Lab Exercise

Alexandre Rodrigues (2039952)

January 23, 2022

1 Introduction

The objective of this exercise is to find the minimum time needed to drill all the holes in a electric panel. For part 1, I used the model described in the exercise text. For part 2, I used the Tabu Search with Aspiration Criteria, made some changes and added a heuristic initial solution procedure.

2 Technical Approach

2.1 Cplex model

Sets:

- N = graph nodes, representing the holes;
- $A = arcs(i,j), \forall i,j \in N$, representing the trajectory covered by the drill to move from hole i to hole j.

Parameters:

- c_{ij} = time taken by the drill to move from $itoj, \forall (i,j) \in A$;
- 0 = arbitrarily selected starting node, $0 \in N$.

Decision variables:

- x_{ij} = amount of the flow shipped from $itoj, \forall (i,j) \in A$;
- $y_{ij} = 1$ if arc (i,j) ships some flow, 0 otherwise, $\forall (i,j) \in A$;

The improved formulation in the exercise text is the following:

$$min \sum_{i,j:(i,j)\in A} c_{ij} y_{ij} \tag{1}$$

$$s.t \sum_{i:(i,k)\in A} x_{ik} - \sum_{j:(k,j),j\neq 0} x_{kj} = 1 \qquad \forall k \in N \backslash 0 \tag{2}$$

$$\sum_{j:(i,j)\in A} = 1 \qquad \forall i \in N \tag{3}$$

$$\sum_{i:(i,j)\in A} = 1 \qquad \forall j \in N \tag{4}$$

$$x_{ij} \leq (|N|-1) y_{ij} \qquad \forall (i,j) \in A, j \neq 0 \tag{5}$$

$$x_{ij} \in R_+ \qquad \forall (i,j) \in A, j \neq 0 \tag{6}$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{7}$$

$$\tag{8}$$

2.2 Instancing

Class 1 - Random Randomly selected n positions of the n x n map. Then compute the cost matrix.

Implementation: 1. Compute all possible positions (3x3 example) (0,0) (0,1) (0,2) (1,0) (1,1) (1,2) (2,0) (2,1) (2,2) 2. Randomly shuffle them 3. Select the first n positions from the set.

Class 2 - Domain specific random Same as before but for a n-1 x n-1 map. Limited to 1,2,. . .,N-2 instead of 0,2,. . .,N-1. This because there are basically no electric panels with holes near the border.

Class 3 - Handmade instances for n = 10 I created the following 6 instances from what seemed more realistic. I then created the positions files and a function to compute the cost matrix.

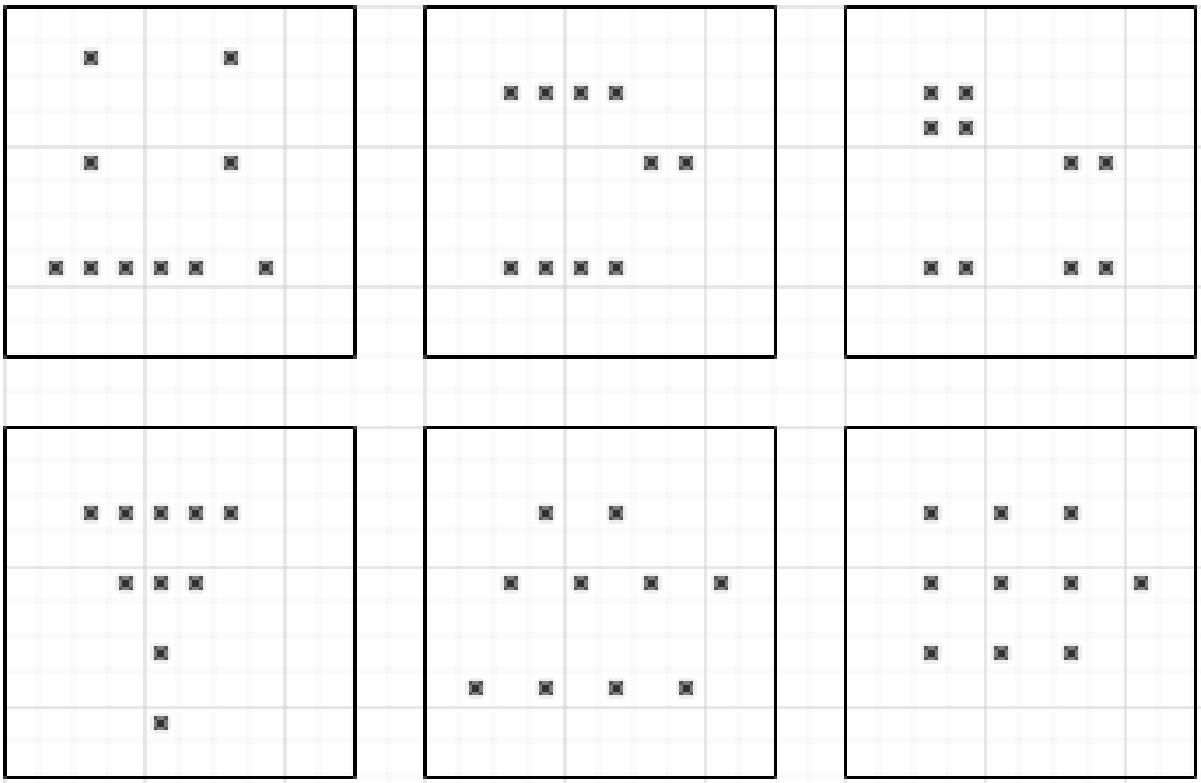

Figure 1: Instances of class 3

2.3 Part 2

I started from the TSAC implementation from the Labs. I implemented the same initialization methods from part1. 1 to randomly select holes positons (either class1 or class1 DS). 1 to compute the costs matrix from these positions. 1 to read from the dists10 intansces as they use positions 1 to better seed the random methods with a mix() function.

Additional Heuristic initialization: Iterate the cost matrix and find the minimum value in each row, excluding the already visited nodes. Sames as slide 24 on m08

3 Results

I tested class 3 only for n=10 with 6 instances, 5 runs each. The cost matrix was computed from the hole positions (random or not) using Manhattan distance. The class 2 instances are random but slightly domain specific. In this case I simply removed the possibility to have holes in the borders of the board.

Class 1 is fully random: board as size = N so $x \in 0, N-1$ hole of size 1

Both class 1 and 2 were tested using 30 random instances for $10 \leq n \leq 70$. Due to the considerable time I reduced it to 10 instances for $80 \leq n \leq 100$. Each instance was run 5 times to get the average computational time.

The time to drill a hole is constant so we can disregard it. The total cost would be $cost_{real} = cost_{exp} + cN, c \in \mathbb{R}$.

Comparison	**exact**	**TSAC1**	**TSAC2**
Time	$0.1165s$	$4.05 \times 10^{-5}s$	$1.06 \times 10^{-5}s$
InitialSol	N/A	47.4	24.0
FinalValue	22.33	22.5	22.33

Table 1: Comparison for class3

TSAC is clearly faster to converge Using an heuristic initial solution improves convergence time and final solutions found. On can also see that our initial heuristic solution is half of the random initial solytoons. This will be even more noticeable for larger n, however the time differences will become desprezabel.

n	**class1**	**class2**
10	$0.1199s$	$0.1285s$
20	$0.430s$	$0.461s$
30	$1.575s$	$1.397s$
40	$6.603s$	$6.570s$
50	$14.04s$	$14.55s$
60	$28.80s$	$28.14s$
70	$46.66s$	$53.88s$
80	$90.96s$	$86.53s$
90	$215.1s$	$207.97s$
100	$458.5s$	$312.64s$

Table 2: Average Time for Exact

There are no notcieable differences reagrding convergence time between the classes.

For part2 the TSAC method found a good enough solution in a maximum of 0.0082 seconds for n=100. Ther were no significant differences between the classes and initializatuion methods used.

Assuming a max time as 20 seconds.... we can solve for up to 50 nodes.

n	**class1**	**class1 init2**	**class2**	**class2 init2**
10	1.54×10^{-5}	9.35×10^{-6}	1.97×10^{-5}	1.04×10^{-5}
20	9.11×10^{-5}	1.40×10^{-4}	1.00×10^{-4}	1.21×10^{-4}
30	2.33×10^{-4}	2.54×10^{-4}	2.31×10^{-4}	3.47×10^{-4}
40	3.49×10^{-4}	7.53×10^{-4}	3.42×10^{-4}	7.22×10^{-4}
50	7.97×10^{-4}	6.74×10^{-4}	6.68×10^{-4}	9.80×10^{-4}
60	1.13×10^{-3}	2.18×10^{-3}	1.70×10^{-3}	2.05×10^{-3}
70	1.91×10^{-3}	2.54×10^{-3}	3.16×10^{-3}	3.11×10^{-3}
80	2.56×10^{-3}	3.96×10^{-3}	2.97×10^{-3}	3.07×10^{-3}
90	4.98×10^{-3}	5.93×10^{-3}	3.87×10^{-3}	5.73×10^{-3}
100	5.83×10^{-3}	8.20×10^{-3}	5.14×10^{-3}	8.66×10^{-3}

Table 3: Average Time for TSAC

All this computational times are below 1 second, so very easily solvable. As an extra test, for n=1000, times are around 5 seconds. The following tables show the final solution values obtained

n	**exact**	**initial1**	**final1**	**initial2**	**final2**
10	34.067	66.067	34.067	38.067	34.067
20	94.33	258.2	95.067	110.8	94.933
30	167.067	606.53	170.2	202.6	169.6
40	257.4	1076.87	266.3	310.8	263.93
50	356.6	1630.47	366.8	428.067	366.467
60	461.93	2418.6	482.73	570.4	479.267
70	578.93	3264.13	607.93	732.73	599.6
80	703.4	4343.2	739.6	869.6	724.2
90	833.6	5476.8	888.2	1052.2	869
100	976.6	6670.2	1042	1228.2	1013.4

Table 4: Solutions for class1

n	**exact**	**initial1**	**final1**	**initial2**	**final2**
10	28.533	52.333	28.533	32	28.533
20	84.467	233.13	84.733	98	84.6
30	160.933	562.73	164.4	197.53	163.067
40	243.4	1040.4	252.53	289.33	248.8
50	339.73	1583.53	350.2	412.4	346.53
60	445	2356.67	466.2	527.93	458.267
70	560.13	3203.8	593.6	898.867	582.4
80	687.6	4163.4	730	859.8	711.2
90	815.4	5358.4	845.4	1016	845
100	951.8	6665	1022	1172.2	988

Table 5: Solutions for class2

The initialization using the min each row heuristic is clearly a better approximation to the optimal solution. Although theoretically there is no guarantee that starting from a better solution gets us a better final solution, this was mostly the case for this implementation.

We can although notice a tendency to worse solutions when we increase n.

NOTE: After most testing there as a clear slowdown due to someone running an intensive program in the Math department server I was using. There were some bug fixes after this but only the solution values were saved and used in this report. Computational time should ideally be approximately the ones showed here.

4 Conclusions

The TSAC method is very fast and can get almost optimal solutions for small n. For large n the cplex is very slow but can be the better option if we want to reduce the drilling time as much as possible.