Computational Geometry

Dr. Marco Pellegrini
Lecture 9: closest points

1

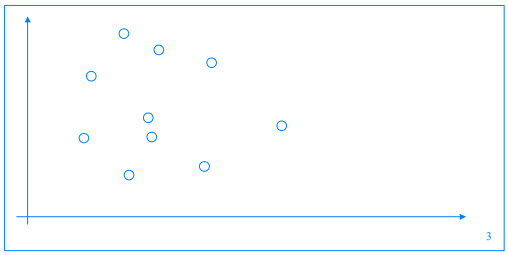
Summary of lecture 9

- Summary of lecture 9.
  Finding the closest pair of points among many points.

2

Finding the closest pair of points

- The distance between 2 points $p_1=(x1,y1)$, $p_2=(x2,y2)$ is $d(p1,p2) = \sqrt{(x1-x2)^2 + (y1-y2)^2}$.
- Given n points $p_1,..p_n$, find the two distinct points that are closest to each other.



3

Applications

- Traffic control: if the points represent vehicles, then the two closest one are those to watch for collisions. (airplanes, cars, people)
- If the points are celestial bodies, the two closest one could be a system of twin stars or a star and planet.
- In biology, if the points represent cells in a culture, then the closet pair of cells are those more likely to be generated by a single original cell.
- Exercise: find your own application in any field you are familiar with (chemistry, biology, statistics, physics, engineering, economics, etc...)

4

Observations

- We can easily find the closest pair by computing all pairs of distances and selecting the smallest one, in the vary same way as we find the smallest among n numbers.
- This algorithm would have complexity $O(n^2)$.
- If $d(p1,p2) < d(p3,p4)$ it is also true that the same holds for the squares: $d(p1,p2)^2 < d(p3,p4)^2$, so we do not need to compute the square root to compare two distances. We can avoid the square root operation which has round-off errors.
- Can we do better than $O(n^2)$?

5

Designing a fast algorithm

We will now go through the design of a fast algorithm (Shamos-Hoey algorithm).

The starting point is: let us see if we gain something by using a divide-and-conquer strategy.

There are many ways to divide a set of points in two sets with roughly the same number of points each.
(1) randomly assign points to the two sets.
(2) all the points in the first set are to the left of all points in the second set.

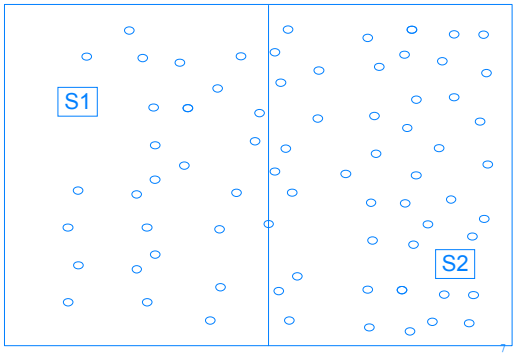
If we choose (1) there is no hint that this is indeed an improvement, so we try (2).

Also, (2) is easy to compute in linear time using a fast

MEDIAN algorithm (see course in algorithms and data structures)

6

Divide step



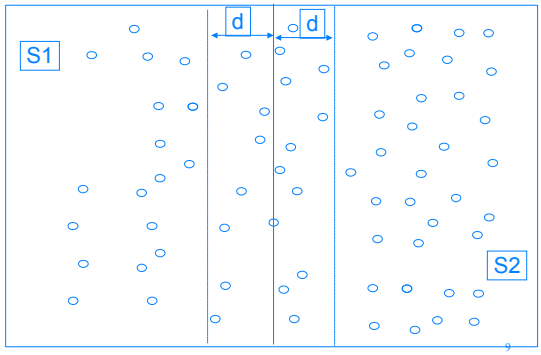
7

Conquer step

- In the conquer step, we just call the procedure to find the closest pair on the two smaller sets of points we have created in the divide phase.
- We are still writing down this procedure, but if we give it a name we can call it. (e.g CLOSEST-PAIR(S:set of points))
- Since this procedure will do what it is supposed to do we will have as a result the two closest distances d1 for points in S1 and d2 for points in S2.
- What is left to find out is if there is a point in S1 and a point in S2 that are closer than the min(d1,d2).
- If yes, return their distance, if not, return min(d1,d2).
- Just to give names, let us call d=min(d1,d2).
- Put a strip of width 2d centered on the median axis.

8

After the conquer step



9

Observations

- Let us call the vertical strip of width 2d H.
- A point of S1 outside H cannot possibly be at a distance less than d from a pint of S2, since there is at least half of H to separate them.
- Similarly, a point of S2 outside H cannot possibly be at a distance less than d from a point of S1, since there is at least half of H to separate them.
- Conclusion, when we look for a pair of points, one in S1, one in S2 closest than d, we need look only in H.

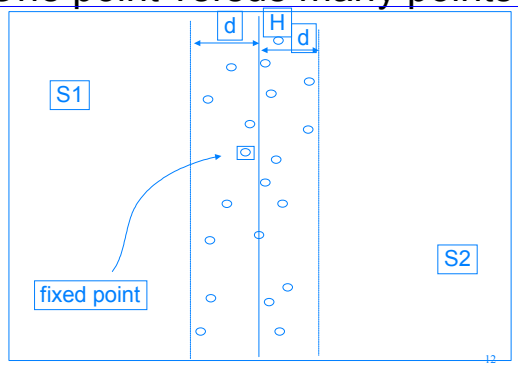
10

More observations.

- The previous observation allowed us to exclude many points form further comparisons. Practically we may have thrown away many points, but to estimate how many is too difficult.
- Now we FIX our attention on 1 point on the left side of H, and on all the points in the right side of H.
- Now we are not trying to compute anything we are just trying to find out if we can exclude more points from the comparisons.

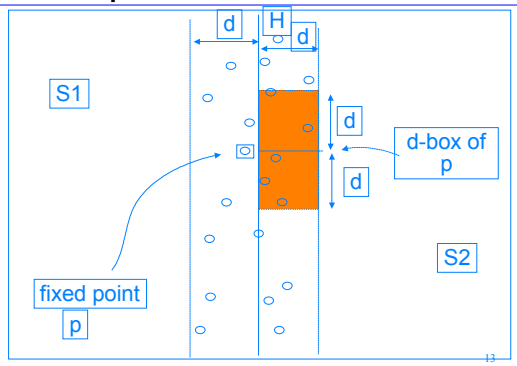
11

One point versus many points



12

A point and its d-box



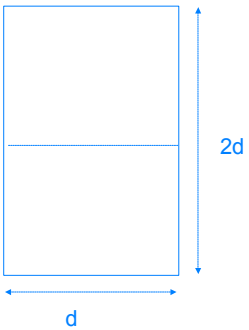
13

Observation 2

- Let us call HR the right side of H.
- A point in HR but outside the d-BOX(p) cannot be at a distance less than d from p since half of the box has height d.
- So for each point p, we need to consider only the points that are inside d-BOX(p).
- We have excluded many points form the comparisons but we do not know yet how many points.
- Exercise: You are given a box 2d times d, how many points you can fit inside the box such that the minimum distance between two points inside the box is equal to or greater than d?

14

A 2d times d box



15

Observations 3.

- In a d-box we can have at most 6 points that are at distance equal to or greater than d among each other.
- We call HL the left side of H. For each point in HL we have to compare such a point with at most 6 points in HR.
- The number of comparison we need is reduced to a number that is less than 6(n/2)=3n.
- How do we find for each p in HL the d-box and the points of S2 in the d-box?
- Also, in the upper half of the d-box there can be at most 4 point.

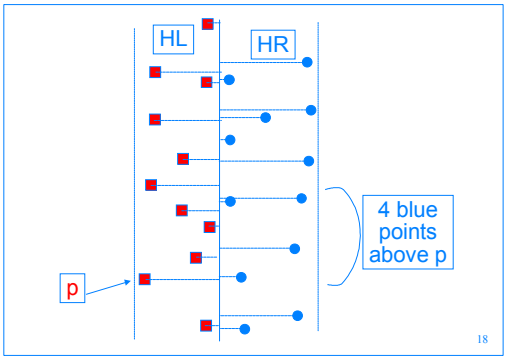
16

Without building boxes

- We do not want to build the d-boxes(p) explicitly for each p in HL, it is too much work.
- We take the points in HL and we call them red, those in HR and we call them blue.
- We sort the red and the blue points by y-coordinate.
- In this sorted order, the blue point in d-box(p) are among the 4 blue points after p (which is red) and the 4 points before p.

17

Without boxes 2



18

Merge phase

```
Combine1(RED,BLUE:sorted array of points, d: distance)
  A = MERGE(RED,BLUE) %the MERGE  of Merge-sort
  i=0
  FOR k=1 TO Length(A) DO
    IF A[k] is red THEN
      i:=i+1, Count[i]:=0
    END IF
    IF A[k] is blue THEN
      j:=i
      WHILE j>0 AND COUNT[j]<4 DO
        d := min(d, Distance(A[k], RED[j])
        Count[j]:= Count[j]+1
        j := j-1
      END WHILE
    END IF
  END FOR
RETURN d
```

19

Merge phase 2

- We need also a procedure COMBINE2 that checks the 4 points before p. (i.e. scans the array A from the last element to the first one).
- Note that the counter makes sure that each red point is involved in only 4 comparisons.
- The MERGE procedure is linear.
- In total COMBINE1 takes time O(n), where n is the number of blue and red points.

20

The whole algorithm

- CLOSEST-POINTS(S: set of points)
  IF |S| < 3 THEN
    Check all pairs, return smallest value.
  ELSE
    m = FIND-MEDIAN(S) % in the x-coordinate
    (S1,S2)= SPLIT(S,m)
    d1 = CLOSEST-POINT(S1)
    d2 = CLOSEST-POINT(S2)
    d = min(d1,d2)
    S1'=POINTS-IN-STRIP(S1,m,d)
    S2'=POINTS-IN-STRIP(S2,m,d)
    d3 = COMBINE1(SORT-Y(S1'),SORT-Y(S2'),d)
    d4 = COMBINE2(SORT-Y(S1'),SORT-Y(S2)',d)
    RETURN min(d,d3,d4)
  END IF

21

Time analysis

- Finding median and splitting takes time O(n). Sorting takes time O(nlog n). COMBINE1 and 2 take linear time.
- Giving name T(n) to the worst case running time we have the equation:
- T(n) = 2T(n/2) + O(nlog n).
- Exercise: Use the tree of procedure calls to prove that the running time is T(n) = O(n $\log^2$ n).

22

Conclusions

- We have discussed the problem of finding the closest pair of points in a set of points.
- It is easy to do it in time $O(n^2)$ for a set of n points.
- We have seen a divide-and-conquer method (Shamos-Hoey method) which attains time O($n\log^2$ n).
- Homework: Can you modify the Shamos Hoey algorithm to make it run in time O(n log n)? Hint: we obtain O($n\log^2$ n) because we sort every time we call recursively CLOSEST-POINT, could you sort only once instead and speed up the algorithm?

23