CHAPTER 1

The Power of Grids – Closest Pair and Smallest Enclosing Disk

In this chapter, we are going to discuss two basic geometric algorithms. The first one computes the closest pair among a set of n points in linear time. This is a beautiful and surprising result that exposes the computational power of using grids for geometric computation. Next, we discuss a simple algorithm for approximating the smallest enclosing ball that contains k points of the input. This at first looks like a bizarre problem but turns out to be a key ingredient to our later discussion.

1.1. Preliminaries

For a real positive number α and a point $\mathsf{p} = (x, y)$ in $\mathbb{R}^2$, define $\mathsf{G}_\alpha(\mathsf{p})$ to be the grid point $(\lfloor x/\alpha \rfloor \alpha, \lfloor y/\alpha \rfloor \alpha)$. We call α the ***width*** or ***sidelength*** of the ***grid*** G_α. Observe that G_α partitions the plane into square regions, which we call grid ***cells***. Formally, for any $i, j \in \mathbb{Z}$, the intersection of the halfplanes $x \geq \alpha i$, $x < \alpha(i+1)$, $y \geq \alpha j$, and $y < \alpha(j+1)$ is said to be a grid ***cell***. Further we define a ***grid cluster*** as a block of 3×3 contiguous grid cells.

Note that every grid cell $\square$ of G_α has a unique ID; indeed, let $\mathsf{p} = (x, y)$ be any point in $\square$, and consider the pair of integer numbers $\mathrm{id}_\square = \mathrm{id}(\mathsf{p}) = (\lfloor x/\alpha \rfloor, \lfloor y/\alpha \rfloor)$. Clearly, only points inside $\square$ are going to be mapped to $\mathrm{id}_\square$. We can use this to store a set P of points inside a grid efficiently. Indeed, given a point p, compute its $\mathrm{id}(\mathsf{p})$. We associate with each unique id a data-structure (e.g., a linked list) that stores all the points of P falling into this grid cell (of course, we do not maintain such data-structures for grid cells which are empty). So, once we have computed $\mathrm{id}(\mathsf{p})$, we fetch the data-structure associated with this cell by using hashing. Namely, we store pointers to all those data-structures in a hash table, where each such data-structure is indexed by its unique id. Since the ids are integer numbers, we can do the hashing in constant time.

Assumption 1.1. Throughout the discourse, we assume that every hashing operation takes (worst case) constant time. This is quite a reasonable assumption when true randomness is available (using for example perfect hashing [**CLRS01**]).

Assumption 1.2. Our computation model is the unit cost RAM model, where every operation on real numbers takes constant time, including log and $\lfloor \cdot \rfloor$ operations. We will (mostly) ignore numerical issues and assume exact computation.

Definition 1.3. For a point set P and a parameter α, the ***partition*** of P into subsets by the grid G_α is denoted by $\mathsf{G}_\alpha(\mathsf{P})$. More formally, two points $\mathsf{p}, \mathsf{q} \in \mathsf{P}$ belong to the same set in the partition $\mathsf{G}_\alpha(\mathsf{P})$ if both points are being mapped to the same grid point or equivalently belong to the same grid cell; that is, $\mathrm{id}(\mathsf{p}) = \mathrm{id}(\mathsf{q})$.

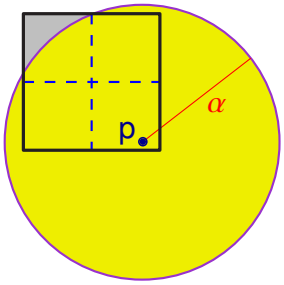
1.2. Closest pair

We are interested in solving the following problem:

PROBLEM 1.4. Given a set P of n points in the plane, find the pair of points closest to each other. Formally, return the pair of points realizing $\mathcal{CP}(\mathsf{P}) = \min_{\mathsf{p}\neq\mathsf{q},\ \mathsf{p},\mathsf{q}\in\mathsf{P}} \|\mathsf{p}-\mathsf{q}\|$.

The following is an easy standard ***packing argument*** that underlines, under various disguises, many algorithms in computational geometry.

LEMMA 1.5. *Let P be a set of points contained inside a square $\square$, such that the sidelength of $\square$ is $\alpha = \mathcal{CP}(\mathsf{P})$. Then $|\mathsf{P}| \leq 4$.*



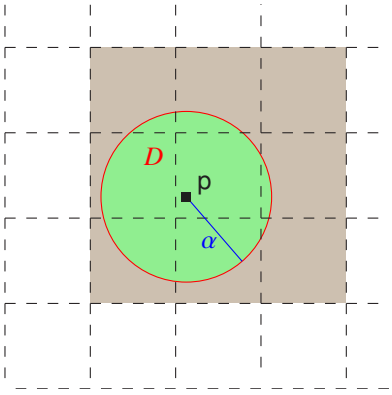
PROOF. Partition $\square$ into four equal squares $\square_1, \ldots, \square_4$, and observe that each of these squares has diameter $\sqrt{2}\alpha/2 < \alpha$, and as such each can contain at most one point of P; that is, the disk of radius α centered at a point $\mathsf{p} \in \mathsf{P}$ completely covers the subsquare containing it; see the figure on the right.

Note that the set P can have four points if it is the four corners of $\square$. ■

LEMMA 1.6. *Given a set P of n points in the plane and a distance α, one can verify in linear time whether $\mathcal{CP}(\mathsf{P}) < \alpha$, $\mathcal{CP}(\mathsf{P}) = \alpha$, or $\mathcal{CP}(\mathsf{P}) > \alpha$.*

PROOF. Indeed, store the points of P in the grid G_α. For every non-empty grid cell, we maintain a linked list of the points inside it. Thus, adding a new point p takes constant time. Specifically, compute $\mathrm{id}(\mathsf{p})$, check if $\mathrm{id}(\mathsf{p})$ already appears in the hash table, if not, create a new linked list for the cell with this ID number, and store p in it. If a linked list already exists for $\mathrm{id}(\mathsf{p})$, just add p to it. This takes $O(n)$ time overall.

Now, if any grid cell in $\mathsf{G}_\alpha(\mathsf{P})$ contains more than, say, 4 points of P, then it must be that the $\mathcal{CP}(\mathsf{P}) < \alpha$, by Lemma 1.5.



Thus, when we insert a point p, we can fetch all the points of P that were already inserted in the cell of p and the 8 adjacent cells (i.e., all the points stored in the cluster of p); that is, these are the cells of the grid G_α that intersects the disk $D = \mathrm{disk}(\mathsf{p}, \alpha)$ centered at p with radius α; see the figure on the right. If there is a point closer to p than α that was already inserted, then it must be stored in one of these 9 cells (since it must be inside D). Now, each one of those cells must contain at most 4 points of P by Lemma 1.5 (otherwise, we would already have stopped since the $\mathcal{CP}(\cdot)$ of the inserted points is smaller than α). Let S be the set of all those points, and observe that $|S| \leq 9 \cdot 4 = O(1)$. Thus, we can compute, by brute force, the closest point to p in S. This takes $O(1)$ time. If $\mathbf{d}(\mathsf{p}, S) < \alpha$, we stop; otherwise, we continue to the next point.

Overall, this takes at most linear time.

As for correctness, observe that the algorithm returns '$\mathcal{CP}(\mathsf{P}) < \alpha$' only after finding a pair of points of P with distance smaller than α. So, assume that p and q are the pair of points of P realizing the closest pair and that $\|\mathsf{p}-\mathsf{q}\| = \mathcal{CP}(\mathsf{P}) < \alpha$. Clearly, when the later point (say p) is being inserted, the set S would contain q, and as such the algorithm would

stop and return '$\mathcal{CP}(\mathsf{P}) < \alpha$'. Similar argumentation works for the case that $\mathcal{CP}(\mathsf{P}) = \alpha$. Thus if the algorithm returns '$\mathcal{CP}(\mathsf{P}) > \alpha$', it must be that $\mathcal{CP}(\mathsf{P})$ is not smaller than α or equal to it. Namely, it must be larger. Thus, the algorithm output is correct. ■

REMARK 1.7. Assume that $\mathcal{CP}(\mathsf{P} \setminus \{\mathsf{p}\}) \geq \alpha$, but $\mathcal{CP}(\mathsf{P}) < \alpha$. Furthermore, assume that we use Lemma 1.6 on P, where $\mathsf{p} \in \mathsf{P}$ is the last point to be inserted. When p is being inserted, not only do we discover that $\mathcal{CP}(\mathsf{P}) < \alpha$, but in fact, by checking the distance of p to all the points stored in its cluster, we can compute the closest point to p in $\mathsf{P} \setminus \{\mathsf{p}\}$ and denote this point by q. Clearly, pq is the closest pair in P, and this last insertion still takes only constant time.

Slow algorithm. Lemma 1.6 provides a natural way of computing $\mathcal{CP}(\mathsf{P})$. Indeed, permute the points of P in an arbitrary fashion, and let $\mathsf{P} = \langle \mathsf{p}_1, \ldots, \mathsf{p}_n \rangle$. Next, let $\alpha_{i-1} = \mathcal{CP}(\{\mathsf{p}_1, \ldots, \mathsf{p}_{i-1}\})$. We can check if $\alpha_i < \alpha_{i-1}$ by using the algorithm of Lemma 1.6 on P_i and α_{i-1}. In fact, if $\alpha_i < \alpha_{i-1}$, the algorithm of Lemma 1.6 would return '$\mathcal{CP}(\mathsf{P}_i) < \alpha_{i-1}$' and the two points of P_i realizing α_i.

So, consider the "good" case, where $\alpha_i = \alpha_{i-1}$; that is, the length of the shortest pair does not change when p_i is being inserted. In this case, we do not need to rebuild the data-structure of Lemma 1.6 to store $\mathsf{P}_i = \langle \mathsf{p}_1, \ldots, \mathsf{p}_i \rangle$. We can just reuse the data-structure from the previous iteration that was used by P_{i-1} by inserting p_i into it. Thus, inserting a single point takes constant time, as long as the closest pair does not change.

Things become problematic when $\alpha_i < \alpha_{i-1}$, because then we need to rebuild the grid data-structure and reinsert all the points of $\mathsf{P}_i = \langle \mathsf{p}_1, \ldots, \mathsf{p}_i \rangle$ into the new grid $\mathsf{G}_{\alpha_i}(\mathsf{P}_i)$. This takes $O(i)$ time.

In the end of this process, we output the number α_n, together with the two points of P that realize the closest pair.

OBSERVATION 1.8. *If the closest pair distance, in the sequence $\alpha_1, \ldots, \alpha_n$, changes only t times, then the running time of our algorithm would be $O(nt + n)$. Naturally, t might be $\Omega(n)$, so this algorithm might take quadratic time in the worst case.*

Linear time algorithm. Surprisingly[1], we can speed up the above algorithm to have linear running time by spicing it up using randomization.

We pick a random permutation of the points of P and let $\langle \mathsf{p}_1, \ldots, \mathsf{p}_n \rangle$ be this permutation. Let $\alpha_2 = \|\mathsf{p}_1 - \mathsf{p}_2\|$, and start inserting the points into the data-structure of Lemma 1.6. We will keep the invariant that α_i would be the closest pair distance in the set P_i, for $i = 2, \ldots, n$.

In the ith iteration, if $\alpha_i = \alpha_{i-1}$, then this insertion takes constant time. If $\alpha_i < \alpha_{i-1}$, then we know what is the new closest pair distance α_i (see Remark 1.7), rebuild the grid, and reinsert the i points of P_i from scratch into the grid G_{α_i}. This rebuilding of $\mathsf{G}_{\alpha_i}(\mathsf{P}_i)$ takes $O(i)$ time.

Finally, the algorithm returns the number α_n and the two points of P_n realizing it, as the closest pair in P.

LEMMA 1.9. *Let t be the number of different values in the sequence $\alpha_2, \alpha_3, \ldots, \alpha_n$. Then $\mathbf{E}[t] = O(\log n)$. As such, in expectation, the above algorithm rebuilds the grid $O(\log n)$ times.*

[1] Surprise in the eyes of the beholder. The reader might not be surprised at all and might be mildly annoyed by the whole affair. In this case, the reader should read any occurrence of "surprisingly" in the text as being "mildly annoying".

PROOF. For $i \geq 3$, let X_i be an indicator variable that is one if and only if $\alpha_i < \alpha_{i-1}$. Observe that $\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1]$ (as X_i is an indicator variable) and $t = \sum_{i=3}^{n} X_i$.

To bound $\mathbf{Pr}[X_i = 1] = \mathbf{Pr}[\alpha_i < \alpha_{i-1}]$, we (conceptually) fix the points of P_i and randomly permute them. A point $\mathsf{q} \in \mathsf{P}_i$ is ***critical*** if $\mathcal{CP}(\mathsf{P}_i \setminus \{\mathsf{q}\}) > \mathcal{CP}(\mathsf{P}_i)$. If there are no critical points, then $\alpha_{i-1} = \alpha_i$ and then $\mathbf{Pr}[X_i = 1] = 0$ (this happens, for example, if there are two pairs of points realizing the closest distance in P_i). If there is one critical point, then $\mathbf{Pr}[X_i = 1] = 1/i$, as this is the probability that this critical point would be the last point in the random permutation of P_i.

Assume there are two critical points and let p, q be this unique pair of points of P_i realizing $\mathcal{CP}(\mathsf{P}_i)$. The quantity α_i is smaller than α_{i-1} only if either p or q is p_i. The probability for that is $2/i$ (i.e., the probability in a random permutation of i objects that one of two marked objects would be the last element in the permutation).

Observe that there cannot be more than two critical points. Indeed, if p and q are two points that realize the closest distance, then if there is a third critical point s, then $\mathcal{CP}(\mathsf{P}_i \setminus \{\mathsf{s}\}) = \|\mathsf{p} - \mathsf{q}\|$, and hence the point s is not critical.

Thus, $\mathbf{Pr}[X_i = 1] = \mathbf{Pr}[\alpha_i < \alpha_{i-1}] \leq 2/i$, and by linearity of expectations, we have that $\mathbf{E}[t] = \mathbf{E}\left[\sum_{i=3}^{n} X_i\right] = \sum_{i=3}^{n} \mathbf{E}[X_i] \leq \sum_{i=3}^{n} 2/i = O(\log n)$. ■

Lemma 1.9 implies that, in expectation, the algorithm rebuilds the grid $O(\log n)$ times. By Observation 1.8, the running time of this algorithm, in expectation, is $O(n \log n)$. However, we can do better than that. Intuitively, rebuilding the grid in early iterations of the algorithm is cheap, and only late rebuilds (when $i = \Omega(n)$) are expensive, but the number of such expensive rebuilds is small (in fact, in expectation it is a constant).

THEOREM 1.10. *For set P of n points in the plane, one can compute the closest pair of P in expected linear time.*

PROOF. The algorithm is described above. As above, let X_i be the indicator variable which is 1 if $\alpha_i \neq \alpha_{i-1}$, and 0 otherwise. Clearly, the running time is proportional to

$$R = 1 + \sum_{i=3}^{n} (1 + X_i \cdot i) .$$

Thus, the expected running time is proportional to

$$\begin{aligned} \mathbf{E}\Big[R\Big] = \mathbf{E}\left[1 + \sum_{i=3}^{n} (1 + X_i \cdot i)\right] &\leq n + \sum_{i=3}^{n} \mathbf{E}\Big[X_i\Big] \cdot i \leq n + \sum_{i=3}^{n} i \cdot \mathbf{Pr}\Big[X_i = 1\Big] \\ &\leq n + \sum_{i=3}^{n} i \cdot \frac{2}{i} \leq 3n, \end{aligned}$$

by linearity of expectation and since $\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1]$ and since $\mathbf{Pr}[X_i = 1] \leq 2/i$ (as shown in the proof of Lemma 1.9). Thus, the expected running time of the algorithm is $O(\mathbf{E}[R]) = O(n)$. ■

Theorem 1.10 is a surprising result, since it implies that uniqueness (i.e., deciding if n real numbers are all distinct) can be solved in linear time. Indeed, compute the distance of the closest pair of the given numbers (think about the numbers as points on the x-axis). If this distance is zero, then clearly they are not all unique.

However, there is a lower bound of $\Omega(n \log n)$ on the running time to solve uniqueness, using the comparison model. This "reality dysfunction" can be easily explained once one realizes that the computation model of Theorem 1.10 is considerably stronger, using hashing, randomization, and the floor function.

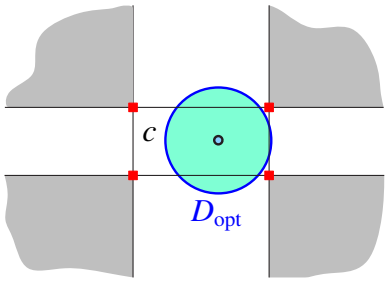


FIGURE 1.1. If the disk $D_{\text{opt}}(\mathsf{P}, k)$ does not contain any vertex of the cell c, then it does not cover any shaded area. As such, it can contain at most $k/2$ points, since the vertical and horizontal strips containing c each has at most $k/4$ points of P inside it.

1.3. A slow 2-approximation algorithm for the k-enclosing disk

For a disk D, we denote by radius(D) the ***radius*** of D. Let $D_{\text{opt}}(\mathsf{P}, k)$ be a disk of minimum radius which contains k points of P, and let $r_{\text{opt}}(\mathsf{P}, k)$ denote the radius of $D_{\text{opt}}(\mathsf{P}, k)$. For $k = 2$, this is equivalent to computing the closest pair of points of P. As such, the problem of computing $D_{\text{opt}}(\mathsf{P}, k)$ is a generalization of the problem studied in the previous section.

Here, we study the easier problem of approximating $r_{\text{opt}}(\mathsf{P}, k)$.

OBSERVATION 1.11. *Given a set P of n points in the plane, a point q, and a parameter k, one can compute the k closest points in P to q in $O(n)$ time. To do so, compute for each point of P its distance to q. Next, using a selection algorithm, compute the k smallest numbers among these n distances. These k numbers corresponds to the desired k points. The running time is $O(n)$ as selection can be done in linear time.*

The algorithm algDCoverSlow(P, k). Let P be a set of n points in the plane. Compute a set of $m = O(n/k)$ horizontal lines $h_1, \ldots, h_m$ such that between two consecutive horizontal lines, there are at most $k/4$ points of P in the strip they define. To this end, consider the points of P sorted in increasing y ordering, and create a horizontal line through the points of rank $i(k/4)$ in this order, for $i = 1, \ldots, \lfloor n/(k/4) \rfloor$. This can be done in $O(n \log(n/k))$ time using deterministic median selection together with recursion.[②] Similarly, compute a set of vertical lines $v_1, \ldots, v_m$, such that between two consecutive lines, there are at most $k/4$ points of P. (Note that all these lines pass through points of P.)

Consider the (non-uniform) grid G induced by the lines $h_1, \ldots, h_m$ and $v_1, \ldots, v_m$. Let X be the set of all the intersection points of these lines; that is, X is the set of vertices of G. For every point $\mathsf{p} \in X$, compute (in linear time) the smallest disk centered at p that contains k points of P, and return the smallest disk computed.

LEMMA 1.12. *Given a set P of n points in the plane and parameter k, the algorithm algDCoverSlow(P, k) computes, in $O(n(n/k)^2)$ deterministic time, a circle D that contains k points of P and* radius(D) $\leq 2r_{\text{opt}}(\mathsf{P}, k)$*, where $r_{\text{opt}}(\mathsf{P}, k)$ is the radius of the smallest disk in the plane containing k points of P.*

[②] Indeed, compute the median in the y-order of the points of P, split P into two sets, and recurse on each set, till the number of points in a subproblem is of size $\leq k/4$. We have $T(n) = O(n) + 2T(n/2)$, and the recursion stops for subproblems of size $\leq k/4$. Thus, the recursion tree has depth $O(\log(n/k))$, which implies running time $O(n \log(n/k))$.

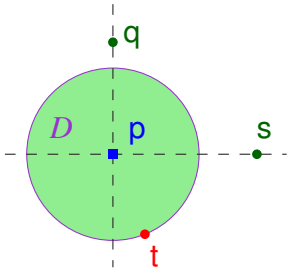
PROOF. Since $|X| = O((n/k)^2)$ and for each such point finding the smallest disk containing k points takes $O(n)$ time, the running time bound follows.

As for correctness, we claim that $D_{\text{opt}}(\mathsf{P}, k)$ contains at least one point of X. Indeed, consider the center u of $D_{\text{opt}}(\mathsf{P}, k)$, and let c be the cell of G that contains u. Clearly, if $D_{\text{opt}}(\mathsf{P}, k)$ does not cover any of the four vertices of c, then it can cover only points in the vertical and horizontal strips of G that contain c. See Figure 1.1. However, each such strip contains at most $k/4$ points, and there are two such strips. It follows that $D_{\text{opt}}(\mathsf{P}, k)$ contains at most $k/2$ points of P, a contradiction. Thus, $D_{\text{opt}}(\mathsf{P}, k)$ must contain a point of X. Clearly, for a point $\mathsf{q} \in X \cap D_{\text{opt}}(\mathsf{P}, k)$, this yields the required 2-approximation. Indeed, the disk of radius $2r_{\text{opt}}(\mathsf{P}, k)$ centered at q contains at least k points of P since it also covers $D_{\text{opt}}(\mathsf{P}, k)$. ∎

COROLLARY 1.13. *Given a set P of n points and a parameter $k = \Omega(n)$, one can compute in linear time a circle D that contains k points of P and* $\text{radius}(D) \leq 2r_{\text{opt}}(\mathsf{P}, k)$.

REMARK 1.14. If **algDCoverSlow**(P, k) is applied to a point set P of size smaller than k, then the algorithm picks an arbitrary point p of P and outputs the minimum radius disk centered at p containing P. This takes $O(|\mathsf{P}|)$ time.

REMARK 1.15. One can sometime encode the output of a geometric algorithm in terms of the input objects that define it. This might be useful for handling numerical issues, where this might prevent numerical errors. In our case, we will use this to argue about the expected running time of an algorithm that uses **algDCoverSlow** in a black-box fashion.



So, consider the disk D output by the algorithm **algDCoverSlow**(P, k). It is centered at point p of radius r. The point p is the intersection of two grid lines. Each of these grid lines, by construction, passes through two input points $\mathsf{q}, \mathsf{s} \in \mathsf{P}$. Similarly, the radius of D is the distance of the intersection point p to a point $\mathsf{t} \in \mathsf{P}$. Namely, the disk D can be uniquely specified by a triple of points $(\mathsf{q}, \mathsf{s}, \mathsf{t})$ of P, where the first (resp. second) point q (resp. s) specifies the vertical (resp. horizontal) line of the grid that passes through this point (thus the two points specify the center of the disk), and the third point specifies the points on the boundary of the disk; see the figure on the right.

Now, think about running **algDCoverSlow** on all possible subsets $\mathsf{Q} \subseteq \mathsf{P}$. The above argument implies that although there the number of different inputs considered is exponential, the algorithm always outputs one of n^3 possible outputs, where $n = |\mathsf{P}|$.

Lemma 1.12 can be easily extended to higher dimensions. We get the following result.

LEMMA 1.16. *Given a set P of n points in $\mathbb{R}^{\mathsf{d}}$ and parameter k, one can compute, in $O\left(n(n/k)^{\mathsf{d}}\right)$ deterministic time, a ball $\mathbf{b}$ that contains k points of P and its radius* $\text{radius}(\mathbf{b}) \leq 2r_{\text{opt}}(\mathsf{P}, k)$, *where $r_{\text{opt}}(\mathsf{P}, k)$ is the radius of the smallest ball in $\mathbb{R}^{\mathsf{d}}$ containing k points of P.*

1.4. A linear time 2-approximation for the k-enclosing disk

In the following, we present a linear time algorithm for approximating the minimum enclosing disk. While interesting in their own right, the results here are not used later and can be skipped on first, second, or third (or any other) reading.

As in the previous sections, we construct a grid which partitions the points into small ($O(k)$ sized) groups. The key idea behind speeding up the grid computation is to construct

```
algGrow(P_i, α_{i-1}, k)
  Output: α_i
begin
  G_{i-1} ← G_{α_{i-1}}(P_i)
  for every grid cluster c ∈ G_{i-1}
                with |c ∩ P_i| ≥ k do
      P_c ← c ∩ P_i
      α_c ← algDCoverSlow(P_c, k)
      //  r_opt(P_c, k) ≤ α_c ≤ 2 r_opt(P_c, k)

  return minimum α_c computed.
end
```

```
algDCover(P, k)
  Output: 2-approx. to r_opt(P, k)
begin
  Compute a gradation of P:
        {P_1, ..., P_m}
  α_1 ← algDCoverSlow(P_1, k)
  for i ← 2 to m do
      α_i ← algGrow(P_i, α_{i-1}, k)
  return α_m
end
```

FIGURE 1.2. The linear time 2-approximation algorithm **algDCover** for the smallest disk containing k points of P. Here, **algDCoverSlow** denotes the "slow" 2-approximation algorithm of Lemma 1.16.

the appropriate grid over several rounds. Specifically, we start with a small set of points as a seed and construct a suitable grid for this subset. Next, we incrementally insert the remaining points, while adjusting the grid width appropriately at each step.

1.4.1. The algorithm.

1.4.1.1. *Preliminaries.* We remind the reader that **algDCoverSlow**(P, k) is the "slow" algorithm of Lemma 1.16.

DEFINITION 1.17. Given a set P of n points, a ***k-gradation*** $(\mathsf{P}_1, \ldots, \mathsf{P}_m)$ of P is a sequence of subsets of P, such that (i) $\mathsf{P}_m = \mathsf{P}$, (ii) P_i is formed by picking each point of P_{i+1} with probability $1/2$, (iii) $|\mathsf{P}_1| \leq k$, and (iv) $|\mathsf{P}_2| > k$.

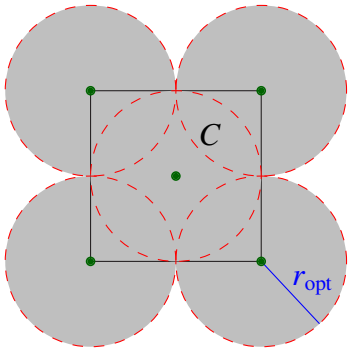
We remind the reader that a ***grid cluster*** is a block of 3×3 contiguous grid cells.

DEFINITION 1.18. Let $\mathtt{gd}_\alpha(\mathsf{P})$ denote the maximum number of points of P mapped to a single grid cell by the partition $\mathsf{G}_\alpha = \mathsf{G}_\alpha(\mathsf{P})$; see Definition 1.3.

Also, let $\mathtt{depth}(\mathsf{P}, \alpha)$ be the maximum number of points of P that a circle of radius α can contain.

LEMMA 1.19. *For any point set P and $\alpha > 0$, we have that if $\alpha \leq 2 r_{\mathrm{opt}}(\mathsf{P}, k)$, then any cell of the grid G_α contains at most $5k$ points; that is, $\mathtt{gd}_\alpha(\mathsf{P}) \leq 5k$.*

PROOF. Let C be the grid cell of G_α that realizes $\mathtt{gd}_\alpha(\mathsf{P})$. Place 4 points at the corners of C and one point in the center of C. Placing at each of those points a disk of radius $r_{\mathrm{opt}}(\mathsf{P}, k)$ completely covers C, as can be easily verified (since the sidelength of C is at most $2 r_{\mathrm{opt}}(\mathsf{P}, k)$). Thus, $|\mathsf{P} \cap C| = \mathtt{gd}_\alpha(\mathsf{P}) \leq 5\,\mathtt{depth}(\mathsf{P}, r_{\mathrm{opt}}(\mathsf{P}, k)) = 5k$; see the figure on the right. ■



1.4.1.2. *The algorithm.* We first compute a gradation $\{\mathsf{P}_1, \ldots, \mathsf{P}_m\}$ of P. We use **algDCoverSlow** on P_1 to get the desired approximation for P_1. Next, we iteratively refine this approximation moving from one set in the gradation to the next one. Specifically, motivated by the above, we will maintain the following invariants at the end of the ith round: (i) there is a distance α_i such that $\mathsf{gd}_{\alpha_i}(\mathsf{P}_i) \le 5k$, (ii) there is a grid cluster in G_{α_i} containing k or more points of P_i, and (iii) $r_{\mathrm{opt}}(\mathsf{P}_i, k) \le \alpha_i$.

At the ith round, the algorithm constructs a grid G_{i-1} for points in P_i using α_{i-1} as the grid width. We use the slow algorithm of Lemma 1.16 (i.e., **algDCoverSlow**), on each of the non-empty grid clusters, to compute α_i.

In the end of this process, the number α_m is the desired approximation.

The new algorithm **algDCover** is depicted in Figure 1.2.

Intuition. During its execution, the algorithm never maps too many points to a single grid cell being used. Indeed, we know that there is no grid cell of G_{i-1} containing more than $5k$ points of P_{i-1}. We expect every cell of G_{i-1} to contain at most $10k$ points of P_i, since $\mathsf{P}_{i-1} \subseteq \mathsf{P}_i$ was formed by choosing each point of P_i with probability $1/2$. (This of course is too good to be true, but something slightly weaker does hold.)

Note that the algorithm of Lemma 1.16 runs in linear time if the number of points in the set is $O(k)$. In the optimistic scenario, every cluster of $\mathsf{G}_{i-1}(\mathsf{P}_i)$ has $O(k)$ points, and computing the clustering radius for this point set takes $O(k)$ time. Now, every point participates in a constant number of clusters, and it follows that this fixing stage takes (overall) linear time in the size of P_i.

1.4.2. Analysis – correctness.

LEMMA 1.20. *For $i = 1, \ldots, m$, we have $r_{\mathrm{opt}}(\mathsf{P}_i, k) \le \alpha_i \le 2r_{\mathrm{opt}}(\mathsf{P}_i, k)$, and the heaviest cell in $\mathsf{G}_{\alpha_i}(\mathsf{P}_i)$ contains at most $5k$ points of P_i.*

PROOF. Consider the optimal disk D_i that realizes $r_{\mathrm{opt}}(\mathsf{P}_i, k)$. Observe that there is a cluster c of $\mathsf{G}_{\alpha_{i-1}}$ that contains D_i, as $\alpha_{i-1} \ge \alpha_i$. Thus, when **algGrow** handles the cluster c, we have $D_i \cap \mathsf{P}_i \subseteq c$. The first part of the lemma then follows from the correctness of the algorithm of Lemma 1.16.

The second part follows by Lemma 1.19. ∎

Lemma 1.20 implies immediately the correctness of the algorithm by applying it for $i = m$.

1.4.3. Running time analysis.

LEMMA 1.21. *Given a set P, a gradation of P can be computed in expected linear time.*

PROOF. Observe that the sampling time is $O\left(\sum_{i=1}^{m} |\mathsf{P}_i|\right)$, where m is the length of the sequence. Also observe that $\mathbf{E}[|\mathsf{P}_m|] = |\mathsf{P}_m| = n$ and

$$\mathbf{E}\left[|\mathsf{P}_i|\right] = \mathbf{E}\left[\mathbf{E}\left[|\mathsf{P}_i| \;\middle|\; |\mathsf{P}_{i+1}|\right]\right] = \mathbf{E}\left[\frac{|\mathsf{P}_{i+1}|}{2}\right] = \frac{1}{2}\,\mathbf{E}\left[|\mathsf{P}_{i+1}|\right].$$

Now by induction, we get

$$\mathbf{E}\left[|\mathsf{P}_{m-i}|\right] = \frac{n}{2^i}.$$

Thus, the running time is $O\left(\mathbf{E}\left[\sum_{i=1}^{m} |\mathsf{P}_i|\right]\right) = O\left(\sum_{i=1}^{m} n/2^i\right) = O(n)$. ∎

Since $|\mathsf{P}_1| \le k$, the call $\alpha_1 \leftarrow$ **algDCoverSlow**(P_1, k) in **algDCover** takes $O(k)$ time, by Remark 1.14.

1.4.3.1. *Excess and why it is low.* Now we proceed to upper-bound the number of cells of $\mathsf{G}_{\alpha_{i-1}}$ that contains "too many" points of P_i. Since each point of P_{i-1} was chosen from P_i with probability $1/2$, we can express this bound as a sum of independent random variables, and we can bound this using tail-bounds.

Definition 1.22. For a point set P and parameters k and α, the ***excess*** of $\mathsf{G}_\alpha(\mathsf{P})$ is

$$\mathcal{E}(\mathsf{P},\alpha) = \sum_{c \in \mathrm{Cells}(\mathsf{G}_\alpha)} \left\lfloor \frac{|c \cap \mathsf{P}|}{50k} \right\rfloor \leq \frac{|\mathsf{P}|}{50k},$$

where $\mathrm{Cells}(\mathsf{G}_\alpha)$ is the set of cells of the grid G_α.

The quantity $100k \cdot \mathcal{E}(\mathsf{P},\alpha)$ is an upper bound on the number of points of P in heavy cells of $\mathsf{G}_\alpha(\mathsf{P})$, where a cell of $\mathsf{G}_\alpha(\mathsf{P})$ is ***heavy*** if it contains at least $50k$ points.

For a point set P of n points, the radius α returned by the algorithm of Lemma 1.16 is the distance between a vertex q of the non-uniform grid (i.e., a point of the set X) and a point of P. Remark 1.15 implies that such a number α is defined by a triple of points of P (i.e., they specify the three points that α is computed from). As such, α can be one of at most $O(n^3)$ different values. This implies that throughout the execution of **algDCover** the only grids that would be considered would be one of (at most) n^3 possible grids. In particular, let $\mathfrak{G} = \mathfrak{G}(\mathsf{P})$ be this set of possible grids.

Lemma 1.23. *For any $t > 0$, let $\gamma = \lceil 3 \ln n \rceil / 8k$ and $\beta = t + \gamma$. The probability that $\mathsf{G}_{\alpha_{i-1}}(\mathsf{P}_i)$ has excess $\mathcal{E}(\mathsf{P}_i, \alpha_{i-1}) \geq \beta$ is at most* $\exp(-8kt)$.

Proof. Let $\mathfrak{G} = \mathfrak{G}(\mathsf{P})$ (see above), and fix a grid $\mathsf{G} \in \mathfrak{G}$ with excess $\mathcal{E}(\mathsf{P}_i, \varkappa(\mathsf{G}))$ at least β, where $\varkappa(\mathsf{G})$ is the sidelength of a cell of G.

Let $U = \left\{ \{\mathsf{P}_i \cap c\} \;\middle|\; c \in \mathsf{G}, |\mathsf{P}_i \cap c| \geq 50k \right\}$ be the sets of points stored in the heavy cells of G. Furthermore, let $\mathcal{V} = \bigcup_{X \in U} \Pi(X, 50k)$, where $\Pi(X, \nu)$ denotes an arbitrary partition of the set X into disjoint subsets such that each one of them contains ν points, except maybe the last subset that might contain between ν and $2\nu - 1$ points.

This partitions the points inside each heavy cell into groups of size at least $50k$. Now, since each such group lies inside a single cell of G, for G to be the grid computed for P_{i-1}, it must be that every such group "promoted" at most $5k$ points from P_i to P_{i-1}, by Lemma 1.20.

Now, it is clear that $|\mathcal{V}| = \mathcal{E}(\mathsf{P}_i, \varkappa(\mathsf{G}))$, and for any $S \in \mathcal{V}$, we have that $\mu = \mathbf{E}[|S \cap \mathsf{P}_{i-1}|] \geq 25k$. Indeed, we promote each point of $S \subseteq \mathsf{P}_i$, independently, with probability $1/2$, to be in P_{i-1} and $|S| \geq 50k$. As such, by the Chernoff inequality (Theorem 27.18_{p341}), for $\delta = 4/5$, we have

$$\begin{aligned}\mathbf{Pr}\Big[|S \cap \mathsf{P}_{i-1}| \leq 5k\Big] &\leq \mathbf{Pr}\Big[|S \cap \mathsf{P}_{i-1}| \leq (1-\delta)\mu\Big] < \exp\left(-\mu\frac{\delta^2}{2}\right) \leq \exp\left(-\frac{25k(4/5)^2}{2}\right) \\ &= \exp(-8k).\end{aligned}$$

Furthermore, since $\mathsf{G} = \mathsf{G}_{\alpha_{i-1}}$, this imply that each cell of G contains at most $5k$ points of P_{i-1}, by Lemma 1.20. Thus we have

$$\begin{aligned}\mathbf{Pr}\Big[\mathsf{G}_{\alpha_{i-1}} = \mathsf{G}\Big] &\leq \prod_{S \in \mathcal{V}} \mathbf{Pr}\Big[|S \cap \mathsf{P}_{i-1}| \leq 5k\Big] \leq \exp(-8k\,|\mathcal{V}|) = \exp(-8k\mathcal{E}(\mathsf{P}_i, \varkappa(\mathsf{G}))) \\ &\leq \exp(-8k\beta)\,.\end{aligned}$$

Since there are at most n^3 different grids in $\mathfrak{G}$, we have

$$\mathbf{Pr}\Big[\mathcal{E}(\mathsf{P}_i, \alpha_{i-1}) \geq \beta\Big] = \mathbf{Pr}\left[\bigcup_{\substack{\mathsf{G}\in\mathfrak{G},\\ \mathcal{E}(\mathsf{P}_i,\varkappa(\mathsf{G}))\geq\beta}} (\mathsf{G} = \mathsf{G}_{\alpha_{i-1}})\right] \leq \sum_{\substack{\mathsf{G}\in\mathfrak{G},\\ \mathcal{E}(\mathsf{P}_i,\varkappa(\mathsf{G}))\geq\beta}} \mathbf{Pr}[\mathsf{G} = \mathsf{G}_{\alpha_{i-1}}]$$

$$\leq n^3 \exp(-8k\beta) \leq \exp\left(3\ln n - 8k\left(t + \frac{3\lceil \ln n\rceil}{8k}\right)\right) \leq \exp(-8kt)\,. \qquad \blacksquare$$

1.4.3.2. *Bounding the running time.*

Lemma 1.24. *In expectation, the running time of the algorithm in the ith iteration is* $O\big(|\mathsf{P}_i| + \gamma^4 k\big)$, *where* $\gamma = \lceil 3\ln n\rceil/8k$.

Proof. Let Y be the random variable which is the excess of $\mathsf{G}_{\alpha_{i-1}}(\mathsf{P}_i)$. In this case, there are at most Y cells which are heavy in $\mathsf{G}_{\alpha_{i-1}}(\mathsf{P}_i)$, and each such cell contains at most $O(Yk)$ points. Thus, invoking the algorithm of **algDCoverSlow** on such a heavy cell takes $O(Yk\cdot((Yk)/k)^2) = O(Y^3k)$ time. Overall, the running time of **algGrow**, in the ith iteration, is $T(Y) = O\big(|\mathsf{P}_i| + Y\cdot Y^3k\big) = O\big(|\mathsf{P}_i| + Y^4k\big)$.

Set $\gamma = \lceil 3\ln n\rceil/8k$, and observe that the expected running time in the ith stage is

$$\begin{aligned} R_i &= O\left(\sum_{t=0}^{n/k} \mathbf{Pr}[Y = t]\left(|\mathsf{P}_i| + Y^4k\right)\right) \\ &= O\left(|\mathsf{P}_i| + \mathbf{Pr}[Y \leq \gamma]\,\gamma^4k + \sum_{t=1}^{n/k} \mathbf{Pr}[Y = t+\gamma](t+\gamma)^4\,k\right) \\ &= O\left(|\mathsf{P}_i| + \gamma^4k + \sum_{t=1}^{n/k} \exp(-8kt)\cdot(t+\gamma)^4\,k\right) = O\left(|\mathsf{P}_i| + \gamma^4k\right), \end{aligned}$$

by Lemma 1.23 and since the summation is bounded by

$$O\left(\sum_{t=1}^{n/k} 16t^4\gamma^4k\exp(-8kt)\right) = O\left(\gamma^4k\sum_{t=1}^{n/k} t^4\exp(-8kt)\right) = O\left(\gamma^4k\right). \qquad \blacksquare$$

Thus, by Lemma 1.24 and by Lemma 1.21, the total expected running time of **algDCover** inside the inner loop is

$$O\left(\sum_{i=1}^{m}\left(|\mathsf{P}_i| + \gamma^4k\right)\right) = O\left(n + \left(\frac{3\ln n}{8k}\right)^4 km\right) = O(n),$$

since $m = O(\log n)$, with high probability, as can be easily verified.

Theorem 1.25. *Given a set* P *of n points in the plane and a parameter k, the algorithm* **algDCover** *computes, in expected linear time, a radius* α, *such that* $r_{\mathrm{opt}}(\mathsf{P},k) \leq \alpha \leq 2r_{\mathrm{opt}}(\mathsf{P},k)$, *where* $r_{\mathrm{opt}}(\mathsf{P},k)$ *is the minimum radius of a disk covering k points of* P.

1.5. Bibliographical notes

Our closest-pair algorithm follows Golin et al. [**GRSS95**]. This is in turn a simplification of a result of Rabin [**Rab76**]. Smid provides a survey of such algorithms [**Smi00**].

The minimum disk approximation algorithm is a simplification of the work of Har-Peled and Mazumdar [**HM05**]. Note that this algorithm can be easily adapted to any point set in constant dimension (with the same running time). Exercise 1.3 is also taken from there.

Computing the exact minimum disk containing k points. By plugging the algorithm of Theorem 1.25 into the exact algorithm of Matoušek [**Mat95a**], one gets an $O(nk)$ time algorithm that computes the minimum disk containing k points. It is conjectured that any exact algorithm for this problem requires $\Omega(nk)$ time.

1.6. Exercises

EXERCISE 1.1 (Packing argument and the curse of dimensionality). One of the reasons why computational problems in geometry become harder as the dimension increases is that packing arguments (see Lemma 1.5) provide bounds that are exponential in the dimension, and even for moderately small dimension (say, $\mathsf{d} = 16$) the bounds they provide are too large to be useful.

As a concrete example, consider a maximum cardinality point set P contained inside the unit length cube C in $\mathbb{R}^{\mathsf{d}}$ (i.e., the ***unit hypercube***), such that $C\mathcal{P}(\mathsf{P}) = 1$.

(A) Prove that

$$2^{\mathsf{d}} \leq |\mathsf{P}| \leq \left(\left\lceil \sqrt{\mathsf{d}} \right\rceil + 1 \right)^{\mathsf{d}}.$$

(B) The above lower bound is conservative. for example, in four dimensions, it is easy to pack 17 points into the hypercube. Show such a configuration.

(C) Using the formula for the volume of the ball (see Section 19.2.1), and Stirling's formula, prove that $\left(\sqrt{\mathsf{d}}/5\right)^{\mathsf{d}} \leq |\mathsf{P}|$, for d sufficiently large.

EXERCISE 1.2 (Compute clustering radius). Let C and P be two given sets of points in the plane, such that $k = |C|$ and $n = |\mathsf{P}|$. Let $r = \max_{\mathsf{p} \in \mathsf{P}} \min_{\overline{\mathsf{c}} \in C} \left\| \overline{\mathsf{c}} - \mathsf{p} \right\|$ be the ***covering radius*** of P by C (i.e., if we place a disk of radius r around each point of C, all those disks cover the points of P).

(A) Give an $O(n + k \log n)$ expected time algorithm that outputs a number α, such that $r \leq \alpha \leq 10r$.

(B) For $\varepsilon > 0$ a prescribed parameter, give an $O(n + k\varepsilon^{-2} \log n)$ expected time algorithm that outputs a number α, such that $\alpha \leq r \leq (1 + \varepsilon)\alpha$.

EXERCISE 1.3 (Randomized k-enclosing disk). Given a set P of n points in the plane and parameter k, present a (simple) randomized algorithm that computes, in expected $O(n(n/k))$ time, a circle D that contains k points of P and $\text{radius}(D) \leq 2r_{\text{opt}}(\mathsf{P}, k)$.

(This is a faster and simpler algorithm than the one presented in Lemma 1.16.)