus correspond to vertices of an arrangement of segments with endpoints on $C(p, r_1)$ and $C(p, r_2)$, and this arrangement can be swept by a circle shrinking from $C(p, r_2)$ inwards. During this sweep, we can also maintain the maximum depth of a point on the sweeping circle, all that within $O(k \log k)$ time.

It remains to explain how to decrease the number of events below k. We would like to perform a binary search, but for this, we need to select an event within the current interval of radii, which is roughly in the middle of all events in that interval. Clearly it suffices to select a *random* event in the current interval as a dividing point, the search will still require $O(\log k)$ expected steps.

We can use a very simple strategy to pick a random event. We repeatedly pick a random pair of segments, and check if they define an event within the current interval of radii. If yes, we use that as a random event, and if not, we repeat the experiment. There are $O(k^2)$ pairs of segments, so if N of them define an event, the expected number of trials for picking one is $O(k^2/N)$, and after $O(k^2 \log k/N)$ trials we find an event with high probability. One small difficulty is that we do not know how many events are there in the current interval, but if we have executed more than $Ck\log k$ trials for a current interval, we have a high enough probability that it contains no more than k events, and we can proceed with the sweep stage. The total expected time for finding r_p^* is $O(k\log^2 k)$.

Let us remark that one could devise more sophisticated methods for selecting a random event in the given interval of radii, similar to a method for selecting a random inversion of a permutation, see [Mat91]. But the previous solution is sufficiently fast and quite simple.

Now we are ready to discuss the second phase of the overall algorithm for finding r^*. With the above discussed $O(k\log^2 k)$ method for computing r_p^*, we could simply process each p separately, obtaining an algorithm with $O(n)$ space and $O(n\log n+nk\log^2 k)$ expected running time.

We describe two better algorithms. First we observe that given some number $r \leq r_0$ and a subset $Q \subseteq P$, we can in $O(|Q|k\log k)$ time discard all $p \in Q$ with $r_p^* > r$ (for each $p \in Q$, use the above described $O(k\log k)$ test to decide whether $r \leq r_p^*$). We may search for r^* as follows: We start with $Q = P$ and $r = r_0$, and we discard from Q all points p with $r_p^* > r_0$. Then we repeatedly pick a random point p from the current Q, compute r_p^* and discard all q with $r_q^* \geq r_p^*$ (unless $r_p^* = r^*$, which we can determine during the tests). In each step $|Q|$ decreases by a constant factor on the average, and after expected $O(\log n)$ steps we find r^*. Here we can even use the simpler $O(k^2)$ method for computing the r_p^*'s. Altogether we need $O(n)$ space and $O(n\log n + k^2\log n + nk\log k) = O(n\log n + nk\log k)$ expected time, thus establishing one part of Theorem 1.

With space $O(nk)$ at our disposal, we can proceed as follows. First, we pick a random sample R of $m = n/\log^2 k$ points of P, we compute r_p^* for each $p \in R$ and determine r_1, the smallest of them. This needs expected time $O(mk\log^2 k) = O(nk)$. Then we discard all points $p \in P$ with $r_p^* \geq r_1$; the expected number of remaining points is $O(\log^2 k)$. For each of the remaining points, we compute r_p^* and we select r^* as the minimum, which needs $O(\log^2 k.k\log^2 k) = o(nk)$ expected time. The bottleneck here is discarding most of the points of P using r_1. To do it quickly with $O(nk)$ space, we construct the arrangement of the circles $C(p, r_1)$, $p \in P$ and determine the depth of each vertex by a graph search algorithm. The arrangement has $O(nk)$ vertices and it can be constructed in $O(n\log n + nk)$ expected time by a randomized incremental algorithm, see e.g. [Mul88], [CS89]. Thus, the overall expected time is $O(n\log n + nk)$, and the proof of Theorem 1 is finished. $\square$

Acknowledgment. I am grateful to Alon Efrat for comments regarding the presentation and for bringing the new references to my attention.

References

[CEGS92] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Diameter, width, closest line pair, and parametric searching. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 120–129, 1992.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[DLSS93] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for k-point clustering problems. In *Proc. 3rd Workshop Algorithms Data Struct.*, Lecture Notes in Computer Science, 1993.

[DMN92] M. B. Dillencourt, D. M. Mount, and N. S. Netanyahu. A randomized algorithm for slope selection. *Internat. J. Comput. Geom. Appl.*, 2:1–27, 1992.

[EE93] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 64–73, 1993.

[ESZ93] A. Efrat, M. Sharir, and A. Ziv. Computing the smallest k-enclosing circle and related problems. *Workshop on Algorithms and Data Structures (WADS) 1993*, to appear.

[KS93] M. Katz and M. Sharir. An expander-based approach to geometric optimization. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages ??–??, 1993.

[Mat91] J. Matoušek. Randomized optimal algorithm for slope selection. *Inform. Process. Lett.*, 39:183–187, 1991.

[Meg83] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30:852–865, 1983.

[Mul88] K. Mulmuley. A fast planar partition algorithm, I. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 580–589, 1988.