Assignment 5

Computational Geometry (CS60064)

Suhas Jain | 19CS30048 | suhasjain142@gmail.com
Sarthak Johnson Prasad | 18CS10049 | sarthakjohnsonprasad@gmail.com

Question 1

You are given two sets of points in the plane, $P1$ and $P2$, where $|P_1 \cup P_2| = n$. A partial classifier is a pair of lines $l1$ and $l2$, such that all the points of $P1$ lie on or above l1 and all the points of $P2$ lie on or below $l2$. The cost of the partial classifier is the vertical distance between these lines (see the figure below). Give a geometric interpretation of a partial classifier in the dual plane. What is the cost in the dual setting?

Answer: Given two point sets P_1 and P_2 in the plane, where $|P_1 \cup P_2| = n$ and a partial classifier l_1 and l_2 of the two point sets P_1 and P_2, respectively. A partial classifier is a pair of lines l_1 and l_2, such that all the points of P_1 lie on or above l_1 and all the points of P_2 lie on or below l_2. The cost of the partial classifier is the vertical distance between these lines.
We apply duality transform to all the n points, l_1 and l_2, such that each point p in $P_1 \cup P_2$ maps to a line p^* in the dual plane and each line l in the partial classifier maps to a point l^* in the dual plane.
This duality transformation of $p \mapsto p^*$ and $l \mapsto l^*$ has the following properties:

- It is order-preserving: p lies above l if and only if l^* lies above p^*.
  Since all the points $p \in P_1$ lie on or above l_1 in the primal setting, point l_1^* will lie on or above all the lines $p^* \in P_1^*$ in the dual setting. Also, since all the points $p \in P_2$ lie on or below l_2 in the primal setting, point l_2^* will lie on or below all the lines $p^* \in P_2^*$ in the dual setting.
  We can also say that point l_1^* lies on or in the upper envelope defined by the lines in P_1^* and point l_2^* lies on or in the lower envelope defined by the lines in P_2^*.

- It is vertical distance preserving. Hence the vertical distance between the points $p \in P1 \cup P2$ and lines in the partial classifier is the same as the distance between the points l_1^* and l_2^* and the lines $p^* \in P_1^* \cup P_2^*$.

- Since the lines l_1 and l_2 are parallel in the primal plane, the points l_1^* and l_2^* have the same value of x-coordinate.

The cost of the partial classifier in the dual setting is equal to the absolute difference in the y-coordinates of the points l_1^* and l_2^*.

Question 2

Given a set of n data-points in the 2D-plane and an axis-parallel rectangular box R, the problem is to report all data points included in box R. Show that this problem can be solved using kdtree in $O(k + \sqrt{n})$. time, where k is the number of data points included in the rectangular query-box. Write the recurrence relation and justify the proof.

Answer: We are given a set of n data-points in the 2D-plane and an axis-parallel rectangular box R. Let k be the number of data points included in the rectangular query-box R. We need to show that all the data points included in the box R can be reported in $O(k + \sqrt{n})$ time.
Let v be any node of the KD-tree and let $lc(v)$ and $rc(v)$ denote the left and right child of a node v respectively. The recursive query algorithm to report all the data points included in box R is described below:

Algorithm 1 searchKdTree(v, R)

```
1: if $v$ is a leaf then
2:     Report the point stored at $v$ if it lies in $R$
3: else
4:     if $region(lc(v))$ is fully contained in $R$ then
5:         reportSubtree($lc(v)$)
6:     else if $region(lc(v))$ intersects $R$ then
7:         searchKdTree($lc(v)$, $R$)
8:     end if
9:     if $region(rc(v))$ is fully contained in $R$ then
10:        reportSubtree($rc(v)$)
11:    else if $region(rc(v))$ intersects $R$ then
12:        searchKdTree($rc(v)$, $R$)
13:    end if
14: end if
```

The above algorithm takes as arguments the root of a kd-tree and the query range R. It uses a subroutine reportSubtree(v), which traverses the subtree rooted at a node v and reports all the points stored at its leaves.
We now analyze the time a query takes.

Time complexity Analysis

- First of all, note that the time to traverse a subtree and report the points stored in its leaves is linear in the number of reported points. Hence, the total time required for traversing subtrees in steps 6 and 12 is $O(k)$, where k is the total number of reported points.

- It remains to bound the number of nodes visited by the query algorithm that are not in one of the traversed subtrees. For each such node v, the query range properly intersects $region(v)$, that is, $region(v)$ is intersected by, but not fully contained in the range. In other words, the boundary of the query range intersects $region(v)$. To analyze the number of such nodes, we shall bound the number of regions intersected by any vertical line. This will give us an upper bound on the number of regions intersected by the left and right edge of the query rectangle. The number of regions intersected by the bottom and top edges of the query range can be bounded in the same way.
  Let l be a vertical line, and let T be a kd-tree. Let $l(root(T))$ be the splitting line stored at the root of the kd-tree. The line l intersects either the region to the left of $l(root(T))$ or the region to the right of $l(root(T))$, but not both. This observation seems to imply that $Q(n)$, the number of intersected regions in a kd-tree storing a set of n points, satisfies the recurrence $Q(n) = 1 + Q(n/2)$. But this is not true, because the splitting lines are horizontal at the children of the root. This means that if the line intersects for instance $region(lc(root(T))$, then it will always intersect the regions corresponding to both children of $lc(root(T))$. Hence, the recursive situation we get is not the same as the original situation, and the recurrence above is incorrect. To overcome this problem we have to make sure that the recursive situation is exactly the same as the original situation: the root of the subtree must contain a vertical splitting line. This leads us to redefine $Q(n)$ as the number of intersected regions in a kd-tree storing n points whose root contains a vertical splitting line. To write a recurrence for $Q(n)$ we now have to go down two steps in the tree. Each of the four nodes at depth two in the tree corresponds to a region containing $n/4$ points. (To be precise, a region can contain at most $\lceil\lceil n/2\rceil/2\rceil = \lceil n/4\rceil$ points, but asymptotically this does not influence the outcome of the recurrence below.) Two of the four nodes correspond to intersected regions, so we have to count the number of intersected regions in these subtrees recursively. Moreover, l intersects the region of the root and of one of its children. Hence, $Q(n)$ satisfies the recurrence

$$Q(n) = \begin{cases} O(1), & \text{if } n = 1 \\ 2 + 2Q(n/4), & \text{if } n > 1 \end{cases}$$

  This recurrence solves to $Q(n) = O(\sqrt{n})$. In other words, any vertical line intersects $O(\sqrt{n})$ regions in a kd-tree. In a similar way one can prove that the total number of regions intersected by a horizontal line is $O(\sqrt{n})$. The total number of regions intersected by the boundary of a rectangular query range is bounded by $O(\sqrt{n})$ as well.

Hence, the total time complexity of reporting all the data points included in box R is $O(k + \sqrt{n})$.

Question 3

In a town, all streets are axis-parallel (i.e., the network resembles a rectangular grid), and two consecutive parallel streets are 0.5 km apart. There are n houses scattered around the town. A pizza company wants to set up delivery stations in different locations. Assume that all houses and delivery stations are located just on grid intersections. The time for delivery in minute from a station s(x, y) to a house h(w, z) is equal to the taxi-cab distance between s and h , i.e., |x − w| + |y − z|, in km. Sketch an algorithm that minimizes the number of pizza-delivery stations so that any order can be honoured in at most 15 minutes. Discuss its complexity.

Answer: Given a town in which all the streets are axis-parallel with any two consecutive parallel streets are 0.5 km apart and n houses and some pizza-delivery stations located on the street intersection points, we need to find the minimum number of pizza-delivery stations so that any order can be honoured in at most 15 $minutes$ if the time for delivery in minutes from a station $s(x, y)$ to a house $h(w, z)$ is equal to the taxi-cab distance between s and h, i.e., $|x - w| + |y - z|$.
Observe that for any order to be successfully delivered within 15 $minutes$ to a house, it is necessary that atleast one pizza station lies within 15 km of Manhattan distance from this house. This implies that the locus of points around this house in which atleast one pizza-delivery station must lie is a 45-rotated square from the horizontal axis that has a side length of $15\sqrt{2}$ kms and its center as this house. Any station that lies outside this square will take a delivery time of more than 15 $minutes$.
Now, to minimize the number of pizza-delivery stations, we will place these stations such that each square has atleast 1 pizza delivery station inside or on it. This problem of minimizing the number of points in a point set such that each rectangle contains atleast one chosen point from the point set has already been proved to be NP-hard (paper).
Hence we give an approximate algorithm that tries to find minimum number of stations necessary to fulfill all the orders. We will use a sweep-line technique for this as described below:

Algorithm

1. Rotate the grid 45in the clockwise direction so that all the n squares become axis-aligned.
2. Store all the squares in a list L and sort this list w.r.t. the centers of squares. The events will be the edges of the squares.
3. Sweep a line from left to right and find the region that has maximum number of intersecting squares. Store all the squares that are intersecting in this region.
4. Once the sweep is complete and we have a list of region that has maximum number of intersecting squares, we place one delivery station anywhere in this region and delete all the squares that has an intersection with this region from the list L and add this delivery station to the output list S.

5. Go to step - 3 and keep repeating the steps until there is no square left in the list L.

6. Output list S.

Time complexity Analysis

1. Rotating the grid in step - 1 takes $O(n)$ as n squares are to be rotated.

2. Sorting in step - 2 takes $O(n \: log \: n)$.

3. Sweeping a line from left to right in step - 3 takes $O(n \: log \: n)$ time as there are atmost $O(n)$ events and handling each event will take atmost $O(log \: n)$ time as it will be an insertion and deletion in the binary search tree.

4. Deleting in step - 4 from list L will be amortized $O(n \: log \: n)$ as there are atmost n squares that will need to be deleted from the list.

One iteration of sweeping takes $O(n \: log \: n)$ time and there can be atmost $O(n)$ such iterations in the worst case as no squares might intersect in that case, the total time complexity of the algorithm described above will be $O(n^2 \: log \: n)$.