INDR371 - HW6

Umur Berkay Karakaş

January 3, 2023

Question 1

First thing to note is that if warehouse C is selected in the first step of the greedy algorithm, no other warehouse can be added since for both warehouse A and B, cost difference would be 300+(100-200)=200, which would increase the cost. In a solution which does not consist of warehouse C in any of 3 locations, the optimal one would be warehouse A at location 1, warehouse B at location 2, and either warehouse A or B at location 3, with a total cost of 1300. Therefore, if C is selected first by the greedy algorithm with a total cost higher than 1300, the greedy algorithm will 100% result in a non-optimal solution. The total costs in the first step are 1900 for warehouse A and B, and 900+X for warehouse C. So, if X is between 400 and 1000, we would obtain a non-optimal solution. If X is more than 1000, we would either pick warehouse A and B first. In the next step, for both cases the cost difference for warehouse C would be X+(200-1000)=X-800 and for the other warehouse it would be 300+(100-1000)=-600. So it means that warehouses A and B would be opened after step two and warehouse C will not be opened under any condition when it's greater than 1000. To sum up:

- If X is less than 400, then greedy algorithm would be initialized with C and the result will be optimal with a total cost less than 1300.
- If X is between 400 and 1000, then the greedy algorithm would be initialized with C and the result will not be optimal since its total cost would be larger than 1300.
- If X is larger than 1000, then the greedy algorithm would be initialized with A or B and the result will be optimal with a total cost of 1300.

Therefore, the largest integer X value for greedy algorithm not to result in an optimal solution is 999. (1000 is not viable since tie-breaking would favor A or B in the first step and greedy algorithm could still find an optimal solution).

Question 2

There can be at most 3 unique 2x2 layouts considering their mirror versions. The possible unique layouts are:

1:

1	2
3	4

2:

1	4
3	2

3:

1	4
2	3

- TCR of the first layout = A + E + 2O + 0.5(E + O) = A + 1.5E + 2.5O
- TCR of the second layout = 2E + 2O + 0.5(A + O) = 0.5A + 2E + 2.5O
- TCR of the third layout = A + E + 2O + 0.5(E + O) = A + 1.5E + 2.5O

In the question, third layout was assumed to be the optimal one. Among the candidates, only TCR of the second layout is different than the third so it is the only candidate to disprove the assumption.

Assume that TCR of the second is greater than the third:

$$0.5A + 2E + 2.5O \geq A + 1.5E + 2.5O$$

after the arrangements we get:

$$0.5E \geq 0.5A$$

which contradicts with the assumption that an A relationship has a strictly more value than an E relationship and an E relationship has a strictly larger value than an O relation. This is a proof by contradiction, therefore third layout is one of the optimal layouts.

Question 3

```
In [2]: df = pd.read_excel("schedule.xlsx", index_col=0)
        data = df

In [3]: data.head(5)
Out[3]:
```

Job	M1	M2
1	229	448
2	72	157
3	346	495
4	75	323
5	138	387

```
In [4]: L1 = []
        L2 = []
        while len(data)>0:
            min_time = min(data.min())
            cell = data.where(data==min_time).dropna(how='all').dropna(axis=1) ## get the cell of the minimum value
            job = cell.index[0] ## job number of the minimum processing time
            machine = cell.columns[0] ## machine type of the minimum processing time
            if machine == "M1":
                L1.append(job) ## add the job to the end of L1
            elif machine == "M2":
                L2.insert(0, job) ## add the job to the beginning of L2
            data = data.drop(job, axis=0) ## update the job list
        job_sequence = L1 + L2
        print(' -> '.join([str(elem) for elem in job_sequence])) ## you can comment this line and
                                                                 ## uncomment the line below if you want
        ## print(job_sequence)

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

Figure 1: The code snippet for importing the data and implementing Johnson's algorithm

The algorithm results in the following job sequence:

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