CS 390 Homework 1

Diya Singh

January 30, 2025

3.3

Factor of 2 of Min Distance Given a set P of n points in $\mathbb{R}^2$, compute all pairs of points with distance within a factor of 2 of the minimum pairwise distance.

Recursive Specification:
`min-dist-2`(P, δ): Given a set of points P and the minimum distance δ between any two points in P, return all pairs of points whose distance is within 2δ.

Recursive Implementation:

<u>`min-dist-2`(P,δ)</u>

1. // Preprocessing:
2. // We assume P is sorted by y-coordinate. (Otherwise sort P once in a preprocessing step.)
3. // we calculate δ using min-distance(P) in preprocessing as that takes O(nlogn)
4. Let $n \leftarrow \text{size}(P)$.
5. If $n \leq 1$ return $\emptyset$.
6. If $n = 2$ then
   A. If the distance between the two points is less than 2δ then
      1. Return P.
   B. Else
      1. Return $\emptyset$.
7. Let m be the median point by x-coordinate. // $O(n)$
8. Let $P_1 = \{(x, y) \in P : x < m\}$ and $P_2 = \{(x, y) \in P : x \geq m\}$
9. min-pairs-left = `min-dist-2` (P_1, δ)
10. min-pairs-right = `min-dist-2` (P_2, δ)
11. Let min-pairs be an empty set.
12. // Processes below each run in O(n) time

13. Let $P_3 = \{\ (x, y) \in \mathrm{P} : |x - a| < 2\delta\}$ and let $p_1, ..., p_k$ list P_3 in decreasing order of y-coordinate.

14. For all i, j with $i < j < i + 19$ do

    A. If $||p_i - p_j|| < 2\delta$ then append (p_i, p_j) to min-pairs.

15. Return min-pairs-left $\cup$ min-pairs-right $\cup$ min-pairs // $\in O(n)$, since the three sets are disjoint

Using the Recursive Algorithm to Solve the Problem:

To solve the original problem:

1. Pre-process the points P by sorting them by their y-coordinate.
2. Compute the minimum pairwise distance δ using the algorithm `min-distance` (P).
3. Call the recursive algorithm on P and δ: `min-dist-2` (P, δ).
4. The return value will be a list of all pairs of points in P whose distance is within 2δ.

Analysis of Running Time: Let $T(n)$ represent the running time of the recursive algorithm for an input P containing n points. Given that each computation in one function call of `min-dist-2` has a time complexity of at most $O(n)$, the recurrence relation for $T(n)$ is given by

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 2 \\ 2T(n/2) + O(n) & \text{otherwise} \end{cases}$$

where the term $2T(n/2)$ accounts for the recursive calls on the left and right halves of the x-median of P and the term $O(n)$ accounts for the other computations. Note how the restraint applied variables i and j in line 14 makes the for-loop linear.

This recurrence is the same recurrence that models divide-and-conquer sorting algorithms like `merge-sort`, and the recurrence can be evaluated as

$$T(n) = O(n \log n)$$

Thus, the algorithm runs in $O(n \log n)$ time.

Proof of Correctness: We prove by induction on the size $n = |P|$ of the input array P that the algorithm correctly computes all pairs of points whose distance is within 2δ.

Base Case: When $n \leq 1$, there are no pairs to compute; in this case an empty set is correctly returned. When $n = 2$ and the distance between given 2 points is less than 2δ we return the pair of points contained in P. Otherwise, if the two points in P differ by more than 2δ, then we return an empty set.

Inductive Hypothesis: Consider the general case where $n > 2$. We assume the algorithm `min-dist-2` correctly returns all pairs with distance within 2δ of each other for all input sizes strictly less than n.

Inductive Step: For a set of size $n > 2$ by inducting on n = $|P|$:

1. The algorithm divides P into two subsets P_1 and P_2, each of size at most $n/2$. By the inductive hypothesis, it correctly computes all close pairs within each subset.

2. The algorithm then checks for cross pairs between points in the left and right subsets, P_1 and P_2.

   This part is the bulk of what makes the runtime effcient. Essentially, in computing cross-pairs between P_1 and P_2, we can exclude points in P_1 and P_2 that are too far away from the other set of points. In particular, it suffices to only look inside the strip of points $\{(x, y) \in P : |x - m| < 2\delta\}$ since if a point in P_2 is at least than 2δ from the median point, then it cannot be within 2δ of any point in P_1. On the other hand, if a point in P_1 is at least than 2δ from the median point, then it again cannot be within 2δ of any point in P_2.

   So, all the cross-pairs $p_1 \in P_1$, $p_2 \in P_2$ of points that have $||p_1 - p_2|| < 2\delta$ must have $p_1, p_2 \in P_3$, where $P_3 = \{(x, y) \in P : |x - m| < 2/\delta\}$.

   Now, we can further pare down the cross-pairs we examine. Observe that a piece of the strip P_3 with height 2δ can have no more than 19 points, given that each pair of points is at least δ apart (see *The Packing Argument* below).

   This observation makes sorting the points in P_3 by y-value and then only trying pairs that are within 19 points of each other a comprehensive way of finding all cross-pairs that satisfy with distance less than 2δ.

3. Finally, we merge together the sets min-pairs-left, min-pairs-right, and min-pairs together. Recall that by induction, min-pairs-left covers all pairs of points in P_1 within 2δ of each other and, similarly, min-pairs-right covers all pairs of points in P_1 within 2δ of each other. The correctness of min-pairs, which finds all cross-pairs with distance less than 2δ, is discussed above.

   The function therefore correctly returns all pairs of points in P with distance less than 2δ when P is of size n.

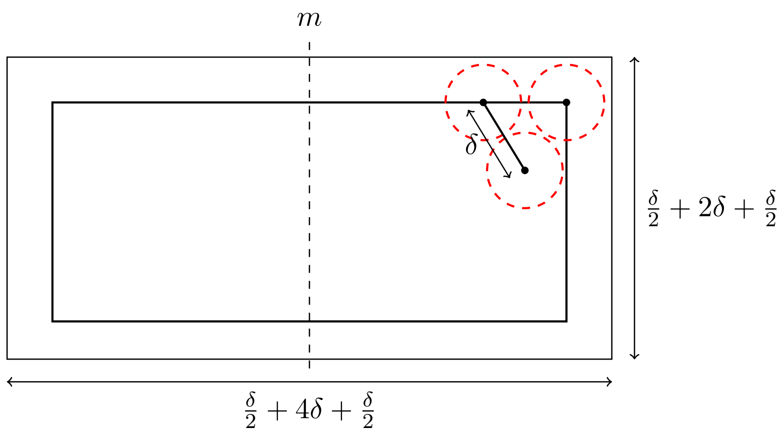
Thus, by induction, the algorithm correctly computes these pairs for all input sizes $n \geq 0$.

The Packing Argument

Here we reiterate the packing argument made for `min-distance`, except now our strip P_3 is 4δ wide. Note that the points in P_3 are still at most δ apart.

Claim. Take a horizontal section of the strip P_3 with height 2δ. No more than 19 points may fit inside this section.

Consider the picture below.



The inner rectangle, which has width 4δ and height 2δ, is section of P_3. Any valid packing of points (such that no pair has distance less than δ) corresponds to a valid packing of balls of radius $\frac{\delta}{2}$ into the outer rectangle, which pads the original rectangle with a frame of width $\frac{\delta}{2}$.

The maximum packing of balls into the outer rectangle is no more than (and probably a bit less than) what we get from comparing areas. Hence, we have

$$\text{max number of points} \leq \left\lfloor \frac{(3\delta)(5\delta)}{\pi(\frac{\delta}{2})^2} \right\rfloor = 19$$

so such a section of P_3 can contain no more than 19 points.