Term Project

IE400- Principles of Engineering Management

Efe Acer - 21602217
Hikmet Demir - 21601158
Talha Murathan Göktaş - 21601917

Bilkent University, CS

Contents

Problem Description

Certain locations on an electronic circuit board (ECB) need to be drilled so that conductive material can be placed between the resulting holes. The drilling machinery used can move its head either horizontally or vertically from a hole to another, in other words, the machinery moves according to the shortest Manhattan distance between the holes. Since the process of moving the head is costly, we want to minimize the total distance covered by the head. Additionally, there exists some blocks on the ECB that may prevent the head to proceed from a hole to another, and the head should return back to its initial position when the process is completed.

Solution Procedure

The input given to us is a set that specifies the locations of the holes (H), and a set that specifies the locations of the blocks (B). These sets are in the following format:

$$H = \{p_1, p_2, ..., p_n\} \text{ where } p_i = (x_i, y_i) \text{ for } i = 1, 2, ..., n$$
$$B = \{b_1, b_2, ..., b_m\} \text{ where } b_i = (x_i, y_i, w_i, h_i) \text{ for } i = 1, 2, ..., m$$

Above, p_i is a 2-tuple representing a point on the ECB by specifying its x and y coordinates; whereas, b_i is a 4-tuple representing a block on the ECB by specifying the x and y coordinates of the blocks lower left corner, together with the width (w_i) and height (h_i) of the block.

The problem description resembles the *Travelling Salesman Problem* (TSP), in which a set of cities and distance between every pair of cities are given and we wish to find the shortest possible circuit that visits every city exactly once and returns to the starting city. In our case, the holes are analogous to the cities and the Manhattan distance between each pair of holes is analogous to the distance between pairs of cities. One slight difference is that since it is is not possible for the head of the drilling machinery to move between certain pairs of holes because of the blocks on the ECB, we need to set the distance between such pairs as infinity.

Cost Matrix

In order to formulate the problem as an instance of TSP, we need to obtain a cost matrix C_{ij} of the form:

$$C_{ij} = \begin{cases} \text{dist}(p_i, p_j) & \text{if } i < j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The function $\text{dist}(p_i, p_j)$ takes the value of the shortest Manhattan distance between holes p_i and p_j if there is no block preventing the drill to move from p_i to p_j, otherwise it takes a value of infinity.

To check whether there exists a preventing block between p_i and p_j, we compute the shortest path between p_i and p_j, then compare the length of the shortest path with the shortest Manhattan distance between p_i and p_j, which is $|x_i - x_j| + |y_i - y_j|$. If the length of the shortest path between the points exceeds the shortest Manhattan distance, then the drill's head must have moved around a block which violates the fact that the head moves according to the shortest Manhattan distance.

To compute the shortest path between the points, we need to implement and execute the *Dijkstra*'s algorithm. We model the ECB as a binary maze where the data corresponding to a location is 1 if that location is occupied by a block and 0 otherwise. The binary maze M_{ij} is defined as follows:

$$M_{ij} = \begin{cases} 1 & \text{if } \exists b_k : (i \geq x_k \wedge i \leq x_k + w_k) \wedge (j \geq y_k \wedge i \leq y_k + h_k) \text{ for } k = 1, 2, ..., m \\ 0 & \text{otherwise} \end{cases}$$

One important note here is that M_{00} corresponds to the location in the ECB where the x coordinate corresponds to $Min_{p_i \in H}\{x_i\}$ and the y coordinate corresponds to $Min_{p_i \in H}\{y_i\}$.

The neighboring cells of M_{ij} in the maze are $M_{i+1,j}$, $M_{i-1,j}$, $M_{i,j+1}$, and $M_{i,j-1}$ provided that those cells are not outside of the maze, to be clearer a cell $M_{i'j'}$ is not outside of the maze if:

$$(i' >= 0 \wedge i' <= Max_{p_k \in H}\{x_k\} - Min_{p_k \in H}\{x_k\}) \quad \wedge$$
$$(j' >= 0 \wedge j' <= Max_{p_k \in H}\{y_k\} - Min_{p_k \in H}\{x_k\})$$

A usual implementation of the Dijkstra's algorithm maintains a priority queue in order to process current neighboring cells in the order of the smallest path length. However, in our maze, every edge has the same unit length. Thus, we can maintain a regular first-in-first-out (FIFO) queue since the values enqueued into the queue will necessarily follow a monotonically increasing order of path lengths. Using a FIFO queue instead of a priority queue turns the Dijkstra's algorithm into a simple *Breadth First Search* (BFS) algorithm, which is indeed a predecessor of Dijkstra's algorithm. A pseudo-code for the BFS algorithm is provided below:

```
1: procedure BFS(source, destination)
2:     Enqueue the source node on a queue and mark it as visited
3:     while the queue is not empty: do
4:         pop the node at the front of the queue
5:         if it is the destination then
6:             return the node and the path to it
7:         for all of the unvisited neighbors: do
8:             Enqueue the neighbor and mark it as visited
9:     If we get here, the destination node is not reachable
```

Now that we have all the mathematical tools in our hand we can write the $\text{dist}(p_i, p_j)$ function explicitly:

$$\text{dist}(p_i, p_j) = \begin{cases} |x_i - x_j| + |y_i - y_j| & \text{if BFS}(p_i, p_j) = |x_i - x_j| + |y_i - y_j| \\ \infty & \text{otherwise} \end{cases} \tag{2}$$

The Model

Since the complete procedure to compute the cost matrix C_{ij} is described we can now move on with the mathematical formulation of the problem in terms of an instance of TSP.

We model the given data as a network $G = (V, E)$, where the set V corresponds to the set H which gathers the locations of the holes, and the set E which corresponds to the edges where the cost of an edge (i, j) is assigned according to C_{ij}. Then, we translate the problem to a network flow model:

Desicion Variables

$$x_{ij} = \begin{cases} 1 & \text{if edge } (i,j) \in E \text{ is in the final circuit} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Objective Function

$$Min \sum_{(i,j)\in E} C_{ij} x_{ij} \tag{4}$$

Constraints

$$\sum_{(i,j)\in E} x_{ij} + \sum_{(j,k)\in E} x_{jk} = 2 \quad , \forall j \in V \tag{5}$$

$$\sum_{(i,j)\in S, i\neq j} x_{ij} \leq |S| - 1 \quad , \forall S \subset V : S \neq \emptyset \tag{6}$$

$$x_{ij} \in 0, 1 \tag{7}$$

Constraint (5) implies that each hole must be connected to two other holes by conductive material. Constraint (6) implies that there should not exist any subtours, meaning that there can not be a case where all holes are connected by conductive material to each other but by means of sub-circuits. Constraint (7) is the integrality constraint. Objective Function (4) enforces the flow to pass from the shortest possible circuit. Decision variables defined by (3) specifies the flow passing from each edge of the graph.

Data Pipeline

The data pipeline we followed to obtain our solution is provided in the diagram below:

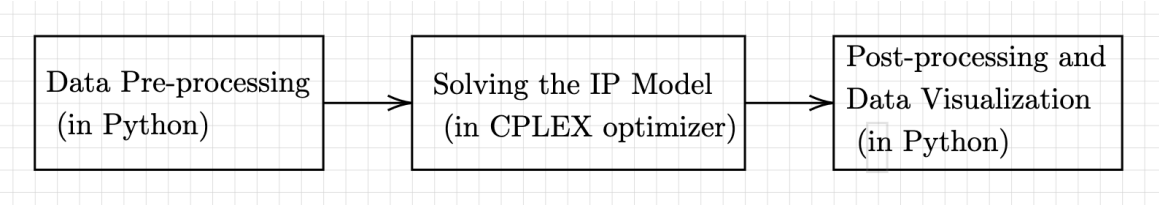

Figure 1: Data Pipeline

The procedure is briefly described below.

Pre-processing

The first step in the pre-processing is to read the excel files given in the assignment and create the corresponding `Numpy` arrays for holes and blocks, this is done using the `Pandas` library.

The second step in the pre-processing is to visualize the dataset so that more insight can be gained. The diagram showing the ECB specified by the dataset is obtained by executing a `Python` script which makes use of `Matplotlib` library. The resulting diagram is given below:

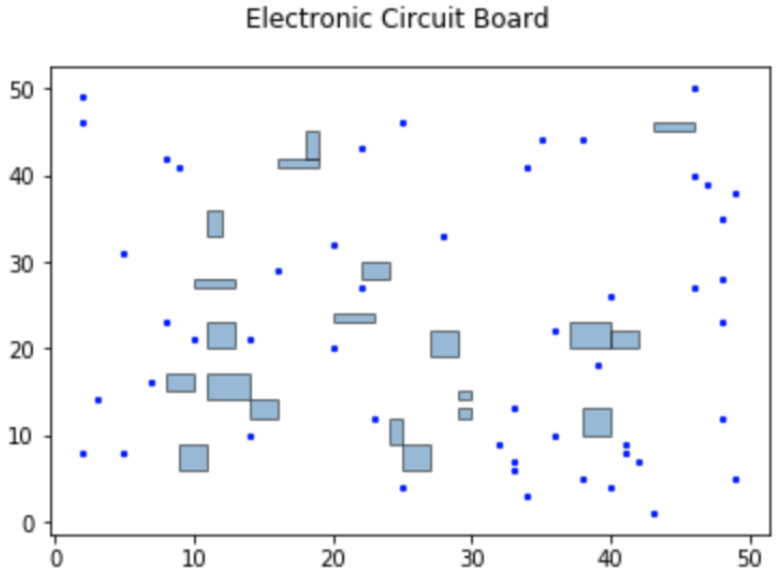

Figure 2: Initial Electronic Circuit Board

The next step is to construct the cost matrix. For that we established a two-dimensional array representing the binary maze that corresponds to the ECB given in the previous figure. Then we run the BFS algorithm on the binary maze from each pair of holes, and calculate costs for these pairs according to equations (1) and (2). We convert the cost matrix to a flattened array which contains the upper triangular part of the matrix in row-major order, this is perfectly valid since the matrix is symmetric. Finally, we convert the flattened matrix to a text file that contains the edge costs in the data format accepted by the `CPLEX` optimizer.

Solving the Model

In this step of the data pipeline, we type our model formulation in `CPLEX` and run the optimizer. To do that we received help from the built-in TSP example that is available in the advanced examples section of the `Oplide` software. We print the resulting visiting order of the circuit and the total circuit length in console. The results for our dataset are:

```
Circuit: 1 -> 14 -> 35 -> 26 -> 18 -> 22 -> 11 -> 12 -> 42 -> 30 -> 43 -> 4 -> 19
    -> 38 -> 8 -> 9 -> 2 -> 23 -> 17 -> 25 -> 36 -> 24 -> 40 -> 13 -> 7 -> 41 ->
    16 -> 37 -> 27 -> 49 -> 29 -> 15 -> 31 -> 21 -> 6 -> 10 -> 44 -> 3 -> 33 -> 39
     -> 34 -> 32 -> 46 -> 45 -> 50 -> 20 -> 47 -> 5 -> 28 -> 48 -> 1
Cost: 382
```

Post-processing

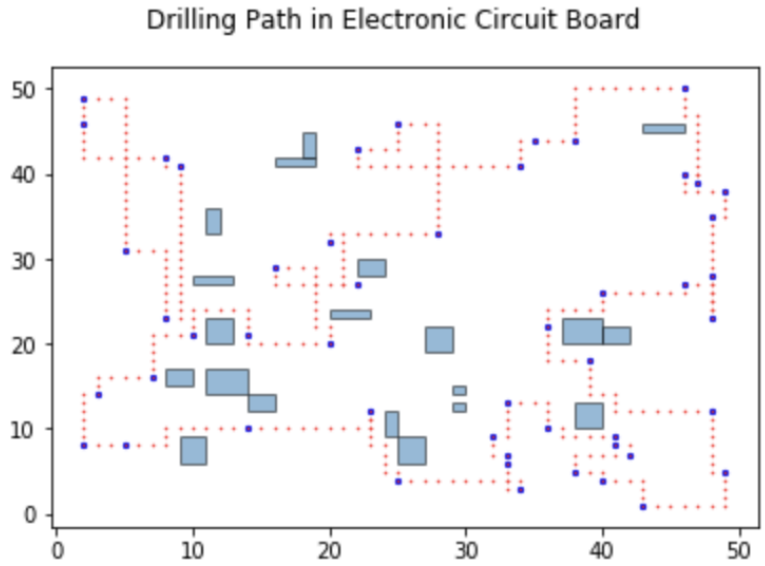

Figure 3: Optimal Drilling Circuit

Once we obtained the console output in the `Oplide` software. We feed this output to our `Python` script as a `string` data. Then we parse this string into an array of points that are represented as locations in our binary maze, which is indeed an array of visit-orders. We iteratively run BFS for all consecutive pairs following the visit-orders to a list of all cells visited in the maze by the optimal circuit. Afterward, we color these cells in our ECB diagram using `Matplotlib`. The result is provided at the end of the previous page.

Note: We submit our `Python` and `CPLEX` codes in separate files. To run the `Python` notebook, `Pandas`, `Numpy`, and `Matplotlib` libraries together with `Jupyter` must be installed.