

Closest pair in a plane

Ashish Sharma

Rengakrishnan Subramanian

November 28, 2001

Given the Cartesian coordinates of n points in a plane, this paper is concerned with finding the closest pair of points in a time in $O(n \log n)$.

Input

A set P of n points in the plane.

$P = \{p(1), p(2), \dots, p(n)\}$, where $p(i) = \{x(i), y(i)\}$

Output

A pair of points that is closest in the input set.

Algorithm

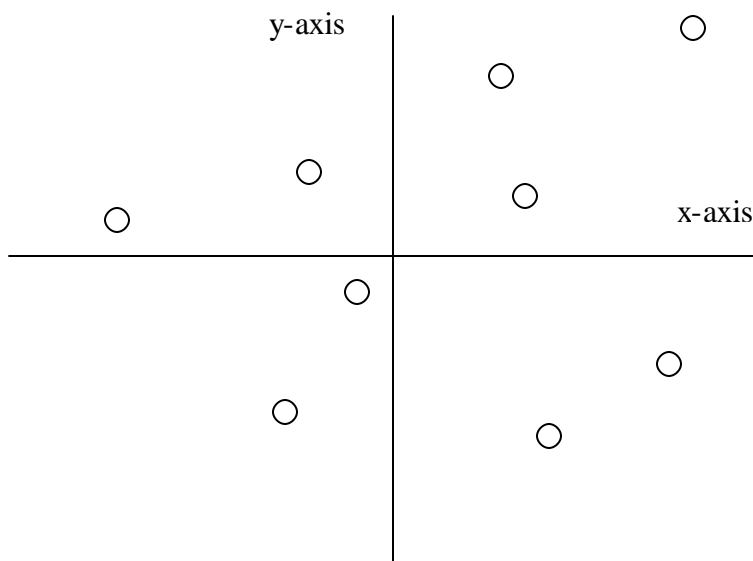
Introduction

The algorithm described here uses a divide and conquer approach to find out the closest pair of points in a plane. It takes an input array of points of the type described as input above. The distance between any two points in Cartesian geometry is defined as:

Distance $\{(x_1, y_1), (x_2, y_2)\} = \text{Square root of } ((x_1 - x_2)^2 + (y_1 - y_2)^2)$.

In this algorithm, we assume that the expression $\text{DISTANCE}(i, j)$ defines the distance between $p(i)$ and $p(j)$ in the input array.

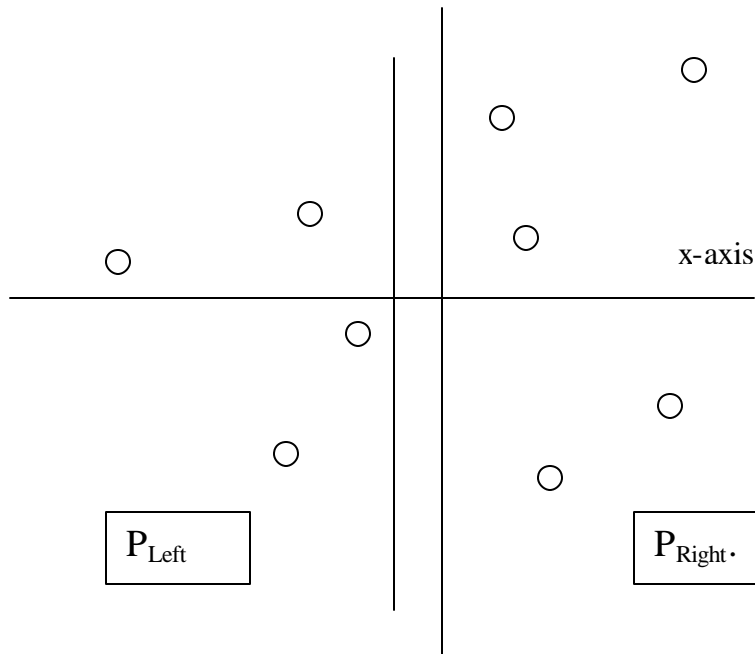
The input can be viewed in the plane as shown below:



Working

We assume that the input array is sorted along the x coordinate of each element. In such a case, for each j , $x(j) \leq x(j+1)$.

The algorithm proceeds by drawing a vertical line across the plane and dividing the plane into two halves, P_{Left} and P_{Right} . The vertical line is placed exactly at the middle between the leftmost and the rightmost point in the input array, which is exactly at $(x_0 + x_n)/2$. The plane can now be viewed as:



The plane is now divided into two halves in such a way that:

$$P_{\text{Left}} = \{ p(1), p(2), \dots, p(n/2) \}$$

$$P_{\text{Right}} = \{ p(n/2 + 1), \dots, p(n) \}$$

There are three possibilities at this stage:

- a. The closest pair is in the left sub-plane
- b. The closest pair is in the right sub-plane
- c. The closest pair is such that one point is in the left sub-plane and other point is in the right sub-plane

The algorithm now keeps on dividing each sub-plane (P_{Left} and P_{Right}) and stops dividing until it finds a sub-plane with only two or three points in it. In this plane, the algorithm uses a procedure SHORTEST (S), which takes in input S , a set of three or lesser points and returns a pair of points which is the closest pair in the set S . This procedure is not

explained in detail in this paper. SHORTEST (S) always runs in constant time, as the size of its input is always less than or equal to three. So, it does not affect the time complexity of the algorithm.

Algorithm

procedure findclosest (P, int n)

{ n is the number of elements in set P }

Set: P_{Left} , P_{Right} , Pleftmin, Prightmin, Pclosest

```

    if (n ≤ 3)
        return SHORTEST (P)

    else
         $P_{\text{Left}} \leftarrow \{ p(1), p(2), \dots, p(n/2) \}$ 
         $P_{\text{Right}} \leftarrow \{ p(n/2 + 1), \dots, p(n) \}$ 
        Pleftmin ← findclosest ( $P_{\text{Left}}$ , n/2)
        Prightmin ← findclosest ( $P_{\text{Right}}$ , n/2)
        Pclosest ← MERGEPLANES (Pleftmin, Prightmin)
        return Pclosest

    endif
endprocedure

```

procedure MERGEPLANES (P1, P2)

```

     $\Delta \leftarrow \text{MINIMUM} (P1, P2)$ 

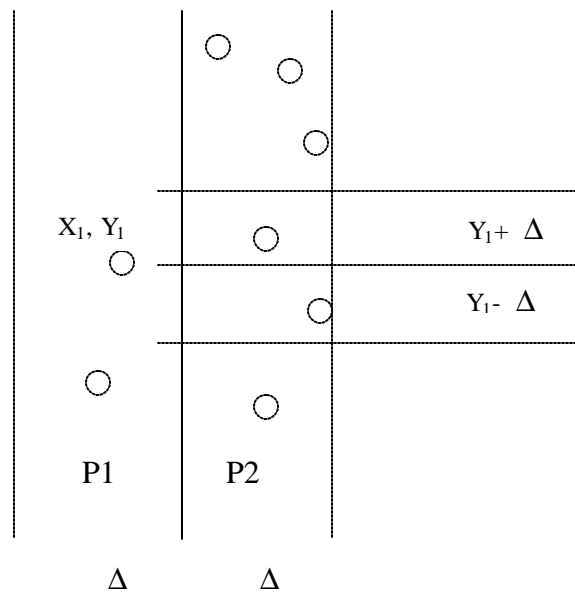
    for (every i in P1)
        for every j in P2 such that
             $x_i < x_j + \Delta$  AND  $(y_i + d > y_j > y_i - d)$ 
            if DISTANCE (i, j) <  $\Delta$ 
                return (i, j)
            endif
        endfor
    endfor
endprocedure

```

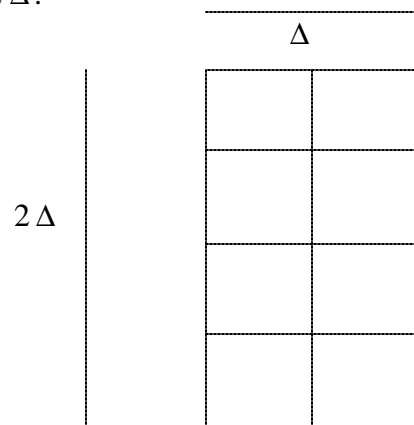
Theorem I: The algorithm MERGEPLANES takes a time in $O(n)$ to find the closest pair between two given planes.

Proof:

We can show that the algorithm MERGEPLANES has the time complexity in $O(n)$ by showing that for every element in plane 1, there are only a fixed number of points to be evaluated. For a point x_i to be considered, it should not be more than Δ distance from the partition. In other words, all points in both the planes which are not more than Δ distance apart will have to be considered in order to find the closest pair.



Also for a particular point on plane P1, the points which we need to consider on plane P2 must have y-coordinates between $Y_1 + \Delta$ and $Y_1 - \Delta$. Hence, we can say that, for a particular point in P1, there exists a fixed area in P2 in which all the points which need to be evaluated exist. This area can be represented in the form of a rectangle with width as Δ and height as 2Δ .



The above rectangle can be divided into 8 equal squares with each square having a dimension of $\Delta/2$. The diagonal distance of a square is $\Delta/\sqrt{2}$, which is less than Δ . Hence, we can say that for all the squares in the specifies rectangle on P2 there cannot be more than one points since Δ is the minimum distance between any two points, and if there are two points then their distance would be less than Δ . Thus, we can say that there can be at most 8 points in P2 that need to be considered while evaluating any point in P1.

In the algorithm, MERGEPLANES, the second for loop will find all points that are inside this rectangle of dimension Δ by 2Δ . So, the second for loop will always execute for only 8 times, which is $O(1)$. The first for loop will execute for $O(n)$ times. So, together the MERGEPLANES algorithm will run in a time complexity of $O(n) * O(1)$. That is, it will run in $O(n)$.

Theorem II: *The algorithm findclosest takes a time in $O(n \log n)$ to find the closest pair on a given plane.*

Proof:

Basis:

The time complexity analysis of this algorithm can be divided into the several stages. Initially, the input set is sorted in the non-decreasing order of the x coordinate. Time complexity of this algorithm is in $O(n \log n)$. The partitioning of the given set of points can take a linear time $g(n) \in O(n)$. After partitioning, there are recursive calls on the two sub-sets each of size $n/2$. It then remains to merge the two sets returned by the recursive calls, which can be proven to take a time in $m(n) \in O(n)$. Thus, in total, the algorithm takes a time complexity of

$$\begin{aligned} t(n) &= t(n/2) + t(n/2) + cn \\ &= 2t(n/2) + cn \end{aligned}$$

Note that the procedure MERGEPLANES takes $O(n)$. This is proved in the Theorem I.

Induction Step:

Consider the induction step for the next iteration in which the set P is further divided into two subsets. The time complexity for both the sets becomes

$$\begin{aligned} t(n) &= 2(2t(n/4) + cn/2) + cn \\ &= 4t(n/4) + 2cn \end{aligned}$$

Similarly, for the next step, the time complexity becomes

$$\begin{aligned} t(n) &= 2(4t(n/8) + 2cn/2) + cn \\ &= 8t(n/8) + 3cn \end{aligned}$$

By Induction hypothesis,

$$t(n) = 2^k t(n/2^k) + k cn$$

Substituting, $n = 2^k$
 $k = \log_2 n$

$$t(n) = n t(1) + n c \log n$$

$$t(n) = n + n \log n$$

$$\mathbf{t(n) = O(n \log n)}$$

References

- [1] Gilles Brassard and Paul Bratley. *Fundamentals of Algorithmics*. Prentice Hall of India, 1998.
- [2] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest. *Introduction to Algorithms*. Prentice Hall of India, 1999.