COMP307/AIML420 — *Introduction to AI*

Assignment 4: Planning and Scheduling

8% of Final Mark — Due: 23:59 Wednesday 1 June 2022

1 Objectives

The goal of this assignment is to help you understand the basic concepts of planning and scheduling, and simple algorithms for them. In particular, the following topics should be reviewed:

- Represent a schedule of job shop scheduling problem,
- Generate a schedule by a dispatching rule for a given job shop scheduling problem,
- Solve vehicle routing problem using heuristics.

This assignment is largely writing report, and you will need to write the program for solving the vehicle routing problem in Part 3.

Part 1: Job Shop Scheduling [40 marks]

In this part, you are required to find solutions (schedules) for a job shop scheduling problem.

Problem Description

The table below gives a job shop schedule problem with 3 jobs and 2 machines.

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

- (Number of operations) Each job J_j has two operations O_{j1} and O_{j2}.
- (Order constraint) The operations strictly follow the order constraint. That is, O_{j2} ($j = 1, 2, 3$) cannot be processed until O_{j1} has been completely processed.
- (Arrival time) Each job has an arrival time (ArrivalTime). For each job J_j, the first operation O_{j1} cannot be processed earlier than its arrival time.
- (Resource constraint) Each operation can only be process by a particular machine. For example, operation O_{11} can only be processed by machine M_1. Each machine can process at most one operation at a time.

Solution/Schedule Representation

A solution/schedule for a job shop scheduling problem is a sequence of actions. Each action is composed of the processed operation, the machine to process the operation, and the starting time. The finishing time of an action is the starting time plus the processing time of the processed operation. The actions are sorted in the increasing order of starting time, i.e. the former action starts no later than the latter one. In this assignment, the following format is adopted to represent a schedule:

$$Process(O_{11}, M_1, 0) \rightarrow Process(O_{21}, M_2, 10) \rightarrow \ldots,$$

where $Process(o, m, t)$ stands for an action that processes the operation o with machine m and starts at time t.

Questions

1. (10 marks) Given a schedule whose action sequence is as follows: $Process(O_{11}, M_1, t_1) \rightarrow Process(O_{21}, M_2, t_2) \rightarrow Process(O_{31}, M_1, t_3) \rightarrow Process(O_{12}, M_2, t_4) \rightarrow Process(O_{22}, M_1, t_5) \rightarrow Process(O_{32}, M_2, t_6)$. Since the sequence is sorted in the non-decreasing order of starting time, we know that $t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5 \leq t_6$. Calculate the **earliest starting time** (t_1 to t_6) of each action. You can draw a gantt chart to help you think.

   Hint: the earliest starting time of an action is the later time between the earliest ready time of the operation and the earliest idle time of the machine.

2. (10 marks) 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.

3. (10 marks) Write the **final solution** obtained by the **Shortest Processing time (SPT)** dispatching rule. You may draw a figure (gantt chart) to help you find the solution.

4. (5 marks) 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.

   Note: the solution in Question 1 is obtained by the First-Come-First-Serve (FCFS) rule.

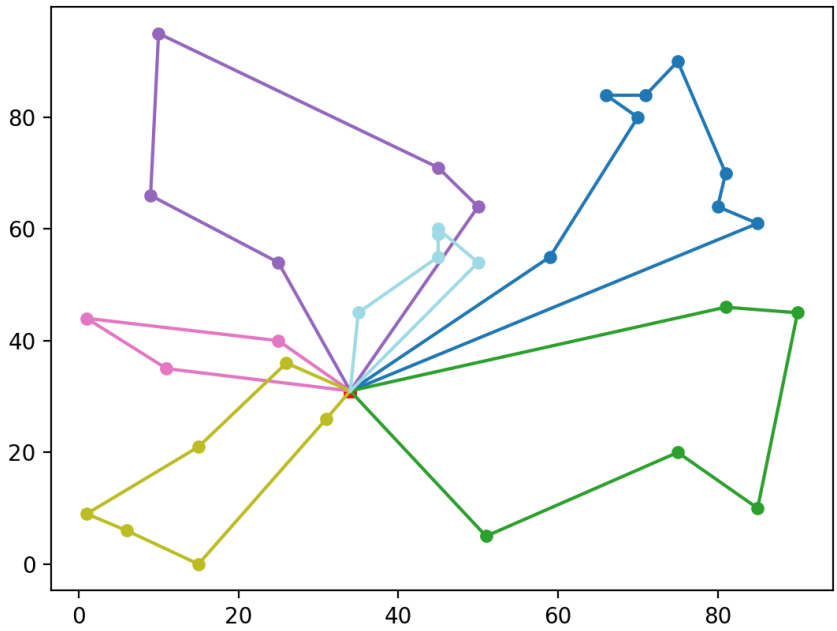
5. (5 marks) The two compared solutions are obtained by the SPT and FCFS rules, respectively. If one solution is better then the other, does it mean that the rule that generates the better solution is better than the other rule? Why or why not?

6. **(for AIML420 ONLY, 10 marks)** Often in practice, neither the SPT nor FCFS rules are good enough for solving the job shop scheduling problem. Suggest two methods to solve the job shop scheduling problem. One method should consider the problem to be static (all information is known in advance), and the other method should consider it to be dynamic (e.g. unpredicted job arrivals can happen in real time). **Clearly describe your methods (e.g. algorithm framework, solution representation/encoding, search operators).**

Part 2: Vehicle Routing [60 marks]

Every day, a delivery company needs to deliver goods to many different customers. The deliveries are achieved by dispatching a fleet of vehicles from a centralised storage warehouse. The goal of this problem is to design a route for each vehicle so that all of the customers are served by exactly one vehicle and the travel distance of the vehicles is minimised. Additional problem complexity comes from the fact that the vehicles have a fixed storage capacity and the customers have different demands.

Below is an example of the Vehicle Routing Problem (VRP). The red box indicates the central warehouse (*depot*), and each circle is a customer distributed in the 2D space. There are 6 vehicles routes, each with a different colour. Each route starts from the depot, serves a subset of customers, and returns to the depot again. Each route satisfies the capacity constraint, i.e. the total demand of the customers served by the route does not exceed its capacity.



Data

In the provided `vrp-data.zip`, there are two VRP instances, along with their corresponding optimal solutions. The instance is named with `n(n)-k(k).vrp`, where n is the number of customers and k is the number of vehicles. The corresponding solution is named as `.sol`.

The instance file is basically self-explained.

- It starts with some information rows including "NAME", "COMMENT", "TYPE", "DIMENSION", "EDGE_WEIGHT_TYPE", which you can ignore/skip safely.
- The row "CAPACITY" gives the capacity of the vehicles.
- Under "NODE_COORD_SECTION", each row shows the coordinates of each node (customer), in the format of "index x-coordinate y-coordinate".
- Under "DEMAND_SECTION", each row shows the demand of each node, in the format of "index demand".
- Under "DEPOT_SECTION", the first row is the index of the depot, and the second is -1, indicating the end of the file.

To help you code, you are provided two template codes, one in Python and the other in Java. You can use either of them to work on this part. Of course, you can also write your own code from scratch if you like.

Python Template Code

In the provided `python.zip`, there are three Python template code file to help you load the `.vrp` instance file, read a `.sol` solution file, and visualise a solution.

- The `loader.py` file contains the following functions:
  - `load_data()`: read a VRP instance from a `.vrp` file.
  - `load_solution()`: read a VRP solution from a `.sol` file.
- The `utility.py` file contains the following functions:
  - `calculate_euclidean_distance()`: calculate the Euclidean distances between two nodes. This is a **TODO** item if you use this template code to do the assignment.
  - `calculate_total_distance()`: calculate the total Euclidean distance of a solution. This is a **TODO** item if you use this template code to do the assignment.
  - `visualise_solution()`: visualise the solution on a 2D figure.
- The `main.py` file contains the following functions:
  - `main()`: the main function with three examples that read the `.vrp` file, and then (1) read its optimal solution from the `.sol` file, and visualise the optimal solution; (2) construct a solution using the *nearest neighbour heuristic* and visualise it; (3) construct a solution using the *savings heuristic* and visualise it.
  - `nearest_neighbour_heuristic()`: construct a solution using the *nearest neighbour heuristic*. This is a **TODO** item if you use this template code to do the assignment.
  - `savings_heuristic()`: construct a solution using the *savings heuristic*. This is a **TODO** item if you use this template code to do the assignment.

The template code relies on the `numpy` and `matplotlib` Python libraries. You should install these libraries if you haven't.

- Installing `numpy`: `https://numpy.org/install/`.
- Installing `matplotlib`: `https://matplotlib.org/stable/users/installing.html`

Java Template Code

In the provided `java.zip`, there are the following `.java` template Java files.

- The `VRPInstance.java` file is the data structure for a VRP instance.
- The `VRPSolution.java` file is the data structure for a VRP solution.
- The `VRPNode.java` file is the data structure for a node.
- The `VRPIO.java` file is the I/O file containing the following functions:
  - `loadInstance()`: read a VRP instance from a `.vrp` file.
  - `loadSolution()`: read a VRP solution from a `.sol` file.
  - `writeSolution()`: write a VRP solution into a `.sol` file.
- The `main.java` file contains a `main()` function. It reads a `.vrp` instance file. Then, it generates a `nnSol` solution by the *nearest neighbour heuristic*, a `svSol` solution by the *savings heuristic*. Finally, it writes the `nnSol` into the `nn.sol` file, and `svSol` into the `sv.sol` file.

  - **NB: You can use the `main.py` Python file to read the obtained `.sol` solution files and visualise them.**

- The `Utility.java` file contains the following functions:
  - `calculateEuclideanDistance()`: calculate the Euclidean distances between two nodes. This is a **TODO** item if you use this template code to do the assignment.
  - `calculateTotalCost()`: calculate the total cost (Euclidean distance) of a solution. This is a **TODO** item if you use this template code to do the assignment.
  - `nearestNeighbourHeuristic()`: construct a solution using the *nearest neighbour heuristic*. This is a **TODO** item if you use this template code to do the assignment.
  - `savingsHeuristic()`: construct a solution using the *savings heuristic*. This is a **TODO** item if you use this template code to do the assignment.

Implementation

- A straightforward way is to fill in the following **TODO** items in the template code.
  - For the **Python template code**, they are in
    - The `main()`, `nearest_neighbour_heuristic()` and `savings_heuristic()` functions in `main.py`.
    - The `calculate_euclidean_distance()` and `calculate_total_distance()` functions in `utility.py`.
  - For the **Java template code**, they are in the `Utility.java` file.
    - After obtaining the `.sol` files, complete the `main()` function in `main.py` to read the `.vrp` file and the `.sol` files generated by your code, and visualise them.
    - This can simply be done by calling the `loader.load_solution()` and `utility.visualise_solution()` functions.
- If you do not want to use the template code, you can
  - use any other language to (1) read the `.vrp` file, (2) generate the solution using the required heuristics, and (3) output the solution in the same format as in the given `.sol` files.
  - complete the `main()` function in `main.py` to read the `.vrp` file and the `.sol` files generated by your code, and visualise them.
  - write a clear `readme` file to run your source code on an **ECS Desktop** to generate the `.sol` files.

Requirement

1. (20 marks) Implement the *nearest neighbour heuristic* to generate a VRP solution. The distance between two nodes is defined as the Euclidean distance. That is, given two nodes $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$, the distance is calculated as
$$dist(v_1, v_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
2. (20 marks) Implement the *savings heuristic* to generate another VRP solution.
3. (20 marks) In the report, use the Python template code to visualise the solutions obtained by the heuristics, compare the solutions generated by the two heuristics and the optimal solution and discuss the differences between their performance.
4. (**for AIML420 ONLY, 10 marks**), Suggest a more advanced method that can be better than the above two heuristics. **Clearly describe your methods (e.g. algorithm framework, solution representation/encoding, search operators).**

2 Notes

During the time between the assignment handout and submission, the tutor(s) will run a number of helpdesks to provide assistance.

3 Submission Guidelines

3.1 Submission Requirements

1. Programs (**Executive program file and source files**) for Part 2. Please provide a `readme file` that specifies how to compile and run your program. A script file called `sampleoutput.txt` should also be provided to show how your program run properly. If you programs cannot run properly, you should provide a `buglist` file.

2. A report document that consist of **the answers of all the individual parts**. The document should mark each part clearly. The document can be written in PDF, text or the DOC format.

3.2 Submission Method

The programs and the PDF report should be submitted through the web submission system (accessible from the COMP307 or AIML420 course web site) **by the due time**. Please ensure you submit to the correct system based on which course you are enrolled in!

3.3 Late Penalties

The assignment must be submitted on time unless you have made a prior arrangement with the **course coordinator** or have a valid medical excuse (for minor illnesses it is sufficient to discuss this with the course co-ordinator.) The penalty for assignments that are handed in late without prior arrangement is one grade reduction per day. Assignments that are more than one week late will not be marked.

Remember that you have three late days for this course (which can be used fractionally), but that these apply across the whole course, not per-assignment! Please save some late days in case you need them later on!