Master 1 MoSIG

Algorithmic Problem Solving

APP3 Report
Hole Drilling
Team:
SACAD

Members:
Andrey **SOSNIN**
Majdeddine **ALHAFEZ**
Antoine **COLOMBIER**
Eman **AL-SHAOUR**
Son Tung **DO**

Grenoble, 12 November, 2017

1 Discussion on the current algorithm

1.1 Not an α-approximation

Let us assume that we have N points to drill and they are arranged as shown in figure1 where the distance between two horizontal points equals $2N$ and the distance between two vertical points equals to 1.

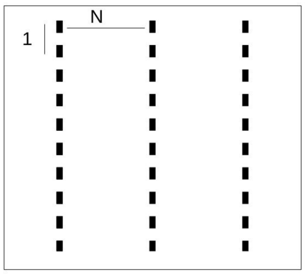

Figure 1: circuit board

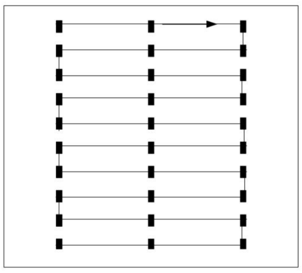

Figure 2: behavior of the current algorithm

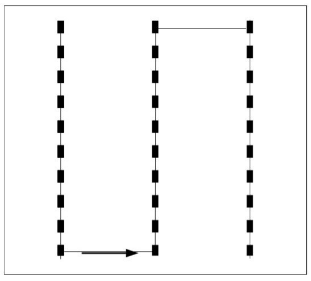

Figure 3: behavior of the greedy algorithm

Figure 2 shows the behavior of the current algorithm (this is the best behavior where after drilling all the points in a row, we drill the point just bellow the last drilled point). The total cost of the algorithm is $C = O(N^2)$

Figure 3 shows the behavior of what seems to be a better solution where we have $C^* = O(N)$ The ratio is $C = O(N)$ and therefore such α doesn't exist and the current algorithm is not an α-approximation.

2 Greedy Approach

2.1 Propose an algorithm

This greedy approach consists of going through the set of points picking up the point with the minimum distance to the current point.

In order to compute the algorithm using the notation of point, we have a data structure for point

```
point {
        int x; /* x-coordinate */
        int y; /* y-coordinate */
}
```

Algorithm 1 Greedy Approach

Input: $A[N]$ array of N points to drill

Output: Array $A[N]$ with the order in which we drill the points

```
for k from 0 to N do
    $index \leftarrow -1$
    $min\_distance \leftarrow \infty$
    for i from k+1 to N do
        $distance \leftarrow find\_distance(A[k], A[i])$
        if $distance < min\_distance$ then
            $index = i$
            $min\_distance \leftarrow distance$
        end if
    end for
    $swap\_elements(k+1, index)$
end for
```

The function $find\_distance(p_1, p_2)$ calculate the Euclidean distance between two points which is given by: $distance = \sqrt{(x_1 - x_2)^2 + (y_1 + y_2)^2}$

2.2 Showing it is not optimal

2.3 Proof it is a 2-approximation

2.4 Proof it is not a 2-approximation

3 Minimum Spanning Tree - MST

3.1 Propose an algorithm

In MST, we have nodes (the notation of point in our circuit board) and edges (represent the distance between 2 points in our circuit board).The idea of proposed algorithm based on the MST is to start with 1 node, we travel to the neighbors nodes. When we come to one neighbor node, we need explore its neighbors, except for the node we come from, until there is no neighbor in that node. At the node which have no neighbor but the previous node, we come back.

Algorithm 2 Algorithm based on MST

Input: MST for the point

Output: The *result_array* stores points in order that we need to move the drill

```
procedure EXPLORE(node, previous_node)    ▷ first call, we can have the previous_node is null
    $result\_array \leftarrow node$                 ▷ add the current node to the result_array
    while exist a neighbor except the previous node do
        $explore(oneneighbor, node)$
    end while
    $result\_array \leftarrow node$     ▷ add the current node to the result_array again due to revisit
end procedure
```

3.2 2-approximation proof

We know that every spanning tree is more expensive than the weight of the MST, thus making a lower bound regarding the optimal solution : $MST \leq C^*$ (1)
Our algorithm consider the MST of the holes, in the worst case, we go through every edge in MST twice. $C = 2 \times MST$ (2)

From (1) and (2) we have $C \leq 2 \times C^*$.
We conclude that the algorithm is a 2-approximation.

4 Kruskal's/Prim's Algorithm