Chapter 7

Closest Pair

By Sariel Har-Peled, April 26, 2022[1]

The events of September 8 prompted Foch to draft the later legendary signal: "My centre is giving way, my right is in retreat, situation excellent. I attack." It was probably never sent.

John Keegan, The first world war

7.1. How many times can a minimum change?

Let $a_1, \ldots, a_n$ be a set of n numbers, and let us randomly permute them into the sequence $b_1, \ldots, b_n$. Next, let $c_i = \min_{k=1}^{i} b_i$, and let X be the random variable which is the number of distinct values that appears in the sequence $c_1, \ldots, c_n$. What is the expectation of X?

Lemma 7.1.1. *In expectation, the number of times the minimum of a prefix of n randomly permuted numbers change, is $O(\log n)$. That is $\mathbb{E}[X] = O(\log n)$.*

Proof: Consider the indicator variable X_i, such that $X_i = 1$ if $c_i \neq c_{i-1}$. The probability for that is $\leq 1/i$, since this is the probability that the smallest number of $b_1, \ldots, b_i$ is b_i. (Why is this probability not simply equal to $1/i$?) As such, we have $X = \sum_i X_i$, and $\mathbb{E}[X] = \sum_i \mathbb{E}[X_i] = \sum_{i=1}^{n} \frac{1}{i} = O(\log n)$. ■

7.2. Closest Pair

Assumption 7.2.1. Throughout the discourse, we are going to 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]). We will revisit this issue later in the course.

For a real positive number r and a point $\mathsf{p} = (x, y)$ in $\mathbb{R}^2$, define

$$G_r(\mathsf{p}) := \left(\left\lfloor \frac{x}{r} \right\rfloor r \;,\; \left\lfloor \frac{y}{r} \right\rfloor r \right) \in \mathbb{R}^2.$$

The number r is the ***width*** of the ***grid*** G_r. Observe that G_r partitions the plane into square regions, which are *grid cells*. Formally, for any $i, j \in \mathbb{Z}$, the intersection of the half-planes $x \geq ri$, $x < r(i+1)$, $y \geq rj$ and $y < r(j+1)$ is a ***grid cell***. Further a ***grid cluster*** is a block of 3×3 contiguous grid cells.

For a point set P, and a parameter r, the partition of P into subsets by the grid G_r, is denoted by $G_r(\mathsf{P})$. More formally, two points $\mathsf{p}, \mathsf{u} \in \mathsf{P}$ belong to the same set in the partition $G_r(\mathsf{P})$, if both points are being mapped to the same grid point or equivalently belong to the same grid cell.

Note, that every grid cell C of G_r, has a unique ID; indeed, let $\mathsf{p} = (x, y)$ be any point in C, and consider the pair of integer numbers $\mathrm{id}_C = \mathrm{id}(\mathsf{p}) = (\lfloor x/r \rfloor, \lfloor y/r \rfloor)$. Clearly, only points inside C are going to be mapped to id_C. This is useful, as one can 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 that stores all the

[1]This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

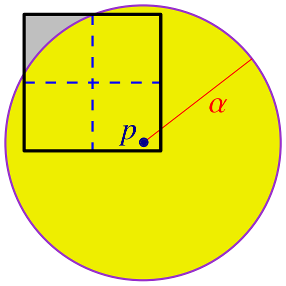
points falling into this grid cell (of course, we do not maintain such data-structures for grid cells which are empty). For our purposes here, the grid-cell data-structure can simply be a linked list of points. So, once we computed $\mathrm{id}(\mathsf{p})$, we fetch the data structure for this cell, 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.

We are interested in solving the following problem.

Problem 7.2.2. 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 $\mathcal{CP}(\mathsf{P}) = \min_{\mathsf{p},\mathsf{u}\in P} \|\mathsf{p} - \mathsf{u}\|$.

We need the following easy packing lemma.

Lemma 7.2.3. *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 7.2.4. *Given a set P of n points in the plane, and a distance r, one can verify in linear time, whether or not $\mathcal{CP}(\mathsf{P}) < r$ or $\mathcal{CP}(\mathsf{P}) \geq r$.*

Proof: Indeed, store the points of P in the grid G_r. 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. Indeed, compute $\mathrm{id}(p)$, check if $\mathrm{id}(p)$ 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 data-structure already exist for $\mathrm{id}(p)$, just add p to it.

This takes $O(n)$ time. Now, if any grid cell in $\mathrm{G}_r(\mathsf{P})$ contains more than four points of P, then, by Lemma 7.2.3, it must be that the $\mathcal{CP}(\mathsf{P}) < r$.

Thus, when inserting a point p, the algorithm fetch all the points of P that were already inserted, for the cell of p, and the 8 adjacent cells. All those cells must contain at most 4 points of P (otherwise, we would already have stopped since the $\mathcal{CP}(\cdot)$ of the inserted points is smaller than r). Let S be the set of all those points, and observe that $|S| \leq 4 \cdot 9 = O(1)$. Thus, we can compute by brute force the closest point to p in S. This takes $O(1)$ time. If $\mathsf{d}(p, S) < r$, we stop and return this distance (together with the two points realizing $\mathsf{d}(p, S)$ as a proof that the distance is too short). Otherwise, we continue to the next point, where $\mathsf{d}(p, S) = \min_{s\in S} \|p - s\|$.

Overall, this takes $O(n)$ time. As for correctness, first observe that if $\mathcal{CP}(\mathsf{P}) > r$ then the algorithm would never make a mistake, since it returns '$\mathcal{CP}(\mathsf{P}) < r$' only after finding a pair of points of P with distance smaller than r. Thus, assume that p, q are the pair of points of P realizing the closest pair, and $\|p - q\| = \mathcal{CP}(\mathsf{P}) < r$. Clearly, when the later of them, say p, is being inserted, the set S would contain q, and as such the algorithm would stop and return "$\mathcal{CP}(\mathsf{P}) < r$". ■

Lemma 7.2.4 hints to a natural way to compute $\mathcal{CP}(\mathsf{P})$. Indeed, permute the points of P, in an arbitrary fashion, and let $P = \langle p_1, \ldots, p_n \rangle$. Next, let $r_i = \mathcal{CP}(\{p_1, \ldots, p_i\})$. We can check if $r_{i+1} < r_i$, by just calling the algorithm for Lemma 7.2.4 on P_{i+1} and r_i. If $r_{i+1} < r_i$, the algorithm of Lemma 7.2.4, would give us back the distance r_{i+1} (with the other point realizing this distance).

So, consider the "good" case where $r_{i+1} = r_i = r_{i-1}$. Namely, the length of the shortest pair does not change. In this case we do not need to rebuild the data structure of Lemma 7.2.4 for each point. We can just reuse it from the previous iteration. Thus, inserting a single point takes constant time as long as the closest pair (distance) does not change.

Things become bad, when $r_i < r_{i-1}$. Because then we need to rebuild the grid, and reinsert all the points of $\mathsf{P}_i = \langle p_1, \ldots, p_i \rangle$ into the new grid $\mathrm{G}_{r_i}(\mathsf{P}_i)$. This takes $O(i)$ time.

So, if the closest pair radius, in the sequence $r_1, \ldots, r_n$, changes only k times, then the running time of the algorithm would be $O(nk)$. But we can do even better!

Theorem 7.2.5. *Let* P *be a set of* n *points in the plane. One can compute the closest pair of points of* P *in expected linear time.*

Proof: Pick a random permutation of the points of P, and let $\langle p_1, \ldots, p_n \rangle$ be this permutation. Let $r_2 = \| p_1 - p_2 \|$, and start inserting the points into the data structure of Lemma 7.2.4. In the ith iteration, if $r_i = r_{i-1}$, then this insertion takes constant time. If $r_i < r_{i-1}$, then we rebuild the grid and reinsert the points. Namely, we recompute $\mathrm{G}_{r_i}(\mathsf{P}_i)$.

To analyze the running time of this algorithm, let X_i be the indicator variable which is 1 if $r_i \neq r_{i-1}$, and 0 otherwise. Clearly, the running time is proportional to

$$R = 1 + \sum_{i=2}^{n} (1 + X_i \cdot i).$$

Thus, the expected running time is

$$\mathbb{E}\big[R\big] = 1 + \mathbb{E}\Big[1 + \sum\nolimits_{i=2}^{n} (1 + X_i \cdot i)\Big] = n + \sum_{i=2}^{n} \big(\mathbb{E}[X_i] \cdot i\big) = n + \sum_{i=2}^{n} i \cdot \mathbb{P}[X_1 = 1],$$

by linearity of expectation and since for an indicator variable X_i, we have that $\mathbb{E}[X_i] = \mathbb{P}[X_i = 1]$.

Thus, we need to bound $\mathbb{P}[X_i = 1] = \mathbb{P}[r_i < r_{i-1}]$. To bound this quantity, fix the points of P_i, and randomly permute them. A point $\mathsf{u} \in \mathsf{P}_i$ is ***critical*** if $\mathcal{CP}(\mathsf{P}_i \setminus \{\mathsf{u}\}) > \mathcal{CP}(\mathsf{P}_i)$.

(A) If there are no critical points, then $r_{i-1} = r_i$ and then $\mathbb{P}[X_i = 1] = 0$.

(B) If there is one critical point, than $\mathbb{P}[X_i = 1] = 1/i$, as this is the probability that this critical point would be the last point in a random permutation of P_i.

(C) If there are two critical points, and let p, u be this unique pair of points of P_i realizing $\mathcal{CP}(\mathsf{P}_i)$. The quantity r_i is smaller than r_{i-1}, if either p or u are p_i. But 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 can not be more than two critical points. Indeed, if p and u are two points that realize the closest distance, than if there is a third critical point v, then $\mathcal{CP}(\mathsf{P}_i \setminus \{\mathsf{v}\}) = \|\mathsf{p} - \mathsf{u}\|$, and v is not critical.

We conclude that

$$\mathbb{E}\big[R\big] = n + \sum_{i=2}^{n} i \cdot \mathbb{P}[X_1 = 1] \leq n + \sum_{i=2}^{n} i \cdot \frac{2}{i} \leq 3n.$$

As such, the expected running time of this algorithm is $O(\mathbb{E}[R]) = O(n)$. ■

Theorem 7.2.5 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. However, there is a lower bound of $\Omega(n \log n)$ on uniqueness, using the comparison tree model. This reality dysfunction, can be easily explained, once one realizes that the model of computation of Theorem 7.2.5 is considerably stronger, using hashing, randomization, and the floor function.

7.3. Bibliographical notes

The closest-pair algorithm follows Golin *et al.* [GRSS95]. This is in turn a simplification of a result of the celebrated result of Rabin [Rab76]. Smid provides a survey of such algorithms [Smi00]. A generalization of the closest pair algorithm was provided by Har-Peled and Raichel [HR15].

Surprisingly, Schönhage [Sch79] showed that assuming that the floor function is allowed, and the standard arithmetic operation can be done in constant time, then every problem in PSPACE can be solved in polynomial time. Since PSPACE includes NPC, this is bad news, as it implies that one can solve NPC problem in polynomial time (finally!). The basic idea is that one can pack huge number of bits into a single number, and the floor function enables one to read a single bit of this number. As such, a real RAM model that allows certain operations, and put no limit on the bit complexity of numbers, and assume that each operation can take constant time, is not a reasonable model of computation (but we already knew that).

References

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press / McGraw-Hill, 2001.

[GRSS95] M. Golin, R. Raman, C. Schwarz, and M. Smid. *Simple randomized algorithms for closest pair problems. Nordic J. Comput.*, 2: 3–27, 1995.

[HR15] S. Har-Peled and B. Raichel. *Net and prune: A linear time algorithm for Euclidean distance problems*. *J. Assoc. Comput. Mach.*, 62(6): 44:1–44:35, 2015.

[Rab76] M. O. Rabin. *Probabilistic algorithms. Algorithms and Complexity: New Directions and Recent Results*. Ed. by J. F. Traub. Orlando, FL, USA: Academic Press, 1976, pp. 21–39.

[Sch79] A. Schönhage. *On the power of random access machines*. *Proc. 6th Int. Colloq. Automata Lang. Prog.* (ICALP), vol. 71. 520–529, 1979.

[Smi00] M. Smid. *Closest-point problems in computational geometry. Handbook of Computational Geometry*. Ed. by J.-R. Sack and J. Urrutia. Amsterdam, The Netherlands: Elsevier, 2000, pp. 877–935.