Assignment 2 - Triangulation and Linear Programming

Duy Pham - 0980384
Mazen Aly - 0978251
Pattarawat Chormai - 0978675

November 21, 2015

1

- Given a simple polygon P a with n vertices.
- Perform triangulation on P.
- Construct the dual graph G of P.
- Find a root node R of G whose degree is 1.
- Perform $LabelNumberOfChildNodes(G, R)$.
- Select the node v_i whose label is n where $n = max(label_i, 0 < i < n$ and $label_i \leq \lfloor 2n/3 \rfloor)$
- Find the diagonal line corresponding to the edge between v_i and its parent.

Correctness

The first case is when we can pick the node with the exact label, that is $\lfloor 2n/3 \rfloor$. In this case, one polygon that the algorithm returns has exactly $\lfloor 2n/3 \rfloor + 2$ vertices. The other polygon has exactly $\lfloor n/3 \rfloor + 2$ vertices. Hence the algorithm is correct.

The second case, shown in Figure 1, is when we cannot find the exact label, so we have to find the closest node v^*, whose label is the maximum one that is less than $\lfloor 2n/3 \rfloor$. In this case, there is 2 branches starting from the parent of v^*. Hence, the label of the parent of v^* is the sum of the labels of its 2 children, and it is greater than $\lfloor 2n/3 \rfloor$. Therefore, if v^* is the maximum value between the 2 children, then the label of v^* is greater than $\lfloor n/3 \rfloor$. That is,

$$n/3 \leq label(v^*) < 2n/3$$

Algorithm 1 LabelNumberOfChildNodes

```
Require: a dual graph $G$ and a node $v_i$
  Label $v_i$ as $Visited$
  if $degree(v_i) > 1$ then
      $NumNodes = 0$
      for Each neighbor $v_j$ of $v_i$ do
          if $v_j$ is not $Visited$ then
              $NumNodes = NumNodes + LabelNumberOfChildNodes(G, v_j)$
          end if
      end for
      $label_i = 1 + NumNodes$
      Return $label_i$
  else
      $label_i = 1$
      Return $label_i$
  end if
```

So if we cut by the edge between v^* and its parent, the neither of the 2 polygons has more than $\lfloor 2n/3 \rfloor + 2$ vertices. Then the algorithm is correct.

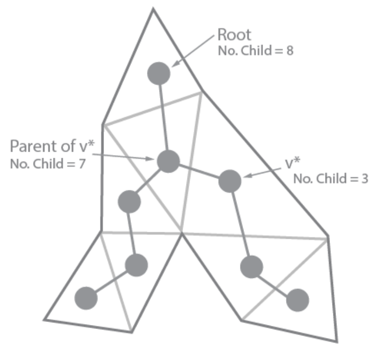

Figure 1: A polygon whose the tree of its dual graph has 2 branches

Running Time

- Performing triangulation on P takes $O(n \log n)$
- Constructing the dual graph G of P takes $O(n)$
- Finding a root node R of G whose degree is 1 takes $O(n)$

- Performing $LabelNumberOfChildNodes(G, R)$ takes $O(n)$ because we traverse each node only once.
- Selecting the appropriate node v_i take $O(n)$.
- Finding the diagonal line corresponding to the edge between v_i and its parent takes constant time.

Thus, the algorithm performs in $O(n \log n)$

3

a

This classification problem will be solved by a randomized incremental algorithm, by first shuffling the points randomly and initially choosing two points of different classes (P_1 and P_2). The line that pass through both of them will be our initial separating line l.

Note: that the separating line divides a plane into 2 halves. For a point to be in a proper position. that means it lies in the half plane which contains only points of the same class.

Then, we will process the next points one by one. and for each point, we will continue looping if it's already in the proper position , if it's not. Our separating line will be changed to pass through this new point and the point of the different class in the previous line. then we will loop on every point that we have so far (we will store every point that we already processed in an Array data structure Z) and check if every point in the proper position.

Algorithm:

$FindSeparatingLine(S)$:

1. Given: Set S of m points of class P_1 and n point of class P_2
2. Output: Separating line (it contains 2 points of different classes) if any or "not found"
3. Initialize empty array Z
4. S $\leftarrow$ RandomPermutation(S)
5. Initialize l our initial separating line to be the line between 2 different random points of P_1 and P_2
6. For every other point $p \in S$:
   (a) insert p to Z

```
    (b) If $p$ in the proper position
        i. continue
    (c) Else
        i. $l \leftarrow$ create new line between $p$ and the point of different class in $l$
        ii. loop on every point in $Z$, if there is a point that not in the proper position, return "not found"
7. return $l$
```

Correctness

Indeed, the algorithm is correct. We will prove it by induction.

Based on our selection, the classifier l always contains 1 point from P_1 and 1 point from P_2.

If the dataset contains only one point in P_1 and one point in P_2, then the algorithm pick the line l passing these points to be the classifier, and this is the correct one.

Assume that after processing point i, the algorithm has already had the correct solution. Let p^* be the next point, and p_i is the point of the same class (P_1 or P_2) which is currently on l. When we insert p^*, the new classifier should group it to the correct group.

If p^* is in the correct order already (the point of P_1 should be above or on l, and the point of P_2 should be below or on l), then the classifier l is correct and the algorithm does not do anything. If p^* is not in the correct order, which means it is on the "outer" space of the group of p_i, the line l should be modified to group p^* to the proper position. Our algorithm decides to rotate the line l so that it contains p^*. This is a proper position for p^* because every point can be on l, and p_i is also in that group because l was rotated to the "outer" direction. Since the old l containing p_i was the correct classifier, this guarantees that the new line l correctly classify the class of p^*.

For the opposite class, the algorithm performs a check to see whether the new classifier l is doing well for them or not. If all of the points are correctly positioned, then l is obviously the correct classifier. If there are some points which are not in the right position, then there are no line classifiers available for this setting. This is because if we want to include those points, we have to move the line l to the "outer" direction of this group, which means the "inner" direction of the class of p^*. Such a movement will remove p^* from the line l, and p^* will be outside of the its proper group. Therefore, there are no agreements in this case. The algorithm indeed returns false, which is correct.

So, if the previous step is correct, then the current step is also correct. This implies that our incremental approach is correct.

Running Time

In order to shuffle all points of P, the algorithm takes $O(n+m)$, then the algorithm will iterate the other points and if the algorithm needs to find a better separating line it has to check all previous processed points for checking correctness, this process takes takes $O(i)$ where i is the number of points have been processed. Hence, the expected running time is :

$$O(n+m)+\sum_{i=1}^{m+n}(Pr[\,p_i \in \text{ProperPosition}\,] * O(1) + Pr[\,p_i \notin \text{ProperPosition}\,] * O(i))$$

We know that $Pr[\,p_i \in \text{ProperPosition}\,] \leq 1$ and $Pr[\,p_i \notin \text{ProperPosition}\,]$ is never greater than the probability of selecting 2 points from i points. Thus, we have :

$$Pr[\,p_i \notin \text{ProperPosition}\,] \leq 2/i$$

Hence,

$$\begin{aligned} T(n+m) &= O(n+m) + \sum_{i=1}^{n+m}(O(1) + O(2/i)O(i) \\ &= O(n+m) \end{aligned}$$

(b)

The worst case is when the algorithm has to determine l every time processing new point. It is obvious to see that the running time will become $O(n^2)$.

For the case that we perform shuffling on P_2 only which takes $O(n)$. Next, the algorithm takes $O(m)$ to find the first separating line between all points in P_1 and one point of P_2. Then, for iterating the other points of P_2, the running time when the algorithm has to find a better separating line changes sligthly to $O(m+i)$. Hence, the expected running time is :

$$O(n)+O(m)+\sum_{i=1}^{n}(Pr[\,p_i \in \text{ProperPosition}\,] * O(1) + Pr[\,p_i \notin \text{ProperPosition}\,] * O(m+i))$$

Hence,

$$\begin{aligned}
T(n+m) &= O(n) + O(m) + \sum_{i=1}^{n} (O(1) + O(2/i)O(m+i)) \\
&= O(m) + 3O(n) + O(m) \sum_{i=1}^{n} 1/i \\
&= O(m) + 3O(n) + O(m)O(\log n) \\
&= O(m \log n)
\end{aligned}$$

We can conclude that if we shuffle only some subset of data, in this case only P_2, the algorithm would perform worse.