Chapter 9

The Power of Grids: Closest Pair

By Sariel Har-Peled, February 8, 2023[①]

The Peace of Olivia. How sweat and peaceful it sounds! There the great powers noticed for the first time that the land of the Poles lends itself admirably to partition.

The tin drum, Gunter Grass

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.

9.1. Preliminaries

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

$$\mathsf{G}_\alpha(p) = \Big(\lfloor x/\alpha \rfloor \alpha,\ \lfloor y/\alpha \rfloor \alpha\Big).$$

The number α is the ***width*** or ***sidelength*** of the ***grid*** G_α. Observe that G_α partitions the plane into square regions, which are 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 $y < \alpha(j+1)$ defines 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.

Definition 9.1.1. Let $p = (x, y)$ be any point in a grid cell $\square \in \mathsf{G}_\alpha$, and consider the pair of integer numbers $\mathrm{id}(\square) = \mathrm{id}(p) = (\lfloor x/\alpha \rfloor, \lfloor y/\alpha \rfloor)$. The pair $\mathrm{id}(p)$ is the ***grid ID*** of p.

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 $\mathrm{id}(p)$. 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 $\mathrm{id}(p)$, 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 9.1.2. 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 9.1.3. 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 9.1.4. 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 $p, q \in \mathsf{P}$ 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, $\mathrm{id}(p) = \mathrm{id}(q)$.

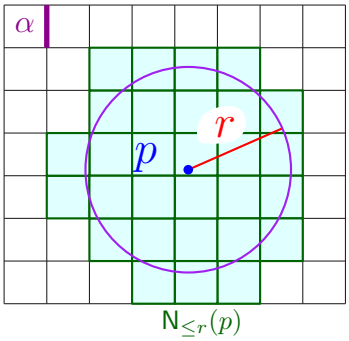

Figure 9.1.1

Definition 9.1.5. For a number $r \geq 0$, and a point p let $\mathsf{N}_{\leq r}(p)$ denote the set of grid cells of G_α in distance $\leq r$ from p, which is the ***neighborhood*** of p. Note, that the neighborhood also includes the grid cell containing p itself, and if $\alpha = \Theta(r)$ then $|\mathsf{N}_{\leq r}(p)| = \Theta\Big((2 + \lceil 2r/\alpha \rceil)^d\Big) = \Theta(1)$.

See Figure 9.1.1 for an example of $\mathsf{N}_{\leq r}(p)$.

9.2. Closest pair in linear time

Definition 9.2.1. For a set P of n points in the plane, let $\mathbf{d}_{\min}(\mathsf{P})$ denote the minimum distance between a pair of points in P. Formally, we have

$$\mathbf{d}_{\min}(\mathsf{P}) = \min_{p \neq q,\ p,q \in \mathsf{P}} \|pq\| .$$

The pair of points realizing this minimum, is the ***closest pair*** in P, and it is denoted by $\mathcal{CP}(\mathsf{P})$.

We are interested in solving the following problem:

Problem 9.2.2. Given a set P of n points in the plane, find the closest pair of points in P.

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

Lemma 9.2.3. *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 $p \in \mathsf{P}$ completely covers the subsquare containing it, see Figure 9.2.1.
Note that the set P can have four points if it is the four corners of $\square$. ■

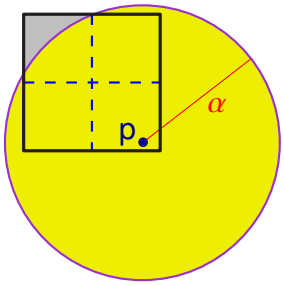

Figure 9.2.1

Lemma 9.2.4. *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 $\mathrm{id}(p)$, 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 linked list already exists for $\mathrm{id}(p)$, 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 9.2.3.

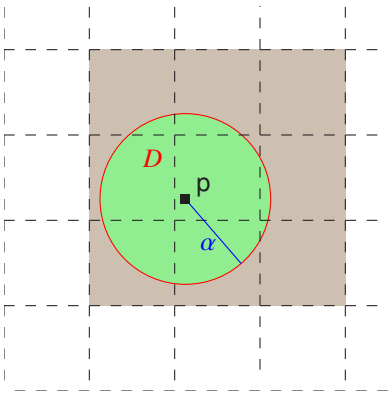

Figure 9.2.2

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 $D = \mathrm{disk}(p, \alpha)$ centered at p with radius α; see Figure 9.2.2. 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 9.2.3 (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 $\mathbf{d}(p, S) < \alpha$, 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 $\|pq\| = \mathcal{CP}(\mathsf{P}) < \alpha$. 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 9.2.5. Assume that $\mathcal{CP}(\mathsf{P} \setminus \{p\}) \geq \alpha$, but $\mathcal{CP}(\mathsf{P}) < \alpha$. Furthermore, assume that we use Lemma 9.2.4 on P, where $p \in \mathsf{P}$ 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 $\mathsf{P} \setminus \{p\}$ 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 9.2.4 provides a natural way of computing $\mathcal{CP}(\mathsf{P})$. Indeed, permute the points of P in an arbitrary fashion, and let $\mathsf{P} = \langle p_1, \ldots, p_n \rangle$. Next, let $\alpha_{i-1} = \mathcal{CP}(\{p_1, \ldots, p_{i-1}\})$. We can check if $\alpha_i < \alpha_{i-1}$ by using the algorithm of Lemma 9.2.4 on P_i and α_{i-1}. In fact, if $\alpha_i < \alpha_{i-1}$, the algorithm of Lemma 9.2.4 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 9.2.4 to

store $\mathsf{P}_i = \langle p_1, \ldots, p_i \rangle$. 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 $\mathsf{P}_i = \langle p_1, \ldots, p_i \rangle$ 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 9.2.6. *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[2], 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 $\langle p_1, \ldots, p_n \rangle$ be this permutation. Let $\alpha_2 = \|p_1 p_2\|$, and start inserting the points into the data-structure of Lemma 9.2.4. 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 9.2.5), 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 9.2.7. *Let t be the number of different values in the sequence $\alpha_2, \alpha_3, \ldots, \alpha_n$. Then $\mathbb{E}[t] = O(\log n)$. As such, in expectation, the above algorithm rebuilds the grid $O(\log n)$ times.*

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 $\mathbb{E}[X_i] = \mathbb{P}[X_i = 1]$ (as X_i is an indicator variable) and $t = \sum_{i=3}^{n} X_i$.

To bound $\mathbb{P}[X_i = 1] = \mathbb{P}[\alpha_i < \alpha_{i-1}]$, we (conceptually) fix the points of P_i and randomly permute them. A point $q \in \mathsf{P}_i$ is ***critical*** if $\mathcal{CP}(\mathsf{P}_i \setminus \{q\}) > \mathcal{CP}(\mathsf{P}_i)$. If there are no critical points, then $\alpha_{i-1} = \alpha_i$ and then $\mathbb{P}[X_i = 1] = 0$ (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 $\mathbb{P}[X_i = 1] = 1/i$, 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 $\mathcal{CP}(\mathsf{P}_i \setminus \{\mathsf{s}\}) = \|pq\|$, and hence the point s is not critical.

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

Lemma 9.2.7 implies that, in expectation, the algorithm rebuilds the grid $O(\log n)$ times. By Observation 9.2.6, the running time of this algorithm, in expectation, is $O(n \log n)$. However, we can

[2] 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".

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 9.2.8. *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

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

by linearity of expectation and since $\mathbb{E}[X_i] = \mathbb{P}[X_i = 1]$ and since $\mathbb{P}[X_i = 1] \leq 2/i$ (as shown in the proof of Lemma 9.2.7). Thus, the expected running time of the algorithm is $O(\mathbb{E}[R]) = O(n)$. ∎

Theorem 9.2.8 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 9.2.8 is considerably stronger, using hashing, randomization, and the floor function.

9.3. An alternative closest pair algorithm

We present an alternative linear time algorithm for the closest pair problem. Let **algCP**(P) denote the new algorithm, where P is the point set under consideration.

Some preliminaries. For a point $p \in P$, let

$$\ell(p) = \mathsf{d}(p, P - p) = \min_{q \in P - p} \|pq\|$$

denote the ***nearest neighbor distance*** of p to its closest point in $P \setminus \{p\}$. For a parameter r, a point $p \in P$ is r***-far*** if $\ell(p) \geq r$.

Fortunately, we can identify all the r-far points quickly.

Lemma 9.3.1. *Let* P *be a set of* n *points in the plane. For a parameter* r*, one can compute the set* $P_{\geq r} = \{\ell(p) \geq r \mid p \in P\}$ *of all* r*-far points in* $O(n)$ *time.*

Proof: Throw the points of P into the grid $\mathsf{G}_{r/2}$. Clearly, a point can be r-far only if it is the only point in its grid cell. Indeed, a grid cell has diameter $\sqrt{2}r/2 < r$, and any two points inside the same grid cell are too close to each other to be r-far. For each such isolated point, we scan all the grid cells in distance at most r from its cell, and compute its nearest neighbor in these cells. If the computed distance is $< r$, then the point is not r-far. Otherwise, we add it to the far point set $P_{\geq r}$.

Observe, that a grid cell has at most $O(1)$ grid cells in distance at most r from it. Furthermore, a grid cell and the point set stored in it get scanned if it is in distance at most r from a grid cell that contains a single point. It follows, that a grid cell (and its point set) get scanned at most $O(1)$ time by the above algorithm, and as such this algorithm has linear running time as stated. ∎

The algorithm. If $|P| = O(1)$, it computes the closest pair using brute force. Otherwise, the algorithm picks a random point $p \in P$. The algorithm computes $r = \ell(p)$ by scanning the points of P explicitly – this takes $O(n)$ time. Next, the algorithm computes $P_{\geq r}$. If $P_{\geq r} = P$, then r is the closest pair distance, and the algorithm returns p and its nearest neighbor as the desired closest-pair. Finally, the algorithm call recursively **algCP**$(P \setminus P_{\geq r})$ and returns the result as the desired answer.

Correctness. If $P_{\geq r} = P$ then clearly r is the closest pair distance, as this distance is the point in P that realizes the minimum of $\ell(p)$. Otherwise, $P_{\geq r}$ does not contain all the points of P. In particular, the closest pair $q, s \in P$, has $\ell(q) < r$ and $\ell(s) < r$. Namely, the closet pair in P and in $P \setminus P_{\geq r}$ is the same. Furthermore, every time the algorithm performs a recursive call, the size of the point set shrinks, as $|P_{\geq r}| \geq 1$ as it contains p (it probably contains way more points, as we see next). Namely, the algorithm terminates and returns the closest pair.

Running time analysis. Consider the multiset of distances

$$L = L(P) = \{\ell(p) \mid p \in P\}.$$

As a reminder, an element $x \in L$ has ***strict rank*** k, if $L_{<x} = \{y \in L \mid y < x\}$ is of size k – that is, there are $k-1$ elements smaller than x in L.

The algorithm randomly pick a point p, which is equivalent to randomly picking a value from the multiset L uniformly. In particular, the algorithm is ***lucky*** if the strict rank of p is at most $n/2$. Clearly, this happens with probability at least half. If this happens, then the algorithm recurses only on the points that are in $P_{<r} = P \setminus P_{\geq r}$ (i.e., the points that corresponds to the values in $L_{<r}$. Namely, the algorithm calls recursively on a point set of half the size. If the algorithm is not lucky, then it is a big fat proven loser, and it calls recursively on a set that might be potentially of size as large as $n-1$. We thus, have the following recurrence on the expected running time:

$$T(n) \leq O(n) + \mathbb{P}[\text{lucky}]T(n/2) + \mathbb{P}[\text{loser}]T(n-1) \leq O(n) + \frac{1}{2}T(n/2) + \frac{1}{2}T(n),$$

since $T(n-1) \leq T(n)$. Rearranging, we get

$$T(n) \leq 2 \cdot O(n) + 2 \cdot \frac{1}{2}T(n/2) \leq O(n) + T(n/2).$$

Clearly, the solution to this recurrence is $O(n)$.

Theorem 9.3.2. *Given a set P of n points in the plane, one can compute the closest pair of points in P in $O(n)$ expected time.*

9.4. 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 alternative closest pair algorithm of Theorem 9.3.2 is from [**hr-nplta-15**]. The later also shows that the algorithm can be modified so that the linear running time holds with high probability.

One can extend the closet pair algorithm to approximate the minimum disk covering k points. See [HS05].

Beware of the floor function.. In the real RAM model (which is not real, that is the point), then one can manipulate arbitrarily large floating point numbers to arbitrary precision in constant time per operation. If you amend this model by allowing it use the floor function, then one can solve PSPACE-hard problems in polynomial time [Sch79]. Arguably, the operations we do in the algorithms presented in this chapter are quite reasonable on a real world computer, but it does point out that the real RAM model falls short in this case.

9.5. Exercises

Exercise 9.5.1 (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 for example Lemma 9.2.3) 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 $\mathcal{CP}(\mathsf{P}) = 1$.

(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, and Stirling's formula, prove that $\left(\sqrt{\mathsf{d}}/5\right)^{\mathsf{d}} \leq |\mathsf{P}|$, for d sufficiently large.

Exercise 9.5.2 (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 $r = \max_{p \in \mathsf{P}} \min_{c \in C} \|cp\|$ 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 $\alpha \leq r \leq (1 + \varepsilon)\alpha$.

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.

[HS05] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing disc. *Algorithmica*, 41(3): 147–157, 2005.

[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.