COMP307/AIML420 — *Fundamentals of AI*

Assignment 4: Planning and Scheduling

9% of Final Mark — Due: 23:59 Wednesday 29 May 2024

1 Objectives

The goal of this assignment is to help you understand the basic concepts of planning and scheduling, and simple algorithms for tackling these problems. 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 focuses mainly on report writing. Additionally, you will need to write some programs for solving the vehicle routing problem in Part 2.

IMPORTANT: You must not use external libraries (e.g., PyTorch, sklearn, or any others) or any AI Tools (e.g., ChatGPT, CoPilot) to complete this assignment.

2 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} of job J_j 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/completion time of an action equals to the starting time plus the processing time of the processed operation. The actions are sorted in the increasing order of their 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:

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

where $Process(o, m, t)$ stands for an action that processes operation o on machine m, with the starting 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 the 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 in Question 4 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? Provide a short explanation of your answer.

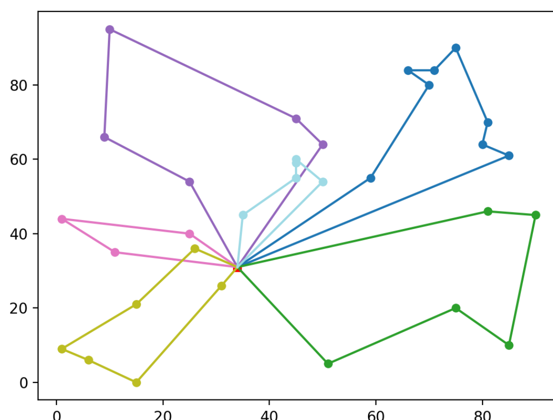
6. **(for AIML420 ONLY, 10 marks)** Often in practice, neither the SPT nor FCFS rules are good enough for solving the job shop scheduling problem. Suggest two methods to solve the job shop scheduling problem. One method is suitable for solving **static** problems (all information for scheduling is known in advance). The other method is suitable for solving **dynamic** problems (e.g., unpredicted job arrivals can happen in real time). **Clearly describe your methods (e.g., algorithm framework, solution representation, search operators).** Support your description with pseudo-code or flowchart diagrams.

NOTE: No programming is required to answer the above questions.

3 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 total travel distance of all vehicles is minimised. Additional problem complexity comes from the fact that each vehicle has a fixed storage capacity and the customers have varied demands.

Below is an example of the *Vehicle Routing Problem* (VRP). The red box indicates the central warehouse (*depot*). Each circle stands for a customer. All customers are distributed in the 2D plane. 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 must satisfy the capacity constraint of a vehicle, i.e., the total demand of the customers served by the route does not exceed the vehicle's 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 code templates, one in Python and the other in Java. You can use either of them to work on this part (recommended). You can also write your own code from scratch, without using any of the two code templates (not recommended).

Python Code Template

In the provided `python.zip`, you can find three Python code template files 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 code template in your assignment.
  - `calculate_total_distance()`: calculate the total Euclidean distance of a solution. This is a **TODO** item if you use this code template in your assignment.
  - `visualise_solution()`: visualise the solution on a 2D figure.
    NOTE: The visualisation function assumes that in the solution, **the depot index is 0 rather than 1**. Therefore, you must **reduce all the node IDs by 1 for visualisation**. For example, if the original route is [2, 5, 4, 3], the corresponding input `vrp_sol` should be [1, 4, 3, 2].
- 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 code template in your assignment.
  - `savings_heuristic()`: construct a solution using the *savings heuristic*. This is a **TODO** item if you use this code template in your 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 Code Template

In the provided `java.zip`, you can find the following `.java` Java template files.

- The `VRPInstance.java` file provides the data structure for a VRP instance.
- The `VRPSolution.java` file provides the data structure for a VRP solution.
- The `VRPNode.java` file provides the data structure for a node.
- The `VRPIO.java` file contains the following I/O related 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 using the *nearest neighbour heuristic*, and subsequently a `svSol` solution by using the *savings heuristic*. Finally, it writes the `nnSol` into the `nn.sol` file, and the `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 code template for your assignment.
  - `calculateTotalCost()`: calculate the total cost (Euclidean distance) of a solution. This is a **TODO** item if you use this code template for your assignment.
  - `nearestNeighbourHeuristic()`: construct a solution using the *nearest neighbour heuristic*. This is a **TODO** item if you use this code template for your assignment.
  - `savingsHeuristic()`: construct a solution using the *savings heuristic*. This is a **TODO** item if you use this code template for your assignment.

Implementation

- A recommended way is to fill in the following **TODO** items in the code template provided.
  - For the **Python code template**, 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 code template**, 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 code template provided (not recommended), you can
  - use any other programming languages 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 explain how to run your source code on an **ECS Desktop** computer 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 a VRP solution.
3. (20 marks) In the report, use the Python code template to visualise the solutions obtained by the heuristics, compare the solutions generated by the two heuristics and the optimal solution. Discuss the differences among these solutions in terms of their total distances.
4. (**for AIML420 ONLY, 10 marks**) Suggest a more advanced method that can potentially perform better than the the *nearest neighbour heuristic* and the *savings heuristic*. **Clearly describe your methods (e.g., algorithm framework, solution representation, search operators).** Support your description with pseudo-code or flowchart diagrams.

4 Submission Guidelines

4.1 Assessment

We will endeavour to mark your work and return it to you as soon as possible, hopefully within 2 weeks after the submission deadline. The tutors will run a number of help desks to provide guidance (but won't tell you the answers!).

4.2 Submission Requirements

1. Programs (**Executive program files 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 programs can run properly. If you programs cannot run properly, you should provide a `buglist` file.

2. A `report` document that consists of **the answers of all the individual parts**. The document should mark each part clearly. Only PDF format is acceptable for the report document. Reports in other document format (e.g., docx, txt) will not be marked.

4.3 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 date and time**. Please ensure you submit to the correct system based on which course you are enrolled in.

Please check **again** that your programs can be run on the ECS machines easily according to your readme. If tutors cannot run your code, you may **lose marks!** Each tutor has a limited amount of time ($\leq$ 5 minutes) to get your code running, so please don't ask them to use Pycharm, IntelliJ IDEA, Visual Studio, etc. to run your code. All these IDEs support exporting runnable code.

4.4 Late Penalties

The assignment must be submitted on time unless you have made a prior arrangement with the course **co-ordinator** or have a valid medical excuse. We use the ECS extension system for all extension requests. Please make an extension request through the ECS system if you think you have a valid reason.

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.

4.5 Plagiarism

Plagiarism in programming (copying someone else's code) is just as serious as written plagiarism, and is treated accordingly. Make sure you explicitly write down where you got code from (and how much of it) if you use any other resources asides from the course material. Relying heavily on borrowed code may lead to deductions in marks, whereas plagiarism warrants a zero mark and may entail disciplinary actions.