Singapore Management University

School of Computing and Information Systems

2024/2025 Semester 2

CS606: AI Planning and Decision Making

Assignment 2 **Main Question** (Due 16 March 2025 1159PM)

Maximum marks = 30 (+ 5 marks bonus refer to optional Challenge Question)

**Submission Instructions**

You are to submit a zipped folder to eLearn (in the Assignment folder), comprising:

(1) a Word or PDF document containing the typed answer for Question (a);

(2) the folder code that contains your revised python codes (namely, psp.py, operators.py and alns\_main.py) for Question (b). **Note that we will test your code by the following command line, so make sure your code can be executed as:**

python alns\_main.py <instance> <random\_seed>

To ease handling, please name your file YourName.zip. Multiple submissions are permitted up to the due time, but only the last submission will be saved and graded.

**Problem Statement**

You operate an online platform that matches tasks with workers registered with the platoform. In this assignment, we will explore the application of ALNS to solve the Platform Worker Scheduling Problem (PSP), defined as follows.

Input:

* A planning horizon *T* days, each day is divided into hourly timeslots 0,…, 23 (representing 12 midnight to 11pm on the same day)
* A set of *m* hourly tasks to be performed, each task is associated with a timeslot (between 0 to 23), and the skill required
* A set of *n* workers, each worker is associated with his/her available timeslots, a set of skills possessed, and an hourly rate

Output: A schedule which assigns tasks to workers in the entire horizon *T* subject to the following constraints:

1. Each task is assigned to at most one worker.
2. Each worker is assigned at most one task at the same time (i.e. no multi-tasking).
3. *Eligibility constraint:* A task may only be assigned to a worker if the start time is in the worker’s available timeslots, and the skill required can be matched by the worker’s skill set.
4. *Block constraint:* A worker is turned out to perform tasks in one or more time blocks. A block is defined as a period of consecutive time slots (e.g. Mon 9am-12noon). Each worker is allocated at most one block per day, and the maximum length of each block is *BMax* hours (e.g. *BMax*=12).
5. For each worker, the allocated blocks over the entire planning period must satisfy the following labor requirements:
   1. *Maximum work constraint*: The sum of the block lengths must not exceed *WMax* hours (e.g. *WMax*=72).
   2. *Rest constraint*: Between two blocks, there must be a rest period of at least *RMin* hours (e.g. *RMin*=12). In other words, the duration between the last assigned task of the current day and the first assigned task of the next day must be at least *RMin* hours (for simplicity, assume *RMin*≤24).

Objective function: The goal is to find a schedule that minimizs the cost function *f*1+ *f*2, where:

1. *f*1: Total cost of unassigned tasks, calculated as *a* timesthe number of unassigned tasks. (You can think of *a* as the monetary penalty cost for each unassigned task)
2. *f*2: Total cost of workers, calculated as follows:

For each worker, the pay for each allocated block is the block length multiplied by the worker’s hourly rate, subject to a minimum value of **$50**. The rationale is that each time a worker turns out for work, he/she should be compensated with a base payment. The total cost of workers is the sum over the pay of allocated blocks.

**Questions**

(a) (10 marks) Write a report that explains your ALNS design for PSP. The report should contain the construction heuristic, destroy methods, repair methods, and the weights adjustment strategy.

(b) (20 marks) Implement your ALNS algorithm presented in (a) in Python based on the code template provided (see **Annex A** below for details). You should test your code against the set of problem instances given in the folder psp\_instances. See **Annex B** below for details.

**References:**

1. Cordeau, Jean-François; Laporte, Gilbert; Pasin, Federico; Ropke, Stefan. Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling*, 2010, Vol.13 (4), p.393-409.
2. Woller, David; Kulich, Miroslav. The ALNS Metaheuristic for the Maintenance Scheduling Problem. Proceedings of the 18th *International Conference on Informatics in Control, Automation and Robotics* (ICINCO), 156–164, 2021.

**Marking Criteria**

(a): ALNS Algorithm Design (10 marks)

Mark allocation:

(3 marks) construction heuristic,

(3 marks) destroy methods,

(3 marks) repair methods, and

(1 mark) weights adjustment strategy.

(b): Implementation (20 marks)

Your code will be graded based on the quality of solutions and run time efficiency, which are measured by running against hidden test instances not given in psp\_instances. The details of the input and expected output format are given in **Annex B** in .

For grading purposes, we will only be running your code on instances **with at most 50 workers and 500 tasks for a planning horizon of 5 days**, and that your code will be executed with **1000 ALNS iterations**. For each instance, we expect improvement on the objective value of the solution over the initial solution obtained by your construction heuristic; and the run time is expected to be less than 5 minutes on the assigned server (see below).

Mark allocation:

* + 1. You will get 5 marks if your code produces feasible solutions after 10000 ALNS iterations within 5 minutes on the assigned server for the hidden dataset, and 0 mark if your code failed to produce feasible solutions.
    2. You will get an additional 5 marks if the objective value of your final solution for the given dataset S2.json is strictly better than our baseline (which is the solution obtained by our construction heuristic).
    3. The remaining 10 marks will be awarded on a curve comparing your results with the results obtained by your classmates on both the objective values obtained and run time on both the given dataset S2.json and hidden dataset.

Running on Server:

1. Server user guide and the template sh file are uploaded under eLearn -> Content -> Course Syllabus/Outline.
2. The server account will be sent by email from Jean, and it is to be shared within your project group.

**\*Please do not copy others’ work. You will get warning letter from MITB office and 0 mark for this assignment if we detect any sign of copying from your submissions.**

**Annex A:** **Explanation of Vanilla ALNS Code Template**

3 skeleton python files are provided in the code folder, along with the alns package[[1]](#footnote-1) (in src/) which you **do not need to modify.** The following is a brief explanation of the skeleton codes. Refer detailed description in the comments in the python files. The places that will need to be modified will include a // within the comment (e.g. # // Modify with your name).

**1. src/helper.py** – contains helper functions that will be used to save and generate an output.txt file containing the solutions after conducting ALNS iterations.

**2. alns\_main\_skeleton.py** – main program. Change the file name to alns\_main.py

**Function imports:**

* destroy\_1 and repair\_1 – import your destroy and repair functions that have been defined in operators.py (refer to **3. operators\_skeleton.py**)

**Lines to be modified:**

* save\_output('YourName\_ALNS', psp, 'initial')

– change 'YourName' to your full name

* save\_output('YourName\_ALNS', solution, 'solution')

– change 'YourName' to your full name

* alns.add\_destroy\_operator(destroy\_1)

– change destroy\_1 to your destroy methods. You can add more lines if you wish to include more destroy methods

* alns.add\_repair\_operator(repair\_1)

– change repair\_1 to your repair methods. You can add more lines if you wish to include more repair methods

* weights adjustment strategy

Text

Description automatically generated

– fill in the values for:

omegas = [...] a list of 4 scores for []

lambda\_ = ... a number between 0 and 1, which is the decay parameter that controls how sensitive the weights are to changes in the performance of the destroy and repair methods

**3. psp\_skeleton.py** – builds solution state for PSP. Change the file name to psp.py (to be used by alns\_main.py)

**Classes**

* Parser – gets the data from the json file and reformat it into correct class **(do not modify)**
* Worker – decides task assignments for each worker
  + can\_assign: takes in the task and returns True or False representing whether this task can be assigned to the worker. **This is the function for you to implement.**
  + assign\_task: takes in the task and assigns the task to the worker, updates the attributes tasks\_assigned, blocks, total\_hours and no return. **This is the function for you to implement.**
  + remove\_task: takes in the task and tries to remove the task from the worker, updates the attributes tasks\_assigned, blocks, total\_hours, return True or False representing whether the task can be removed or not. **This is the function for you to implement.**
  + **Do not modify the existing functions in this class.**
* Task – records the required skillset and the timeslot for each task. **(do not modify)**
* PSP – the solution state.
  + random\_initialize: implement your construction heuristic in this function to produce an initial solution. **This is the function for you to implement.**

**4. operators\_skeleton.py** – contains your **destroy and repair operators**. Change the file name to operators.py to be used by alns\_main.py **(**as well as pspAlnsEnv.py in the Challenge Question)

**Running the program:**

python alns\_main.py <instance> <random\_seed>

**Annex B**: **Explanation of Dataset and Output**

The data set is found in the sample\_instances folder, which contains test instances in the following json format.

<Basic Info>

- name of the dataset (name)

- planning horizon (T)

- penalty cost for each unassigned task (Alpha)

- maximum length of each block (BMax)

- maximum sum of the block lengths (WMax)

- minimum rest period between two blocks (RMin)

<Workers>

- a list of dictionaries

- each dictionary has four keys, the worker id (w\_id), the skill set (skills), the available slots (available) and the hourly rate (rate) for that worker

- the value for "w\_id" is an integer

- the value for "skills" is a list containing strings

- the value for "available" is a dictionary where key is the day index and value is a list of available hours (integer)for that day

- the value for "rate" is an integer

<Tasks>

- a list of dictionaries

- each dictionary has four keys, the task id (t\_id), the required skill set (skill), the required day (day) and hour (hour) for that task

- the values for "t\_id", "day" and "hour" are integers

- the value for "skill" is a string

**Sample instances and output:**

There are 3 sample instances provided for your testing in the sample\_instances folder, with the respective results in the sample\_output folder.

Note: The train folder is for the Challenge question which is optional.

For each sample instance, we provide 2 corresponding sample output files:

1. The file ending with inital.txt – initial solution produced by construction heuristic
2. The file ending with solution.txt – solution after running 1000 ALNS iterations

**Format of output txt files:**

Each file contains the objective value and the task assignments for the given instance.

The first row shows the objective value, and the tasks that are unassigned.

From the second row onwards, it shows the allocated block for each worker and each specific day and the tasks assigned to the worker. Note that if a worker is not allocated a block for a particular day, there will be no output for the worker. For example:

|  |
| --- |
| Objective: 5207, Unassigned: [14, 104]  Worker 0: Day 0 Hours [13, 22] Tasks [55, 64, 79, 105, 117]  Worker 4: Day 1 Hours [5, 8] Tasks [124, 128, 133]  Worker 4: Day 2 Hours [4, 7] Tasks [232]  : |

Note that you do not need to write additional code to generate output files, since the code is already in the skeleton code described in Annex A.

When you run your code against the test instances, the output files are prefixed by your name and instance’s name .

i.e. YourName\_ALNS\_S1\_initial.txt, YourName\_ALNS\_S1\_solution.txt

You may then compare the objective values with the sample outputs provided, so as to gauge the quality of the solutions produced by your code. Note that there is no need for you to submit these files, since we will be running your code and evaluating it based on the output files generated from our end.

1. This package is based on the code available in the GitHub repository <https://github.com/N-Wouda/ALNS>

   Note that the current GitHub version has a lot more features which are not needed for our assignment. [↑](#footnote-ref-1)