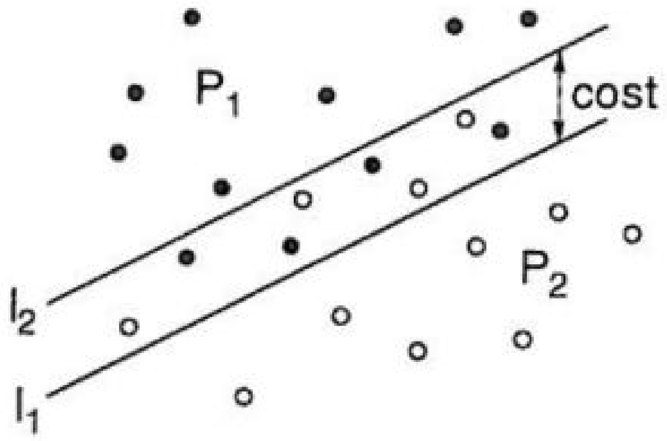
# Assignment 5

**Computational Geometry CS60064**

Pratyush Jaiswal | 18EE35014
Ram Niwas Sharma | 18CS10044

## Question 1

You are given two sets of points in the plane, $P_1$ and $P_2$, where $|P_1 \cup P_2| = n$. A partial classifier is a pair of lines $l1$ and $l2$, such that all the points of $P_1$ lie on or above $l1$ and all the points of $P_2$ 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?

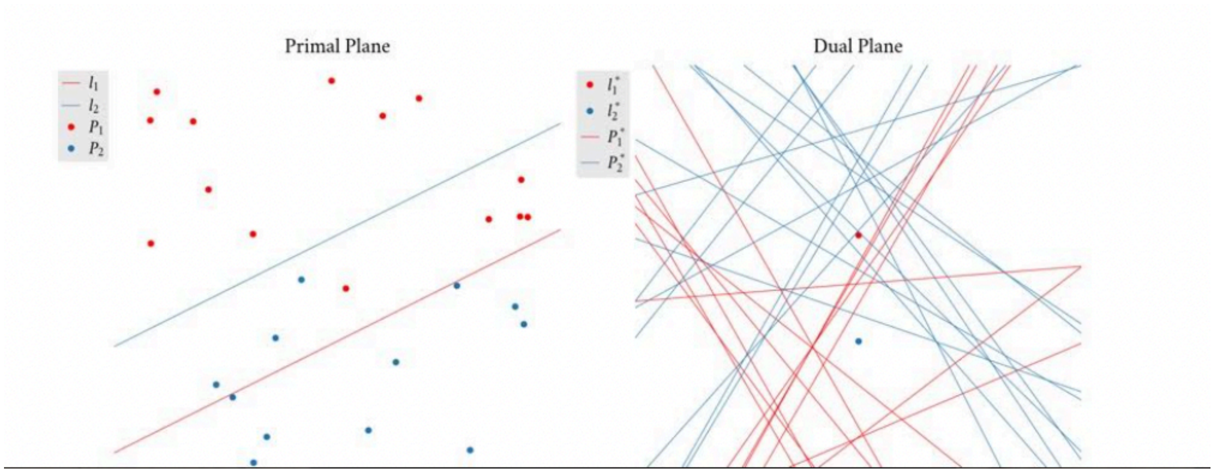


**Answer:**

Let the dual of lines $l_1$ and $l_2$ be the points $l_1^*$ and $l_2^*$ respectively. Further, define $P_1^*$ and $l_1^*$ as the sets of the duals of the points in the original sets.

$$P_1^* = \{p^* | p \in P_1\}$$
$$P_2^* = \{p^* | p \in P_2\}$$

$P_1^*$ is a set of lines, so it represents an arrangement. So does $P_2^*$. $l_1^*$ is a point lying on or above the uppermost level of $P_1^*$, while $l_2^*$ is a point lying on or below the lowermost level of $P_2^*$ as shown in the figure.



Consider an arbitrary $p_1 \in P_1$ and the line $l_1$. I fwe assume some representations for them in the primal plane, we can get their corresponding dual plane representations.

$$\begin{aligned} l_1 \equiv y = mx + c_1 &\implies l_1^* \equiv (m, -c_1) \\ p_1 \equiv (a,b) &\implies p_1^* \equiv y = ax - b \\ p_1 \textit{ lies on or above } l_1 &\implies b \geq ma + c_1 \\ &\implies -c_1 \geq am - b \\ &\implies l_1^* \textit{ lies on or above } p_1^* \end{aligned}$$

This must be true for all $p_1 \in P_1$, since $p_1$ was arbitrary. Hence, $l_1^*$ lies on or above the uppermost level of the arrangement $P_1^*$.
In an identical manner, for some $p_2 \in P_2$:

$$\begin{aligned} l_2 \equiv y = mx + c_2 &\implies l_2^* \equiv (m, -c_2) \\ p_2 \equiv (r,s) &\implies p_2^* \equiv y = rx - s \\ p_2 \textit{ lies on or below } l_2 &\implies s \geq mr + c_2 \\ &\implies -c_2 \geq rm - s \\ &\implies l_2^* \textit{ lies on or below } p_2^* \end{aligned}$$

This implies that $l_2^*$ lies on or below the lowermost level of the arrangement $P_2^*$.

Finally, in the dual plane, a partial classifier for two sets of lines is a pair of points such that all lines of one of the sets lie above or pass through the first points and all lines of the other set lie below or passs through the first point and all lines of the other set lie below or pass through the second point.

The cost in the primal plane is $|c_1 - c_2|$, and it remains the same in the dual plane. The vertical distance between two parallel lines is the same as that between their dual points.

## 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 kD-tree in $O(\sqrt{n} + k)$ time, where $k$ is the number of data points included in the rectangular query-box. Write the recurrence relation and justify the proof.

**Answer:**

Suppose $T$ be a kD-tree containing $n$ points. Let $P$ be the time required to report the $k$ data points in the query box, in which case, $P \in O(k)$. Let $Q(n)$ be the time taken to traverse subtress intersecting (but not lying entirely within $R$) and rooted at a vertical split node.

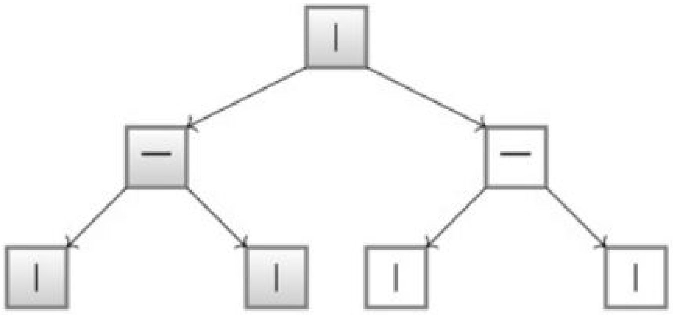


Figure 1: A portion of the kD-tree $T$ used to set up the recursive relation. Vertical split nodes are marked with a "|" symbol; horizontal spit nodes are marked with a "–" symbol. The nodes to be traversed are shaded.

A single node can be traversed in constant time $Q(1) = c$ (say). If the subtree rooted at the topmost node of above figure contains $n$ nodes, the ones rooted at the lowermost shaded nodes contain $\lceil n/4 \rceil$ each.

$$\begin{aligned}
Q(n) &= 2Q(\frac{n}{4}) + 2c \\
&= 2^2Q(\frac{n}{4^2}) + (2^2 + 2)c \\
&= 2^3Q(\frac{n}{4^3}) + (2^3 + 2^2 + 2)c \\
&= .... \\
&= 2^mQ(\frac{n}{4^m}) + (2^m + 2^{m-1} + ...... + 2^3 + 2^2 + 2)c \\
&= 2^mQ(\frac{n}{4^m}) + 2(2^m - 1)c
\end{aligned}$$

If we put $n = 4^m$,

$$\begin{aligned}
&Q(n) = \sqrt{n}c + 2(\sqrt{n} - 1) \\
&Q(n) \in O(\sqrt{n})
\end{aligned}$$

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 reporting is $O(k)$, where $k$ is the total number of reported points.

The total query time is the sum of the traversal time and the reporting time.

$$\begin{aligned}
&T(n) = Q(n) + O(k) \\
&T(n) = O(\sqrt{n} + k)
\end{aligned}$$

So it is showed that this problem can be solved using kD-tree in $O(\sqrt{n} + k)$ time, where $k$ is the number of data points included in the rectangular query-box.

We justify the result as follows. If the query rectangle covers all points, the complexity cannot be better than $O(n)$, so $k$ must appear in the resultant expression. The number of nodes in the subtree rooted at the horizontal split node is $\lceil n/2 \rceil$, by definition of how a kD-tree is constructed. However, treating this number as $n/4$ does not affect asymptotic complexity analysis, because the two values never differ by more than 1, a constant value.

Hence, the recursive relation obtained is justified, and so is the proof.

## Question 3

In a town, all streets are axis-parallel (i.e., the network resembles a rectangular grid), and two consecutive parallel streets are $0.5km$ 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:**

The median variant of the $k$ means clustering technique can be used to find the locations of the delivery stations. With the taxicab norm as the distance measure, the plane is divided into $k$ Voronoi cells. We execute an iterative procedure with exponentially growing values of $k$ because we don't know the value of $k$ (the number of pizza delivery stations we have to minimize).

We propose the randomised approximation algorithm described below.

```
function StattionLocation(H, n, $\tau$):
  for each: $k \in \{2, 2^2, 2^3, ..., 2^{\lceil \log_2 n \rceil}, n\}$
    $C \leftarrow RandomSelection(H, k)$
    $G \leftarrow$ two-dimensional array to store points closest to each point in C

    for $i$ from 1 to $l$ do
      $T \leftarrow KDTree(C)$
      for each: $h \in H$
        if $h \in C$:
          continue
      $Index \leftarrow T.Search(h)$
      $G[Index] \leftarrow G[Index] \cup h$
      $CloseEnough \leftarrow$ true
      for $j$ from 1 to $k$ do
        $C[j] \leftarrow Centroid(G[j])$
        $\delta \leftarrow \max_p TaxiCabDistance(C[i], G[j][p])$
        if $\delta > \tau$
        $CloseEnough \leftarrow$ false
      if $CloseEnough$:
        return $C$
```

Figure 2: Proposed Algorithm

## Explanation

We employ a kD-tree to perform searches instead of the traditional $k$ medians approach. As a result, the asymptotic running time is reduced.

Initially, $k$ points are selected at random from $H$. They are used to build a kD-tree. This tree is used to organise the remaining points into $G[j]$, which is an array of points that are closer to $C[j]$ than any other point in $C$. To put it another way, $G[j]$ denotes a group of points around $C[j]$.

The centroid of each cluster $G[j]$ is then assigned to $C[j]$. Because the question requires it, if the coordinates are not of a grid intersection point, the nearest grid intersection point is chosen. Then we check whether this centroid is inside the cluster's allowed distance from each of the locations. If this is true for all clusters, the algorithm concludes that the answer is those centroids. If not, repeat the steps above with the adjusted $C$.

If $l$ (the number of times the above stages should be performed) is not chosen correctly, the algorithm may fail to find a solution. The usage of $l = \sqrt{n}$ is a typical heuristic. In our situation, this is justified because the number of pizza delivery stations is expected to be significantly lower than the number of residences. Iterations of $l = n$ may be necessary in the worst-case scenario.

All of the preceding steps are repeated for $k \in \{2, 2^2, 2^3, ..., 2^{\lceil \log 2n \rceil, n}$, resulting in a solution with a cardinality that is at least twice that of the optimum. As a result, this algorithm's approximation ratio is $2$.

**Time Complexity**

Each iteration of the loop with $l$ iterations creates a kD tree with $k$ entries. This takes $O(k \log k)$ time to complete. The same tree is searched again with $n - k$ points, taking $O(n \log k)$ time. Following that, a loop of $k$ iterations is run, but its effective complexity is $O(n)$, because the loop processes a total of $n$ points. This reduces a single iteration's complexity to $O(n \log k)$. As a result, $O(n^2 \log k)$ has a total complexity of $O(n^2 \log k)$. This is done for $O(\log n)$ values of $k$, yielding $T(n) \in O(n^2 \log n \log k)$ as the overall time complexity, where $k$ is the number of pizza delivery stations.