Operations Research, Spring 2022 (110-2)
Case Assignment 2

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

1 Submission rules

- This case assignment is due at ***23:59, May 7***. Those who submit their works late but are late by less than one hour gets 10 points off. Works that are late by more than one hour get no point.
- For this case assignment, students should work in teams. Each team should contain ***either four or five*** students. Teams that do not follow this rule gets no point.
- For each team, **one and exactly one** student should submit their work on behalf of all team members. Please submit a ***ZIP file*** containing a ***PDF file*** and a ***Python program*** (i.e., a .py file) through NTU COOL. Make sure that the submitted PDF file contains the student IDs and names. The Python program should follow the Python 3.9 standard. Those who fail to do these will get 10 points off.
- You are required to ***type*** your work with LaTeX (strongly suggested) or a text processor with a formula editor. Hand-written works are not accepted. You are responsible to make your work professional in mathematical writing by following at least those specified in homework assignments. Those who fail to follow these rules may get at most 10 points off.
- The maximum length of the report is ***eight pages***, including everything. Everything starting from page 9 will not be graded.

2 Overview of the tasks

Recall the IEDO company doing food manufacturing introduced in Case Assignment 1. In that assignment, you formulated integer programs that may find good schedules with

a small number of tardy jobs and low makespan through commercial solvers like Gurobi Optimizer.

While that was good, too large instances cannot be solved by integer programming, and a heuristic algorithm is needed in this case. Therefore, in this assignment you are invited (actually forced) to design and implement a heuristic algorithm for the scheduling problem. Your implementation will be graded based on its performance (i.e., objective values), efficiency (i.e., execution time), and design (depending on how you convince the TAs that your algorithm should be good).

Before we describe your tasks in this case assignment, let's first introduce the process of *abstraction* for you.

3 Abstraction

Along with this document, five CSV files are given to you. Each CSV file represents an instance, where three of them represent the three instances given in Case Assignment 1. Once you take a look at any of the CSV files, you will realize that the given instances are *abstraction* of the original concrete instances: Details that are irrelevant to the solution processes are ignored. Let's take the example listed in Case Assignment 1 (which is not those instances in any of the given CSV files) as an example. While the concrete information of that instance was provided in Tables 1 and 2 in that assignment, now it is condensed into Table 1 in this assignment. Each instance in an CSV file follows the same six-column format.

The original concrete instances are condensed into the six-column format in the following way:

- In the new format, each row (except the first one) contains the information of a job. In particular, the column "Job ID" records the IDs of jobs. All values in this column are positive integers in $[1, 1000]$.
- We now do not care whether the factory opens at 7:30, 8:00, or 11:00. We will label the opening time as time 0. Each job has a due time, which is now labeled as the number of hours (may be fractional in general) after the opening time. For example, in the above example instance, the (abstract) due time of job 1 is recorded as 5 because its (concrete) due time is 12:30 and the factory opening time is 7:30.

Job ID	Stage-1 P.T.	Stage-2 P.T.	Stage-1 M.	Stage-1 M.	Due Time
1	3.7	0.5	2,3,4,5	2,3,4,5	5
2	1.6	2.3	1,2,3,4,5	2,3,4,5	5
3	1	2.7	1,2,3,4,5	2,3,4,5	5
4	2.2	1.4	2,3,4,5	2,3,4,5	5
5	1.8	0.9	2,3,4,5	1,2,3,4,5	5
6	2.7	0	1,2,3,4,5	N/A	5
7	1.4	1.5	2,3,4,5	2,3,4,5	10
8	1.1	1.1	2,3,4,5	2,3,4,5	10
9	1.8	0.8	2,3,4,5	1,2,3,4,5	10
10	1.5	2.9	2,3,4,5	2,3,4,5	10
11	2.5	1.5	2,3,4,5	1,2,3,4,5	10
12	2.3	0	1,2,3,4,5	N/A	10

Table 1: An example instance (P.T. means processing time, and M. means machines)

The (abstract) due times are recorded in the column "Due Time". All values in this column are positive real numbers in $(0, 24]$.

- We now do not care whether a job has one process, two processes, or ten processes. As each job may be split at most once, we only care about whether it has one stage or two stages. For example, jobs 6 and 12 have only one stages because they cannot be split. For a job that cannot be split, we define its (abstract) stage-1 processing time as its (concrete) total processing time and its (abstract) stage-2 processing time as 0. On the contrary, the other ten jobs all have two stages because they may be split once. For a job that can be split, we define its (abstract) stage-1 processing time as the sum of (concrete) processing times before the splitting time and its (abstract) stage-2 processing time as the sum of the remaining (concrete) processing times. As an example, the (abstract) stage-1 processing time of job 4 is $0.5 + 0.7 + 1 = 2.2$, and that of stage 2 is $0.6 + 0.3 + 0.5 = 1.4$.

  The (abstract) processing times are recorded in the columns "Stage-1 Processing Time" and "Stage-2 Processing Time". All values in these two columns are real numbers in $[0, 10]$.

- Recall that in Case Assignment 1, machine 1 can only do boiling while machines 2 to 5 may do all three types of processes. We now do not care about any concrete process type. We also do not care whether it is machine 1 or machines 1 and 2 that cannot do a specific type of process. All we need, from the perspective of

solving the problem, is to know for each stage of each job the list of machines that may process that stage. Let's now take job 2 as an example. Recall that job 2's first stage contains only one process, which is boiling, and its second stage contains two processes, baking and boiling. Therefore, its first stage may be processed by machines 1, 2, 3, 4, and 5, but its second stage may only be processed by machines 2, 3, 4, and 5.

The lists of allowable machines are recorded in the columns "Stage-1 Machines" and "Stage-2 Machines". There are two possible types of values in these two columns. If a job does not have stage 2, its value in the "Stage-2 Machines" column will be "N/A" (without quotation marks). Otherwise, a list of machines will be a string of comma-separated numbers inside a pair of double quotation marks, where numbers are all valid machine IDs, no two numbers are identical, and no white space exists.

Abstraction is important in problem solving. In many cases, different problems become the same one after abstraction, and designing a method that solve an abstract problem actually solves all those concrete problems.[1] Writing compact formulations of mathematical programming is one great way of abstraction. In the sequel, we will introduce your tasks according to the abstract problem described above.

4 Tasks

Problem 1

(45 points) Please implement your algorithm in Python 3.9 so that the instructing team may grade it with fifteen testing instances, one in an CSV file. Five of these CSV files have been provided to you for your reference. Please do not forget that the objective values of the optimal solutions to three of the instances have been provided to you in Case Assignment 1. You will not see the ten hidden CSV files until this assignment is due. However, you may assume that they follow the same format described above.

Please note that in these fifteen instances, the number of machines may not be 5, functionality of these machines may vary a lot, there may be more than two timings for due times, etc. In short, you are now dealing with a generalization of the problem described in Case Assignment 1. There is a saying "abstraction is the foundation for

[1] This is also called "reduction" computer science.

generalization." Please try to think about this.

Along with this document and the five CSV files, you are also provided two PY files: algorithm_module.py and CA2_grading_program.py. You should implement your algorithm in algorithm_module.py. DO NOT change the function name and file name. Moreover, please make sure that this function returns a schedule specified in two two-dimensional lists `machine` and `completion_time`. See the comments in the two PY files for more information regarding the two lists. The instructing team will use CA2_grading_program.py to invoke the `heuristic_algorithm` funcion in the file algorithm_module.py you submit. Therefore, please ensure that your function may be invoked correctly. You may check out some more instructions in these two PY files.

One team may earn at most 3 points for each of the fifteen instances. First, one team earns 1.5 points if the generated schedule is feasible (no two stages overlap with each other on a single machine, a stage is not processed by a machine that cannot process that stage, etc.) and its execution time is no longer than three minutes. If the schedule is infeasible or the execution time is longer than three minutes, the team earns 0 points in this category. Please note that your implementation will be executed on a TA's device. The device will not be a very slow one, but to make sure that your program may return a solution in three minutes, you may want to have some buffer on your own device.

Another 1.5 points are based on performance. One team earns all or part of the other 1.5 points according to her/his ranking among all teams. Let r_i be the ranking of team i, where $r_i = 1$ means team i's schedule is the best, team i earns

$$\frac{3}{2}\left(1 - \frac{r_i - 1}{N - 1}\right)$$

points, where N is the number of teams getting a feasible schedule for this instance. Two scheduled are compared according to the rule specified in Case Assignment 1. In particular, we favor a schedule with fewer tardy jobs. If the number of tardy jobs are the same, we favor the one with lower makespan. The rankings for different instances are independent. For example, one team may be ranked 1 in instance 1 but ranked 5 in instance 2.

Important note. There is no restriction on the design of your heuristic algorithm. However, we require you to implement your heuristic algorithm in Python 3.9. We understand that it is better if there is no restriction on the programming language. However, as we need to specify the input/output to give you a precise development instruction, facilitate grading, and use the same basis to time all submissions, we are sorry that we must make such a restriction.

Problem 2

(25 points) Use your own words to introduce your heuristic algorithm to the instructing team. You may write paragraphs of words, draw flow charts, present pseudocodes, and/or provide examples. There should also be a basic big-O analysis regarding the time complexity of your algorithm. Points will be given to you according to the logic of your algorithm and clearness of your description. Please try your best to convince the instructing team that your algorithm design is intuitively good.

Problem 3

(30 points) Recall that in Homework 2 we demonstrated how to conduct numerical experiments to test the performance of a heuristic algorithm. Basically, we generate random instances, let the algorithm solves these random instances, and compare the objective values generate by our algorithm with some benchmark. In this problem, please conduct your own experiments to convince the instructing team that your algorithm performs well.

You may review Homework 2 regarding how to set up an experiment. In Homework 2, you may find an example of setting up probability distributions for parameter values. Ideally, your random instances should cover a wide range of situations. However, if you find that your algorithm cannot deal with some weird cases, it is fine to narrow down the ranges of your parameter values as long as you clearly specify the experiment setting. Do not test your algorithm with just one or two instances. Try to test it with hundreds or thousands of instances.

Do not forget to find a benchmark for your algorithm. One typical candidate is your integer programming formulation. As long as your instances are not too big, you may use a solver to generate an optimal solution and compare your heuristic solution with the optimal one. Another typical candidate is the relaxation of your integer program. You may take away the integer constraint or some other constraints to find an upper bound or lower bound of the objective value of an optimal solution. It is also common for one to compare the proposed algorithm with a very simple (or even stupid) one. After all, if the proposed algorithm cannot outperform a simple one, it cannot be good.

Finally, keep this in mind: Designing an algorithm is not hard; making it convincing is. Please do everything you may do to make your proposed algorithm convincing.