ENGR 212 – PROGRAMMING PRACTICE
SPRING 2015
MINI PROJECT 4

May 3, 2015

While deciding for which classes (or sections of classes) to register for at the beginning of each semester, one of the factors that most students consider is the "niceness" of their weekly time-schedule. A schedule is usually considered as "nice" if courses are lined one after another in a row on a few intensive week days and leaving the other days mostly or completely empty. Another property of a "nice" schedule is that it does not have multiple not-so-long intervals (e.g., 1-2 hours of break) between classes during a day, but rather have long blocks of breaks (if a break is not avoidable).

In this project, you will develop a personal course schedule advisor program that will build and suggest "nice" schedules for students. Figure 1 shows a sample view of your program's user interface.

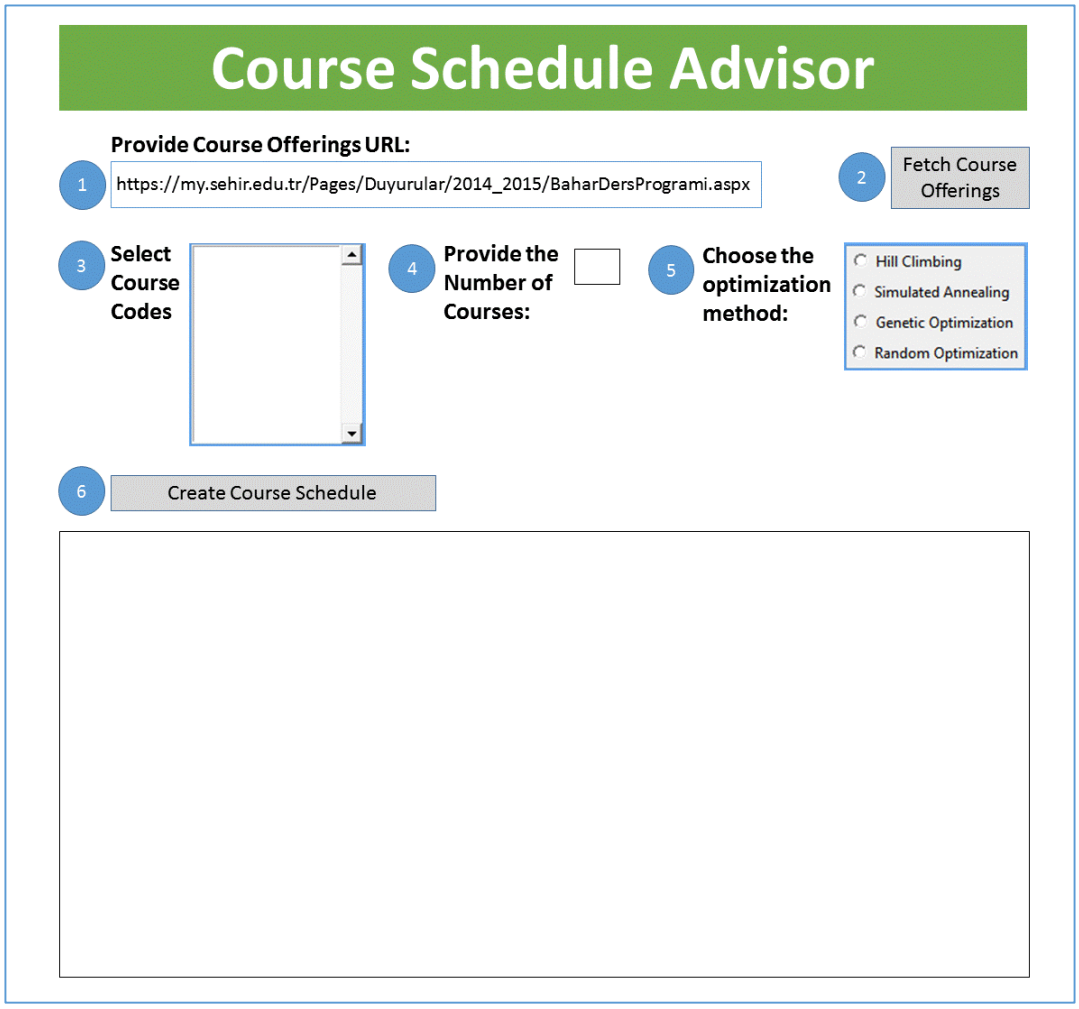

Figure 1

- First, the user will provide the URL to a SEHIR web page that includes the schedule of offered classes. For this project, you may use the following URL for test purposes:

  https://my.sehir.edu.tr/Pages/Duyurular/2014_2015/BaharDersProgrami.aspx

- Then, the user will click on "Fetch Course Offerings" button (step 2), which will download the offered courses and their schedules.
- After downloading and parsing the offered classes (using urllib2 and BeautifulSoup), the list of distinct departmental codes (EE, IE, CS, UNI, etc.) will be automatically extracted from the downloaded list of courses and placed in a list box (step3 in Figure 2). This list box should allow the selection of multiple codes. Here, the user will choose the department codes of

courses that they are interested to register for (students usually follow their department's curriculum for the corresponding semester).

- Next, the user will provide the number of courses that the student wants to register for (step 4)
- As the last input, the user will choose the optimization method that they want to use (step 5). Here, all the specified optimization methods are covered in the class, and provided in optimization.py module as part of project.
- Finally, the user will click on "Create Course Schedule" button (step 6), and see the resulting schedule in a text area (or canvas) at the bottom (Figure 2) (The results are not from a real run. Hence, do not compare them to yours).

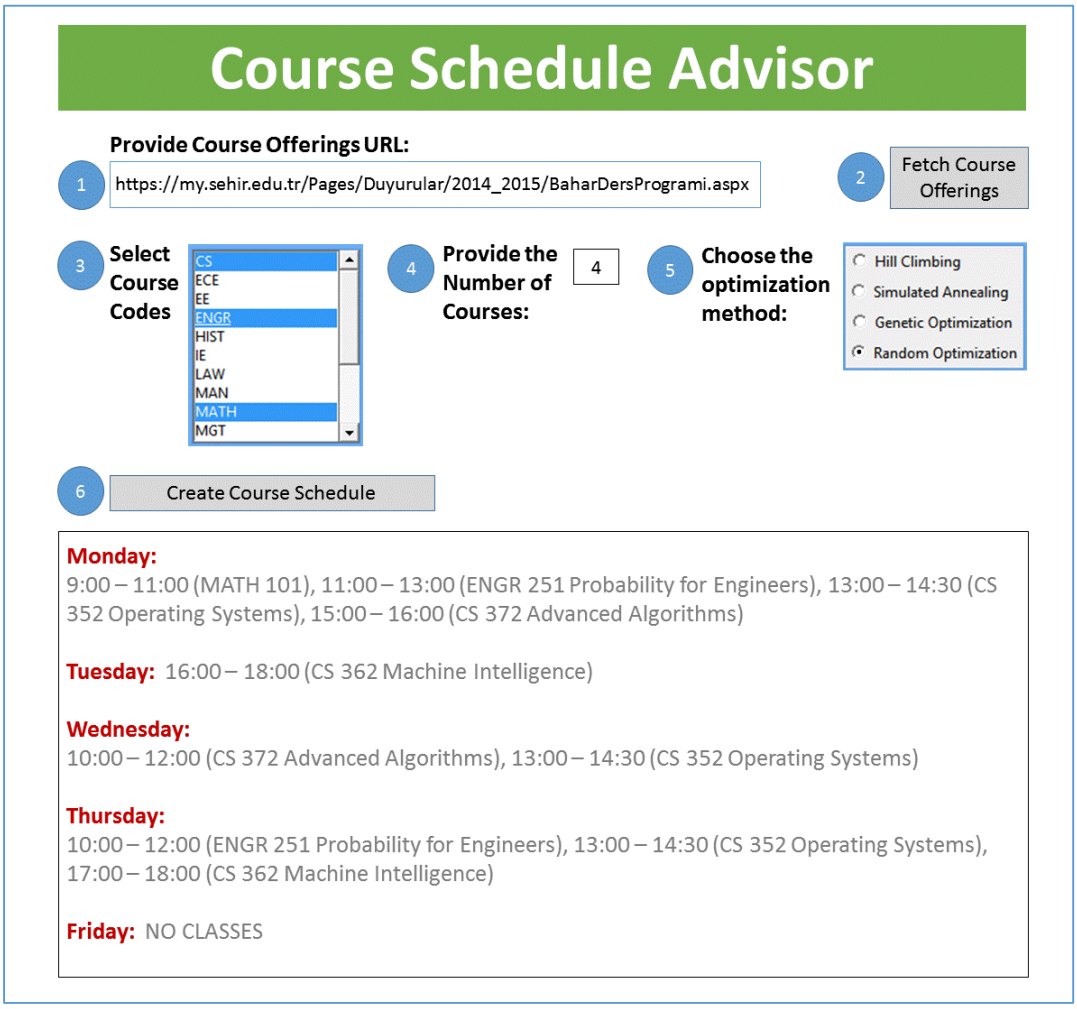

Figure 2

Representation of Solutions:

- You may use a solution representation that is very similar to the one that we used in the dorm example (covered in the class). That is, create a list C of courses that includes all courses whose codes start with any of the selected department codes. That will be the available courses. Now, a solution will be a list of N numbers where N is the number that is provided by the user in step 4. Each number in a solution will be a position index in C. Domain for the numbers in a solution will be very similar to dorm example as well, that is, the domain for the number at index i of a solution will be (0, N-i-1).

Cost Function:

- To keep the project simple, the above described solution representation allows solutions that may introduce course time conflicts. In order to prevent such solutions from appearing as a suggested solution by any of the methods, we will assign a very high cost for solutions that

introduce time conflicts. For each minute where the schedule has at least two courses overlapping, you will add +1000. If in total j minutes are shared by two or more courses, your cost will function will add j*1000 to the overall cost.

- Besides, the breaks between two consecutive courses on a single day will add to the overall cost according to following formula. For a break b, let's say that len(b) is the length of the break in terms of the number of minutes. Then, for a set of breaks, $\{b_1, b_2, \ldots, b_n\}$, the total is computed as $[len(b_1) * len(b_2) * \ldots * len(b_n)]/1000$. Note that a break is always between two courses. Hence, the time before the first class of the day, and the time after the last of the day is not considered as a break.

Can you provide any further pointers that may be helpful? :

- As for the GUI, you **should use** Tkinter that we have covered last semester (see ENGR 211 last week's slides). If you do not like Tkinter, you may use PyQt (we have not covered that in ENGR 211), but in any case, you are **not** allowed to use a designer or any other GUI module (other than the above ones).
- In order to be able to place widgets as shown in Figures, you should heavily use frames. Please see ENGR 211, last week's slides for creating row, col, and grid frames, and their example uses with Swampy.
- To display department codes, please use ListBox. This ListBox should be configured to allow multiple selection, and it should have a vertical scrollbar. The following link has all the information that illustrates how to set multi-select mode, and how to add a vertical scrollbar.
    - http://effbot.org/tkinterbook/listbox.htm
    - Another example to add a vertical scrollbar: http://www.java2s.com/Tutorial/Python/0360__Tkinker/ListBoxwithscrollbar.htm
- For the output area at the bottom, you may use Canvas or Text widget.

How and when do I submit my project? :

- Projects may be done individually or as a small group of two students. If you are doing it as a group, only **one** of the members should submit the project. File name will tell us group members (Please see the next item for details).
- Submit your own code in a **single** Python file (Do **not** include optimization.py that you import). Name it with your and your partner's first and last names (see below for naming).
    - If your team members are Deniz Barış and Ahmet Çalışkan, then name your code file as deniz_baris_ahmet_caliskan.py (Do **not** use any Turkish characters in file name).
    - If you are doing the project alone, then name it with your name and last name similar to the above naming scheme.
- Do **not** copy/paste code from optimization.py into your own code file. Anything that you need from optimization.py should be called with proper dot notation after importing that module.
- Do **not** use any external module other than Swampy, which are not included in standard Python installation.
- Do **not** use Python 3.x. Use Python 2.7.x.
- Submit it online on LMS (Go to the Assignments Tab) by **17:00 on May 17 (Sunday)**.

    Late Submission Policy:

    - -20%: Submissions between 17:01 – midnight (00:00) on the due date.
    - -40%: Submissions which are 24 hour late.
    - -50%: Submissions which are 48 hours late.
    - Submission more than 48 hours late will not be accepted.

Grading Criteria? :

- Does it run? (Submissions that do not run will get some partial credit which will not exceed 30% of the overall project grade).
- Does it implement all the features according to the specifications and produce correct results?
- Code organization (Meaningful names, sufficient and appropriate comments, proper organization into functions and classes, clean and understandable, etc.)?
- Interview evaluation.

Have further questions? :

Please contact your TAs (Mehmet Aytimur and Muhammed Esad Unal) if you have further questions. If you need help with anything, **please use the office hours** of your TAs and the instructor to get help. If office hours are not suitable, please **get** an appointment through email before walking in your TAs offices.