Geometric Approximation Algorithms
Sariel Har-peled

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, $\mathrm{id}(\mathsf{p}) = \mathrm{id}(\mathsf{q})$.

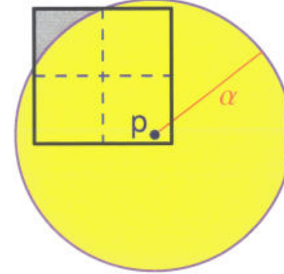
1.2. Closest pair

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* $|\mathsf{P}| \leq 4$.



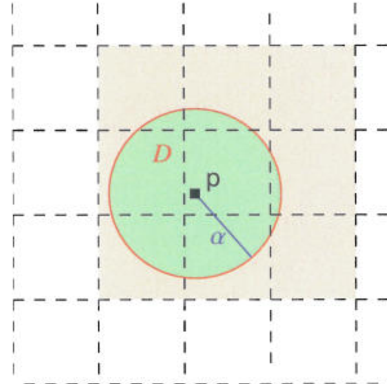
PROOF. Partition $\square$ into four equal squares $\square_1, \ldots, \square_4$, and observe that each of these squares has diameter $\sqrt{2}\alpha/2 < \alpha$, and as such each can contain at most one point of P; that is, the disk of radius α centered at a point $\mathsf{p} \in \mathsf{P}$ completely 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 id(p), check if id(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 id(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

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

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.