Technical Manual

For

SRU Course Scheduling

-Python Update-

Version 1.0

Prepared by William Davis, Samantha Ricketts

Slippery Rock University

March 27 2020

# Overview

This document provides a detailed description of the worked done within the Python Update addition to the SRU Course Scheduling Project.

# System Requirements

* Operating System: Microsoft Windows 7, 8, or 10 (64 bit)
* Working Installation of Python:
  + Python Editor of choice (this will cover the Anaconda Distribution including Spyder)

# Preparing your system

The following steps are needed in order to run convert the generalized Excel document into readable input for the MATLAB algorithm.

3.1.1 Download the GitHub Project

Download the project from the GitHub repository at <https://github.com/williamjwdavis/SRUCourseScheduling> and unzip it to the desired location.

3.1.2 Download & Installation of Anaconda

Download the Anaconda distribution from <https://www.anaconda.com/distribution/>. Note in the “Advanced Installation Options” menu during the installation process, both “Add Anaconda to my PATH…” and “Register Anaconda as my default…” should be checked.

# 3.2 Setup (Only necessary to view and edit the Python scripts)

Upon completion of both downloads, open Spyder (Windows Key + type “Spyder”). Now click “File-->Open-->”and use the browser to navigate to the location of the unzipped GitHub Project. The Python scripts lie within the pythonUpdate folder, and at this point, any files can be viewed or run in isolation using the Spyder environment.

# Running the Script

To run the script from Spyder, simply navigate to the run.py script and press F5, or click run.

Alternatively, the script can be executed by navigating to the pythonUpdate folder through command prompt, GitBash, or any comparable software. To do so, open the terminal and navigate to the file location, followed by using the command “python run.py”.

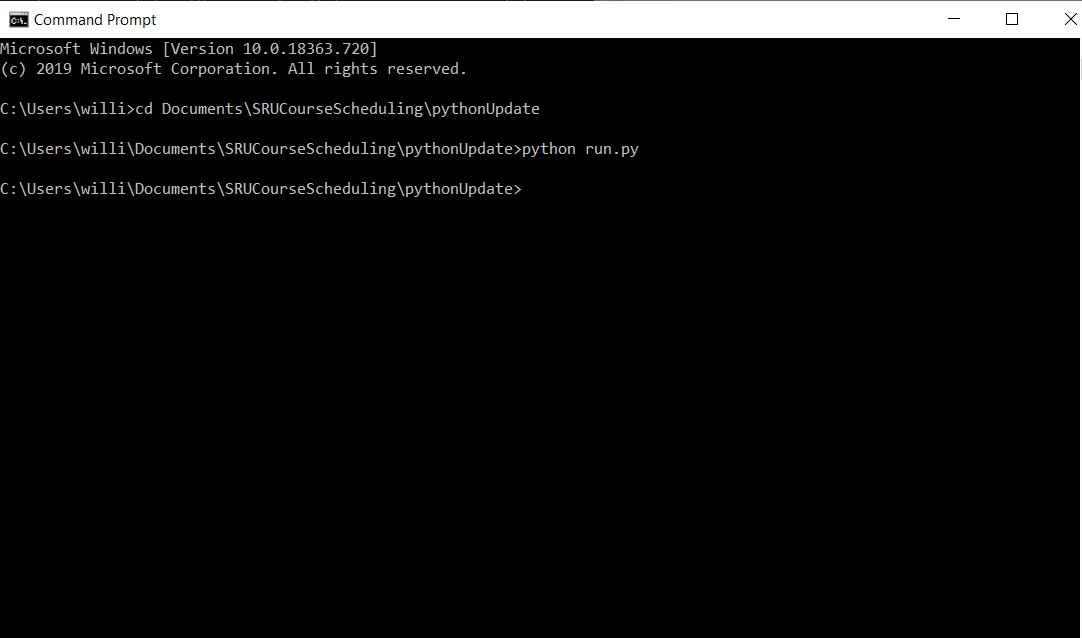


Figure - Command Prompt with the command to run the python script

# Explanation of Files and Folders

5.1 dictionaries

The dictionaries folder contains all of the dictionaries that are used in the preprocessing step. While these dictionaries come premade from GitHub, if these dictionaries aren’t present, they will be created and added to this folder.

5.1.1 classDict.pk1

The classDict dictionary contains a dictionary of class objects that each describe 1 class as directed by the input file in \_\_\_\_\_\_\_\_\_. These objects are directly dependent on the room objects.

5.1.2 profDict.pk1

The profDict dictionary contains a dictionary of professor objects that each describe 1 professor as directed by the input file in \_\_\_\_\_\_\_\_\_. These objects are directly dependent on both the class objects and room objects.

5.1.3 roomDict.pk1

The roomDict dictionary contains a dictionary of room objects that each describe 1 room as directed by the input file in \_\_\_\_\_\_\_\_\_. These objects are not independent of the other objects.

5.1.4 timeEncodingDict.pk1

The timeEncodingDict dictionary contains the first level of encodings of our implementation of the time slots available (this is viewable in \_\_\_\_\_\_\_). This level of encoding still contains meaning and is a product of various levels of modular logic.

5.1.5 timeEncodingDictFinal.pk1

The timeEncodingDictFinal dictionary contains the second level of encodings of the implementation of the time slots available (this is viewable in \_\_\_\_\_). This level of encoding loses the meaning that was present in 5.1.3. This encoding takes the previous encodings and pushes them together, so all of the values are numerically adjacent. This encoding is the final encoding used and passed to MATLAB.

5.2 input

The input folder, containing only Input.xlsx serves as the container for the input file that is used to read in all the information and transform it into a readable MATLAB format. The manual for using Input.xlsx are housed in \_\_\_\_\_\_\_\_\_\_.

5.3 output

The output folder, containing only matlabInputFinal.m serves as the container for the output file that is read into MATLAB, and used as the basis for the algorithm in MATLAB.

5.4 scripts

The scripts folder contains all of the general scripts used in the python preprocessing steps, apart from run.py.

5.4.1 buildTimeConstraints.py

The buildTimeConstraints script takes the timeEncodingDictFinal.pk1 from 5.1.5 and builds an array of forbidden pairs, or time slots that are not permitted to be used for the same class. Time slots are forbidden if they occur on the same day, or on different days but different time windows (i.e. Monday 8AM, Tuesday 9AM). This script is only run if the forbidden pairs array doesn’t exist.

5.4.2 buildTimeDict.py

The buildTimeDict script builds the first encoded dictionary for the time slots. Since the time window in consideration is arbitrary and subjective in the first place, it follows that this level of encoding is rather manual. A visual explanation of the encoding can be found in \_\_\_\_\_\_\_.

5.4.3 buildTimeDict2.py

The buildTimeDict script builds the second encoded dictionary for the time slots. This takes the previous encoding and puts all of the numerical values adjacent to each other according to a regular mod 10 system. This is then exported into timeEncodingDictFinal dictionary.

5.4.4 classInfo.py

The classInfo script contains the Class class, whose attributes come directly from the Input file. In addition to housing the class definition, the getClasses function creates and loads all of the rows from the Class tab in the input file into a dictionary of class objects. Finally, the setAllNonClasses function is called. In order to set the “nonClasses” (classes that cannot occur in the same timeslot as a given class), the dictionary must already be built to call this function. Thus, the dictionary is built before setting the nonClasses and then the function is called. All of this is done with classInfo. This class is also dependent on dictionary of rooms which is detailed in 5.4.9.

5.4.5 createInputVars.py

The createInputVars script essentially takes all the dictionaries, arrays, etc. and loads them into memory so that they may be pulled when exporting the final result. Some attributes are simple and can be loaded in through looping through the elements, while others require a bit more attention. Many of these “forbidden\_\_\_\_” just iterate through each pair and forbid every combination, ensuring that a class has the same professor all days, or a class is in the same room every day.

5.4.6 exportToTxt.py

The exportToTxt script takes all of the variables built from 5.4.5 and places them in a text file. After all of the variables in consideration are placed in this text file, some string manipulation is needed. First, we remove all of the null values. Instead of using a blank as a null space, we use the None Python keyword, which allows us to later remove that keyword from the text output. Second, we format the arrays in such a way that MATLAB is familiar with. Lastly, we remove the commas from the arrays since apparently MATLAB likes numbers together with no separation.

5.4.7 handleTime.py

The handleTime script accepts a dataframe of time values and returns the encoded values. Within this script, there are two functions, one for each of the two levels of encoding. These two functions are used to alleviate any need to manually encode the values.

5.4.8 profInfo.py

The profInfo script contains the Prof class, whose attributes come directly from the Input file. In addition to housing the class definition, profInfo.py also contains getProfs, which builds a dictionary of Prof objects based on the attributes previously mentioned. This class is dependent on both the dictionary of rooms, and dictionary of classes, so profInfo.py is the last script used after both 5.4.4 and 5.4.9 are called.

5.4.9 roomInfo.py

The roomInfo script contains the Room class, whose attributes come directly form the Input file. In addition to housing the class definition, roomInfo.py also contains makeRooms, which builds a dictionary of Room objects based on the attributes previously mentioned. This class is NOT dependent on either of the other two classes, and thus is constructed and called first.

5.4.10 run.py

The run script is used to run the python program. run.py first changes the CWD to the location of the pythonUpdate\scripts folder so that the user doesn’t have to interfere with this, then it checks to see if all of the dictionaries are present. If there are not, run.py will rebuild them. Then we run the exportToTxt script in 5.4.6 to the output folder destination in 5.3

# Time Implementation

The implementation of time is arbitrary and needs to be hard coded into our solution of the problem. This implementation uses a standard Slippery Rock University Schedule where M-F courses are offered, however we ignore night classes. First let’s show the time layout without encoding.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Monday | Tuesday | Wednesday | Thursday | Friday |
| 8AM | 8AM | 8AM | 8AM | 8AM |
| 9AM | 9AM | 9AM | 9AM | 9AM |
| 10AM | 10AM | 10AM | 10AM | 10AM |
| 11AM | 11AM | 11AM | 11AM | 11AM |
| 12PM | 12PM | 12PM | 12PM | 12PM |
| 1PM | 1PM | 1PM | 1PM | 1PM |
| 2PM | 2PM | 2PM | 2PM | 2PM |
| 3PM | 3PM | 3PM | 3PM | 3PM |
| 4PM | 4PM | 4PM | 4PM | 4PM |

This is a standard M-F schedule with normal time slots. Note that the Tuesday Thursday hour windows are considered so that 4 credit courses may be considered. This schedule is overlaid with the standard T-Th schedule that contains 1:45 length classes shown below.

|  |  |
| --- | --- |
| Tuesday | Thursday |
| 8-9:15 | 8-9:15 |
| 9:30-10:45 | 9:30-10:45 |
| 11-12:15 | 11-12:15 |
| Break | Break |
| 2-3:15  3:30-4:45 | 2-3:15 3:30-4:45 |

These are the initial times under consideration. We will proceed encoded with the assumption that this is the schedule format.

The first level of encoding is as follows.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Monday | Tuesday | Wednesday | Thursday | Friday |
| 8 | 00 | 01 | 02 | 03 | 04 |
| 9 | 10 | 11 | 12 | 13 | 14 |
| 10 | 20 | 21 | 22 | 23 | 24 |
| 11 | 30 | 31 | 32 | 33 | 34 |
| 12 | 40 | 41 | 42 | 43 | 44 |
| 1 | 50 | 51 | 52 | 53 | 54 |
| 2 | 60 | 61 | 62 | 63 | 64 |
| 3 | 70 | 71 | 72 | 73 | 74 |
| 4 | 80 | 81 | 82 | 83 | 84 |

8 AM’s always have the first character 0, 9 AM’s 1, and so on.

Similarly, Monday’s always have the second character 0, Tuesday’s 1, and so on.

The logic behind the 1:45 classes remains the same, where the first time slot has a 1, second time slot has a 2, and so on. Tuesdays, now contain a 5, and Thursdays contain a 6 to denote that these time slots are different that the regular hour length courses.

The format for these courses is below.

|  |  |  |
| --- | --- | --- |
|  | Tuesday | Thursday |
| 8-9:15 | 05 | 06 |
| 9:30-10:45 | 15 | 16 |
| 11-12:15 | 25 | 26 |
| Break | 35 | 36 |
| 2-3:15 | 45 | 46 |
| 3:30-4:45 | 55 | 56 |

Considering both of these tables in conjunction allows us to create a correctly encoded schedule.

Now the first round of encoding is complete, and the result is exported as a dictionary where the key is the unique character code for a given time slot, and the value is the encoded numeral.

Lastly, we take this dictionary and condense the values since they are not currently adjacent in a mod 10 system. This is the final encoding used and will be passed to MATLAB and all other functions.