8 Closest pair [KT 5.4]

Definition and motivation. The Closest Pair of Points problem is formulated as follows. Given a set of n points on a 2D plane: $P_1(x_1, y_1), \ldots, P_n(x_n, y_n)$. The goal is to find out a pair of points with closest distance, i.e.

$$\min_{i \neq j} \left\{ dist(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right\}.$$

This is a basic problem in *computational geometry*, which finds applications in graphics, computer vision, GIS system, etc.

Note that if the problem was on 1D, then we can simply use sorting. However, in 2D, sorting on x or y coordinate will not work. Also note that the naive algorithm take time $O(n^2)$ to enumerate all pairs of points. Our goal is to do this faster via divide-and-conquer.

We assume that no two points have the same x or y coordinates; this can be done by a small rotation.

Solution ideas: Now let us dive into details. Given a set of n points, a natural way to divide it into subproblems is to draw a line and let all the points on the left side of the line belong to set Q and all the points on the right side belong to the set R. We draw the line in a way so that both Q and R contain half of the points. See Figure 5.6. Now two obvious subproblems are: δ_1 is the distance between the closest pair of points in Q, and δ_2 is the distance between the closest pair of points in R.

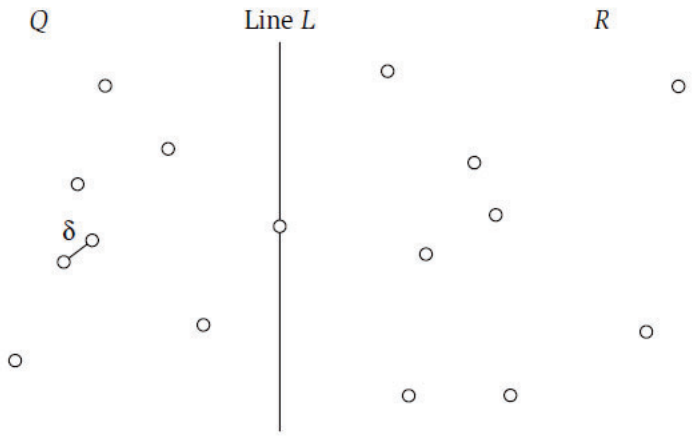

Figure 5.6 The first level of recursion: The point set *P* is divided evenly into *Q* and *R* by the line *L*, and the closest pair is found on each side recursively.

Our final goal is to compute the distance between the closest pair of points in $Q \cup R$. Besides the two subproblems, we also need to consider the pairs crossing the boundary between Q

and R. That is, we want to compute $\min\{\delta_1, \delta_2, \min_{i\in Q, j\in R}\{dist(P_i, P_j)\}\}$. To directly compute $\min_{i\in Q, j\in R}\{dist(P_i, P_j)\}$ via enumeration takes time $O(|Q|\,|R|) = O(n^2)$, which is not desirable. Therefore we need to come up with a smarter way.

Let $\delta = \min\{\delta_1, \delta_2\}$. We note that if all cross pairs have distance greater or equal to δ, then we do not need to know the minimum distance between cross pairs: δ will be our final solution. So we are only looking for the minimum distance between cross pairs when this distance is smaller than δ. This observation can slightly simplify our task as follows. Let σ_Q be the set of points in Q with x coordinate δ-close to x_0, and let σ_R be the set of points in R with x coordinate δ-close to x_0. We are only interested in $\min_{i\in\sigma_Q, j\in\sigma_R}\{dist(P_i, P_j)\}$ as all other points in Q will not contribute a cross pair with distance smaller than δ. Intuitively, σ_Q is a vertical strip of points with x-coordinate in the range $[x_0 - \delta, x_0]$; σ_R is a vertical strip of points with x-coordinate in the range $[x_0, x_0 + \delta]$.

However, computing $\min_{i\in\sigma_Q, j\in\sigma_R}\{dist(P_i, P_j)\}$ directly may still take time $O(n^2)$ as it could be the case that all points are in the strip area σ_Q and σ_R. We still need more ideas to reduce the time complexity.

For each point (x, y) in the strip area, we only need to check its distance with another point whose y-coordinate in the range $[y - \delta, y + \delta]$. This is because if two nodes have y-coordinates difference greater than δ, their distance will be greater than δ. Therefore, we only need to check the nodes in the rectangle area. Note that this rectangle has length 2δ and height 2δ. We can break this rectangle into 16 squares with length $\delta/2$. See Figure 5.7.

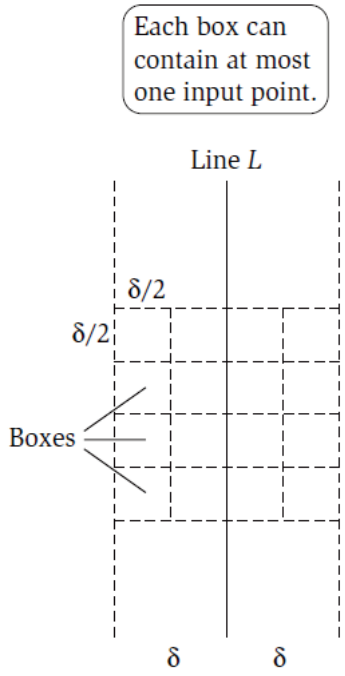

Figure 5.7 The portion of the plane close to the dividing line L, as analyzed in the proof of (5.10).

The longest distance of two points in each square will be at most $\delta/\sqrt{2} < \delta$. Since each square completely lies in either the left strip or the right strip, there can be at most 1 point in each square (otherwise there will be a pair of points with distance smaller than δ in Q or R, which is a contradiction). This shows that there can be at most 16 points in the rectangle area $[x_0 - \delta, x_0 + \delta] \times [y - \delta, y + \delta]$, i.e. we only need to check the distances between any point (x, y) with at most 15 points in the strip area $\sigma_Q \cup \sigma_R$. If we sort all the points in the strip area according to y-coordinate, these 15 points will be next to the given point in the sorted list.

The algorithm. The ideas above gives the following divide and conquer algorithm.

Closest-Pair$(n, P[])$

IF $n \leq 1$ THEN RETURN $+\infty$

$m = n/2$; $Q[] = P[1, m]$; $R[] = P[m + 1, n]$;

$\delta_1 =$ Closest-Pair$(m, Q[])$;

$\delta_2 =$ Closest-Pair$(n - m, R[])$;

$\delta = \min\{\delta_1, \delta_2\}$;

$x_0 = (P[m].x + P[m + 1].x)/2$;

Let $S[]$ be the list of $P[i]$ such that $|P[i].x - x_0| < \delta$;

Sort $S[]$ according to y-coordinates;

FOR each $S[i]$ DO

FOR each $j = i + 1, i + 2, \ldots, \min(i + 8, |S|)$ DO

$\delta = \min\{\delta, dist(S[i], S[j])\}$

RETURN δ

We note that in the pseudocode for each point in $S[i]$, we only consider its distances to 8 other points, not 15 as we discussed earlier. Why this is enough? Think about it. (Of course, the constant will not affect the asymptotic running time at the end.)

Since sorting take $O(n \log n)$ time, the runtime of the procedure above, excluding the two recursions, is $O(n \log n)$. Solving a recurrence as we did in the previous lectures, we conclude that the runtime of the whole algorithm is $O(n \log^2 n)$.

Indeed, we can overcome the sorting bottleneck by preprocessing. We sort all the nodes (both according to x-coordinates and according to y-coordinates) before calling the Closest-Pair procedure, and within the procedure, we always maintain two lists – one sorted according to x-coordinates and the other one sorted according to y-coordinates. The algorithm is describes as follows. P_x and P_y are the lists of same set of nodes, while P_x is sorted according to x-coordinates and P_y is sorted according to y-coordinates.

Closest-Pair-Improved($n, P_x[], P_y[]$)

IF $n \leq 1$ THEN RETURN $+\infty$

$m = n/2$; $x_0 = (P_x[m].x + P_x[m+1].x)/2$;

$Q_x[] = P_x[1, m]$; $R_x[] = P_x[m+1, n]$;

FOR $i = 1$ to n DO

IF $P_y[i].x < x_0$ THEN add $P_y[i]$ to $Q_y[]$ ELSE add $P_y[i]$ to $R_y[]$

$\delta_1 =$ Closest-Pair-Improved($m, Q_x[], Q_y[]$);

$\delta_2 =$ Closest-Pair-Improved($n - m, R_x[], R_y[]$);

$\delta = \min\{\delta_1, \delta_2\}$;

FOR $i = 1$ to n DO

IF $|P_y[i].x - x_0| < \delta$ THEN add $P_y[i]$ to $S[]$

FOR each $S[i]$ DO

FOR each $j = i+1, i+2, \ldots, \min(i+8, |S|)$ DO

$\delta = \min\{\delta, dist(S[i], S[j])\}$

RETURN δ

It is easy to see that the runtime of the algorithm *Closest-Pair-Improved()* is $O(n \log n)$. Of course we need to prepare two sorted lists $P_x[]$ and $P_y[]$ before calling the algorithm. However sorting also takes $O(n \log n)$ time.