Project Report

Project Title: Timetable Generator

First Name: Zhiyu
Last Name: Kang
Student ID: 1000032993
teach.cs-login ID: kangzhiy
Role: CSP encoding, Course builder, Data researcher.

First Name: Zhihong
Last Name: Wang
Student ID: 1002095207
teach.cs-login ID: wangz154
Role: Example designer, Report writer, Ordering encoding.

First Name: Yichong
Last Name: Guan
Student ID: 1002730529
teach.cs-login ID: guanyich
Role: CSP encoding, Output designer, Code debugger.

Type of project: CSP

Number of team members: 3

Project Motivation/Background:
In our project, we used CSP to plan a smart course timetable of U of T automatically for students' convenience and help them save every annoying time of course planning they waste in summer vocation. This Timetable Generator program can help you plan a personal time table which suits your requirements, without any section conflicts. The reason of we choose CSP as approach is we are more familiar with the topic (we are practiced in assignment 2), and CSP is suitable for timetable design (we described the reasons below). In our program, you will enter a list of course codes, then the program will generate possible timetable for you. User can keep adding course to the list, each action will causes the program to generate a different timetable. The program based on the courses of University of Toronto 2017-2018. We explored the use of CSP to solve for the timetable, because as user keep adding courses, the program will need to make sure that such course addition can end up satisfying different constraints. This program start by an empty timetable, then user will keep adding course to the current timetable. If there is no possible time table for the some courses selection, the program will return an alarm for the conflict or exclusion. If the user want to customize study starting or ending time, the program will also generate possible timetable base on the requirements made by user. After all, CSP is most suitable to achieve these functions.

Methods:
Depending on our CSP problem, the state variables contains course codes and sections informations, and the domain is all lecture sections and their time intervals. We encoded our constraints by making sure there are no two courses will overlap each other.
We encoded our first constraint by putting two non-overlapping courses in a satisfy tuple to try to avoid section conflict.
The second constraint is reflect to the user preferred study starting or ending time, by checking course section time.
We have three order heuristics: Remaining Values (MRV) heuristic, Random and Degree Heuristic (DH). Their differences and output results are described in Evaluation and Results part.

Evaluation and Results:
We evaluate our approaches by letting user add course list(s) and requirements, then generating planned fall and winter timetables. We checked a

lot of cases, such as nearly all CSC courses from 100+ to 300+, and they can be reflected in output timetable as a reasonable plan by user's input and setting.

As TA comments, we add two new order heuristics (other then Minimum Remaining Values (MRV) heuristic in assignment 2): Random and Degree Heuristic (DH). We also record the different time costs between these three order heuristics and the reasons of why it is faster or slower below (the case we tested is in python file: test cases):

Minimum Remaining Values (MRV) heuristic: MRV returns the variable with the most constrained current domain (i.e., the variable with the fewest legal values).

Random heuristic: returns a Variable object var at random.

Degree Heuristic (DH): Select variable that is involved in largest number of constraints on other unassigned variables.

```
MRV:
    FC: 0.026935599000000005
        0.031528422
        0.02777873900000001
        0.026210758
        0.02553735800000001

    GAC:0.39997986900000004
        0.40210787800000003
        0.4146619600000001
        0.4194149300000001
        0.40223084900000006
Random:
    FC: 0.14961161500000003
        0.021729947
        0.01758795399999999
        0.02659309600000001
        0.016408289999999992

    GAC:0.20798771200000005
        0.31917247
        0.214515127
        0.243549075
        0.42193098600000006

DH:
    FC: 0.026323347999999996
```

```
28          0.014509223999999987
29          0.026970558000000006
30          0.019723868000000006
31          0.017065623000000002
32
33      GAC:0.25022165500000004
34          0.22038324900000003
35          0.28000223500000004
36          0.23498503799999998
37          0.259626857
```

It is obvious that DH is faster than MRV because Degree Heuristic will deal the course section which overlap others most, and it saves much time than Minimum Remaining Values (MRV) heuristic. Random is really unstable, the result of it can be either fast or slow.

Limitations/Obstacles:
The way to create course obj must satisfy Acorn data requirements.
Make sure the items in satisfy tuples of constraints are reasonable.
We met the "TypeError: Unhashable type: List", which is because we try to hash List type variables.
Sometimes constraints are not working due to forgetting to change place holder.
How to differentiate study starting or ending time in CSP: we add new inputs "start" and "end" and new constraint for CSP to check this.
How to show alarm when there is a conflict: if the output shows wrong message, then it means conflict.
Some error showing up due to some course themselves, if we remove the course from inputs, the whole program will work again.
The wrong way to set up universal variable may cause some "Empty result" error.
User cannot remove courses from the result. If you want to get new timetable without some courses, you should re-enter a new course list as the input of CSP and get a new result.
We do not add the function that let user add a course section even it will cause a conflict, like Acorn, because we think our program should perform a result with no conflict.

Conclusions:
In the end, we can generate a full result with fall and winter terms timeta-

bles, as user's preference of study time, which is really useful for current university students because a lot of they are tired and boring to do schedule planning manually. So, we provide a auto timetable generator with useful functions to attract those students to become our users. The "overlap" and "study starting or ending time" functions are working well. "overlap" is simple to realize: only need to check if two sections in a same time interval. "study starting or ending time" is related to user requirement, which is the "smart" part because timetable is constructed as user wish, and we achieve this by selecting time ranges, like 9am to 6pm, then try to satisfy the time. During this final project, we become much more expert in CSP area. We learned how to get data from open-source project in github, how to refine CSP, how to build ordering heuristic. We might improve our program by adding better constraints to make FC and GAC faster in the future.

Citations and References:
CSC384 Assignment 2.
UofT Acorn data: https://github.com/cobalt-uoft/cobalt