ID: 300635306

COMP307/AIML420 – Fundamentals of AI
Assignment 4: Planning and Scheduling

Part 1: Job Shop Scheduling

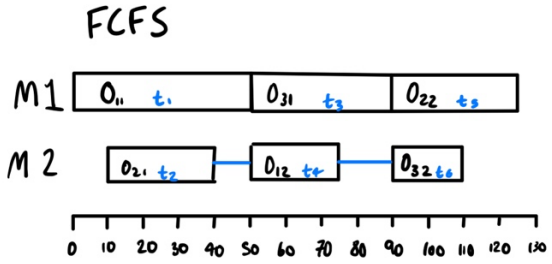
Problem Description

The table below shows the schedule problem for "Part 1" of the assignment:

Job	ArrivalTime	Operation	Machine	ProcTime
J_1	0	O_{11} O_{12}	M_1 M_2	50 25
J_2	10	O_{21} O_{22}	M_2 M_1	30 35
J_3	20	O_{31} O_{32}	M_1 M_2	40 20

1. *"Given a schedule whose action sequence is as follows: P rocess(O11, M1, t1) → P rocess(O21, M2, t2) → Process(O31,M1,t3) → Process(O12,M2,t4) → Process(O22,M1,t5) → Process(O32,M2,t6). Since the se- quence is sorted in the non-decreasing order of the starting time, we know that t1 ≤ t2 ≤ t3 ≤ t4 ≤ t5 ≤ t6. Calculate the earliest starting time (t1 to t6) of each action."*

   Here is my Gantt chart for this sequence of operations:



   Looking at this chart gives the following earliest start times (next page)

Earliest Start Times:

- T1 = 0
- T2 = 10
- T3 = 50
- T4 = 50
- T5 = 90
- T6 = 90

These times firstly follow the order of non-decreasing time as given in the question aswell as being the earliest possible times following the problem descriptions:
- Each job has two operations .
- O_{j1} comes before $O_{j2.}$(Can't do O11 and O12 at the same time for example)
- No jobs are processed earlier that their arrival times.
- Each machine process one operation at most at a time.

2. *"For the solution given in Question 1, find the completion time of each job, which is the finishing time of its last operation. Then, calculate the makespan of the solution, which is defined as the maximum completion time of all the jobs."*

I will first calculate the completion time of each job, which is the finish time of the last operation in each job e.g. time that O12 finishes, after O11 has already happened.

Completion Times:

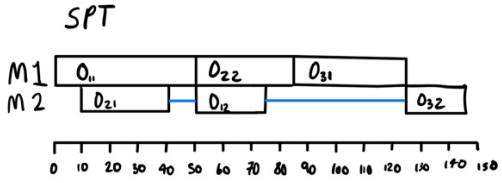
- Job 1 completion time = 75
- Job 2 completion time = 125
- Job 3 completion time = 110

Makespan:

The makespan is just the final/maximum completion time after finishing all of the jobs. In this case the **makespan = 125**

3. *"Write the final solution obtained by the Shortest Processing time (SPT) dispatching rule."*

Gantt chart for the SPT:



Final Solution:

- Process(O11, M1, 0) -> Process(O21, M2, 10) Process(O12, M2, 50) -> Process(O22, M1, 50) -> Process(O31, M1, 85) -> Process(O32, M2, 125)

  This means that these are the following **earliest start times** for each operation:

  - O11 = 0
  - O12 = 50
  - O21 = 10
  - O22 = 50
  - O31 = 85
  - O32 = 125

  This solution follows all of the conditions where OX1 (X = 1,2,3) must come before OX2, and works by processing the shortest process time first. Except in the first one where we do O11 first, as O22 wasn't available at time = 0.

4. *" For the solution obtained by the SPT rule, calculate the completion time of each job and the makespan. Compare the makespan between this solution with that obtained in Question 1 to find out which solution is better in terms of makespan."*

Using the Gantt chart from Q.3 gives the following:

Makespan:
- Makespan = 145

Completion time of each job:
- Job 1 = 75
- Job 2 = 85
- Job 3 = 145

The makespan for the solution obtained with the SPT rule is 145 whilst the solution in Question 1 had a makespan of 125. If makespan was the main concern than the first solution would be a better choice as it is 20 units of time faster than that obtained with SPT. The only use I can see where someone would choose the SPT over Q1 is when you want Job 1 and Job 2 completed shortly after each other as there Is only 10 units of time from Job 1 to Job 2. This however leaves a 60 Units of time gap between the completion of Job 2 and Job 3 compared to a 15 unit of time gap in Q1.

The range of completion times between the two is:
- SPT = 145 – 75 = 70
- Q1 = 125 – 75 = 50

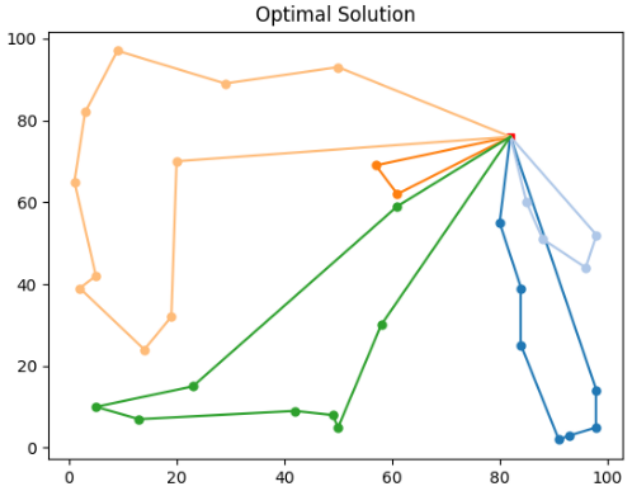
With these values in mind, there is almost no reason to go with the SPT solution over the FCFS solution, unless Job1 and Job2 are the main concern.

Part 2: Vehicle Routing

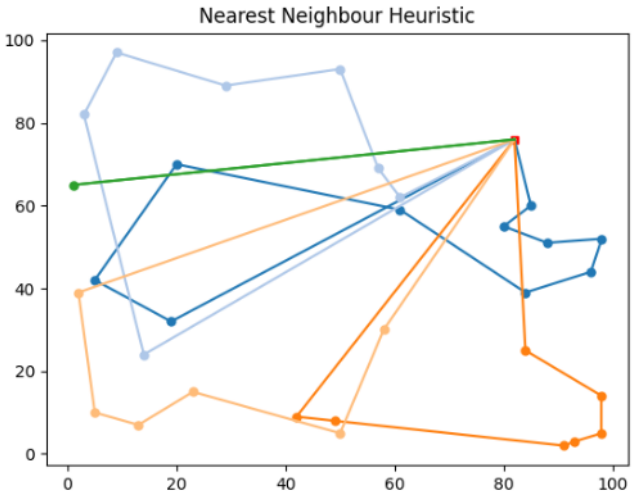
1. *(Code in A4.zip)*
2. *(Code in A4.zip)*
3.

For Part 2 of this assignment I implemented the Nearest Neighbour and Savings heuristics for solving the Vehicle Routing Problem. To do this I first had to get the Euclidean Distance functions in the 'Utility.py' class to work. I will now look over the Python visualisations of the different solutions and discuss the total distances:

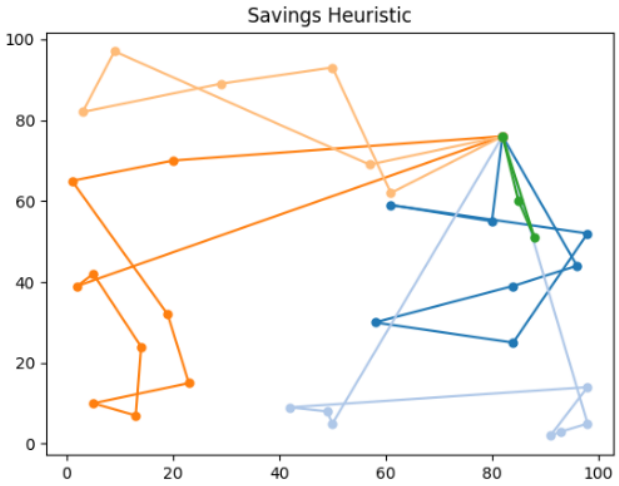
Before looking at the results from the heuristic implementations I will look at the Optimal Solution. Using the Euclidean distance function I added this was the visualisation, where the distance was **787.8**.



Now I will look at the solution from the Nearest Neighbour Heuristic, which had a distance of **1146.4:**



Finally I will look at the solution from the Savings Heuristic, which had a total distance of **998.5**:



Comparing the distances between the 3 solutions in order from best to worst we get:

1. Optimal Solution (Expected)
2. Savings Heuristic
3. Nearest Neighbour

Before talking about the differences of the results I will first briefly talk about the differences between the Nearest Neighbour and Savings Heuristic:

Nearest Neighbour Heuristic:
- This is a (greedy) algorithm which was first used to solve the "travelling salesman problem". In simple terms it selects a starting point (depot) and finds the nearest node, which is calculated using the Euclidean distance. This is done for each route.

Savings Heuristic:
- An algorithm that starts with initialising empty routes from depot to each node, and then calculating the 'savings' for each possible merge, and then merging in a descending order. This normally has a shorter total distance than the Nearest Neighbour Heuristic.

I will add that I'm more confident in the result from my Nearest Neighbour heuristic than my Savings Heuristic as I believe there to be some error in steps 3-4 of the algorithm, but had to move on to other work. If a driver were to follow the routes in my Savings Heuristic they would be in for an interesting ride, whereas the other two options have a more normal route, and would be more useful in a real life scenario.

Overall the differences in total distance is what I expected, as the Optimal Solution having the shortest distance, whilst Savings Heuristic is a bit shorter than the Nearest Neighbour. This is due to the Savings algorithm merging routes based on savings. By considering the savings, it considers the distance between the nodes being merged and also the distances to those nodes from the depot. Whereas the nearest neighbour heuristic works by finding the nearest unvisited node and joining the two then by setting that next closest node as the current node.