MIE562F 2020 Coding Assignment #2

Total Marks: 10

All code must be written by you alone. It is an academic offense to submit code written by someone else and to write code that someone else submits.

Late submissions are not accepted.

The assignment is to write Python 3 code to do the following:

- read in a job shop scheduling problem file (with a specified file format – see below)
- create a *disjunctive* MILP model for the problem and solve it using the Gurobi Python interface
- return the makespan and solution found

We have done our best to answer all questions in this document. Please read it carefully. If you have a question that is not covered here, please post it to piazza.

1 Problem Definition Details

You can find definitions of the job shop scheduling problem and the disjunctive MILP model in the course slides and/or textbook. In this section, we provide details on how we represent the problem instances and the conventions used.

The instance file format is identical to that used for Coding Assignment #1.

Refer to the Coding Assignment #1 for the details.

The instances that are part of the solution to Coding Assignment #1 (see Quercus) are all valid instances for this assignment – however, the solutions to those instances for Coding Assignment #1 are not necessarily correct solutions to this assignment.[1] These are not the test instances that will be used on MarkUs so it is recommended that you also develop your own instances to fully test your code.

2 Coding and Marking Details

You may re-use code **that you wrote for Coding Assignment #1** for this assignment. In particular, because the instance format is the same, the code for reading in the instances can be taken from your existing code.

We will use the automated marking system called MarkUs to mark your code (see "Marking Scheme" below). MarkUs is running Python 3.8.3 and your code needs to work with this version of Python.[2]

In this section, we provide the coding requirements and other necessary information for using the MarkUs grading system.

[1] It is left as an exercise to understand why this is true.

[2] Generally, as long as you are using Python 3.5 or greater, you should not have a problem.

2.1 Function Definition

You must submit a Python file called `coding2.py` that contains (perhaps among other functions) a function with the following signature:

> `run_disjunctive(instance)`, where `instance` is a string of the filename of the instance (e.g., "ft06.txt").

The function should read in the problem instance (assume it is in the same folder as the `coding2.py` file), create a MILP model, send the model to Gurobi to find a solution, and interpret the results from Gurobi.

The function should return a tuple containing (this is the same format as the solution in Coding Assignment #1):

- the makespan: integer
- a schedule: a dictionary with keys corresponding to the job number and values being a list of start times for that job's operations following that job's operation order.

MarkUs will call your function and parse the return value to compare it against the correct answer. It is important that you follow the above instructions exactly.

2.2 Using the Gurobi Library and Python Interface

Gurobi and its Python interface are available on the ECF system that you all have remote access to. You can also download the library for academic use.

For all details about installing and examples files of using the interface, please see the Quercus announcement titled "Heads Up: Software to Use for Coding Assignment #2".

2.3 Restrictions on the Code You Write

PLEASE READ CAREFULLY! A NUMBER OF PEOPLE GOT VERY LOW MARKS ON CODING ASSIGNMENT #1 BECAUSE THEY DID NOT ADHERE TO THE RESTRICTIONS!

There are the following restrictions on the code that you may write (in addition to the usual requirements of academic integrity):

- Do not put any `input` or `print` statements in your code. If you have them in your code for testing purposes, they should be commented out in your final submission.
- Do not put any commands outside a function (unless they are within a `if __name__ == ''__main__''` block). They will cause your code to crash when run by MarkUs (and you will fail all the automated tests).
- Do not import any modules except `gurobipy`. This is for two reasons: first, external modules will really not help you much and, second, your code will crash when run by MarkUs if you do so. If you have a good argument as to why you should be allowed to use a particular external module that does not defeat the purpose of this assignment, I am willing to listen to it. Please email Prof. Beck.

- Do not include the following line in your code: `#!/usr/bin/env python`. The MarkUs system does not have the Python implementation that we are using in the same place and the line will make make your code crash. This line is unnecessary in your Python if you run from an IDE or if you run from the command line (e.g., `python coding2.py`).

2.4 Marking Scheme

The marks for this assignment will be distributed as follows:

- 7 marks from MarkUs. MarkUs will call your `run_disjunctive` function with seven different problem instances. If your code returns the correct answer for an instance, you will get 1 mark. If it doesn't return the correct answer, you will get 0 marks for that instance.

- 3 marks from code review. Your code will be manually reviewed (i.e., by a human) and awarded a mark out of 3 based on the following scale:

  0. no code submitted or the code submitted doesn't even try to solve the problem;
  1. code submitted but horrible;
  2. code is pretty good but a few ugly/painful parts;
  3. good code.

Once marks are released, you will have one week to submit a remark request to Prof Beck. Such a request must specify the reason you believe that your code has been marked incorrectly. (E.g., "The correct answer to test #3 is X. The output of my code in MarkUs is X but I received 0 marks for that instance.").

2.5 Submitting Your Code

By the assignment due date, you must submit your Python code file (`coding2.py`) at: `https://markus.engineering.utoronto.ca/mie562` by logging to MarkUs with your UTORID and password.

To submit your solution file, follow the instructions below.

- On your MarkUs homepage, click the assignment name to go to its submission page.

- Click the "Submissions" tab at the top of the page.

- Click "Add A New File". A new row will be added to the table. Click "Choose File" or "Submit" in the new row. (The button's name will depend on your web browser.) In the window that appears, browse to where you have saved your file and select it. Click the button.

- Your submitted file will now appear in the table. Check that the filename is correct (i.e., `coding2.py`). If you click on the filename, you will see the contents of the file. You should check that the contents are correct, and that you have submitted what you wanted to submit.

MAKE SURE THE FILENAME IS CORRECT (`coding2.py`).
MARKUS IS CASE SENSITIVE

If you are unable to submit your code to MarkUs, email Victor (vic.lo@mail.utoronto.ca) for assistance. If there are technical problems with MarkUs at the deadline (e.g., MarkUs is down) you should email your code to Victor and Prof. Beck. If it arrives before the due date and the systems issues are confirmed, your submission will be considered on-time.

2.6 MarkUs Sanity Check

As much as possible, we want to remove errors due to technical issues with MarkUs and your use of it. Therefore, we have provided 5 test instances *that will not be graded* but that you can use to ensure that your code will run within the MarkUs system.

To run non-graded tests:

- Log into MarkUs (see above)
- Click the 'Automated Testing' tab
- Click 'Run Tests' and wait 30-60 seconds then refresh the page
- Check your auto-marked grade out of 5. This is not your assignment grade, this is only for you. You can submit your code an unlimited times before the deadline, but you can only use the auto-tester once per hour.

You should create/find your own test cases and try them on your own computer. The unreleased test cases will contain corner cases that are not included in the released test cases. You should not rely on the MarkUs as a testing framework. Again, auto-test is available for ONE TEST PER HOUR.

3 Academic Integrity

This is an individual assignment and you are, therefore, expected to complete it entirely by yourself.

We will be using code checking software called MOSS (`https://theory.stanford.edu/~aiken/moss/`) to compare the code you submit with all the other submissions. MOSS is a sophisticated algorithm for detecting similarities in the structure of your code. When MOSS flags significant similarity among submitted code, we manually compare the code and if, in our judgment, there is evidence of plagiarism, we will proceed with the University mandated academic offense procedures.

We will also be monitoring sites such as Chegg and others for the appearance of this assignment. If you are a Chegg user, you should be aware that your user agreement allows Chegg to share your personal information with the University of Toronto for suspected infringements of academic integrity. In the last academic year (in the first year Python programming course, which I also teach), a number of penalties for academic offenses were assessed via such information.

Please see the course syllabus for more information on academic integrity. If you have questions, please contact Prof. Beck.

I apologize if the above stresses you out. The reality is that most of you don't cheat and, therefore, this section is completely irrelevant. If you have no intention of cheating, you have nothing to worry about. I am happy to report that we detected no academic misconduct on Coding Assignment #1. That could mean we were not smart enough to see it but I prefer to believe it is because there was none.