Artificial Scheduling

Class Schedule as a Constraint Satisfaction Problem with backtracking

Jay Park
jp6ud@virginia.edu

Sujin Park
sjp7yf@virginia.edu

Max Ryoo
hr2ee@virginia.edu

Max Zheng
zz5ra@virginia.edu

Introduction

Class scheduling is a common problem for any college student across the world. Students at the University of Virginia (UVA) have to balance between exams, assignments, and projects throughout the semester and schedule sign ups add even more stress to their life. Students have to scroll through SIS and Lou's List in order to plan for the best possible schedule that not only fits in the right time slots but also fulfill the most amount of required credits, which is time consuming and even frustrating at times. When class sign-up times approaches, most students struggle to optimize their class schedules to satisfy as much requirements as possible.

Therefore, we propose a solution based on Constraint Satisfaction Problem (CSP) to fit the most amount of classes possible (with an upper-bound of 15 credits) in an optimal manner.

Our CSP based solution to will be called Artificial Scheduling. Artificial Scheduling will generate a class schedule of up to 15 credits using CSP and backtracking. This problem of selecting classes can be looked as a CSP where the constraints of the problem will be overlaps of classes, requirements, classes already taken, etc. And given those constraints, the Artificial Scheduler will optimize the amount of requirements satisfied. In order to do so, the Artificial Scheduler will consider the data from two sources: Lou's List and the student. Given these data, available classes will be selected based on already available time slots based on a student's schedule and other requirements.

Purpose/Importance

This is an important problem and solution because students have to stress about organizing and planning their schedule in the midst of their already busy list of activities. UVA Students have to constantly look between Course Forum, Lou's List, and SIS in order to optimize their schedule. This can lead to poor work and time management. Additionally, students may fail to come up with an optimal schedule.

Instead of having to look between three different sources to create a class schedule, students may use Artificial Scheduling to generate an optimal schedule in real time by simply providing the list of courses they have already taken.

While we are aware of websites or applications that help students better organize their existing schedule, they require the students to preselect the classes they want to take and manually type in those classes and times for them to view. Therefore, there is no applications or programs that will generate an optimal schedule for students like Artificial Scheduling. There are

many potential use cases of such a program. Faculty advisors can use this scheduler to quickly and efficiently generate many possible potential schedules for their advisees. This can reduce the amount of time that these advisory meetings take, and ensure that all advisees have a chance to meet with their advisors before the class selection time begins. Students can also simply use such a program themselves to generate a list of courses that they could potentially take. From the generated list, they can then select courses that they would find value in taking.

Data Collection

For the purpose of class scheduling for students of B.A. Computer Science at the University of Virginia. Class schedules were needed with information about course number, Professor, Meeting days, Meeting times (Start and End Time), and the class the course is classified as (Lecture vs. Laboratory vs. Independent study). In order to get this list of courses for the upcoming semester (Spring 2020) the website Lou's List was scraped. Lou's list is a repository that pulls data from University of Virginia Student Information System.

For the purpose of the problem of this paper, classes were limited to University of Virginia Undergraduate Computer Science Courses. This limits the courses numbers to courses before CS5000. All the previous courses (CS1xxx to CS 4xxx) are valid courses that undergraduate computer science majors can take without special permission.

University of Virginia students are also given an option to partake in an independent study with a faculty member from the Computer Science department. However, this course (independent study) has a flexible meeting time (Subject to the students' and professor's personal schedule). Also the independent study can range from credit numbers of 0 - 3 credits, which is chosen by both the professor and student decide on.

The purpose of the problem this paper attempted is for automatic scheduling, which is not possible with flexible classes such as independent study, which therefore the experimenters disregarded and only decided to handle the courses that fall into the category of Lectures or Laboratories.

The dataframe composing all the data had the column names of [Course Number, Professor, Meeting Days, Start Time, End Time, Credits, and Class Type]. The columns of Start Time and End Time were generalized to a total number of minutes after 12:00am.

	Course Number	Professor	Meeting Days	Start Time	End Time	Credits	Class Type
20	CS1501	Maxwell Patek	Mo	780	830	1	Lecture
55	CS3240	Mark Sherriff	Mo	840	915	0	Laboratory
56	CS3240	Mark Sherriff	Mo	930	1005	0	Laboratory
31	CS2110	Nada Basit	Mo	1020	1125	0	Laboratory
33	CS2110	Nada Basit	Mo	1020	1125	0	Laboratory
57	CS3240	Mark Sherriff	Mo	1020	1095	0	Laboratory
29	CS2110	Nada Basit	Mo	1140	1245	0	Laboratory
84	CS4730	Mark Floryan	MoWe	660	710	3	Lecture

Figure 1: Web-scraped data from Lou's List from the link:

https://rabi.phys.virginia.edu/mySIS/CS2/page.php?Semester=1202&Type=Group&Group=CompSci

Analysis/Method

The class scheduling problem was factored to a Constraint Satisfaction Problem (CSP) with backtracking algorithm.

The variables of the CSP problem are the different Class Times. For example for monday/wednesday/friday classes at the University of Virginia will have a class time of possibly 9:00am - 9:50am. For days such as Tu/Thursday classes can have a class time of possibly 9:00am - 10:15am.

The domains of the CSP problems will be the different courses such as CS1110 or CS 4710.

The constraints of this problem was that two different courses cannot fall on the same class time. Classes cannot be double booked. Some constraints for the context of University of Virginia Computer Science courses were that certain classes have to satisfy prerequisites

before taking a course and that some courses are required for students to also enroll in a lab. CS 2150 is a prerequisite class for CS 3330. CS 3330 requires a section of lab that is 0 credits.

Constraint Problem Generalization
Variables = {9:00am - 9:50am on MWF, *...* *8:00am - 8:50am on TuTh* *}* *Domains = {CS1110 ... CS4780}* *Constraints = { $Time_1$!= $Time_2$ }*

Scheduling Algorithm

Once the data was collected and the constraints and problem was defined, a scheduling algorithm based on CSP was developed. The algorithm takes as input: a list of courses that the user has previously taken. This list is necessary because from a practical standpoint, there is no reason to take a course twice. For that reason, the algorithm will omit courses that the user has already taken in a previous semester. Also, the input list of previous courses will be used to determine which prerequisites the user satisfies. The algorithm will only select courses that the user has satisfied the prerequisites for.

Based on this user entered information, the original class information from the scraped Lou's List data is filtered to only include courses that the user has not taken, and satisfies the prerequisites for. This filtered list is passed into the main scheduler algorithm.

The scheduler algorithm is a modified version of the general backtracking CSP algorithm shown below.

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
          add {var = value} to assignment
          result ← RECURSIVE-BACKTRACKING(assignment, csp)
          if result ≠ failure then return result
          remove {var = value} from assignment
  return failure
```

Figure 2: Pseudocode for Constraint Satisfaction Problem with back-tracking

The function "recursive backtracking algorithm" is the modified implementation of the general backtracking CSP algorithm shown above. For this function a set of assigned courses and courses available are passed in as arguments and a randomized schedule is returned with information about whether an optimal schedule of 15 credits were found or that with the given courses the algorithm was not able to find an optimal schedule of 15 credits. The algorithm is shown below.

```
def recursive_backtracking(assignment, courses):

    global max_credits
    global max_assignment

    num_of_credits = num_credits(assignment)
    if num_of_credits >= 15:
        return "Pass"

    if num_of_credits > max_credits:
        max_credits = num_of_credits
        max_assignment = assignment

    for course in courses:
        if not (has_course_number(course, assignment)
                and not
                has_time_overlap(course, assignment)):

            if course[0] in set_haslab:
                lab_added = False
                assignment.append(course)

                for labs in valid_labs:
```

```
                if (labs[0] == course[0]
                    and not
                    has_time_overlap(labs, assignment)):
                    lab_added = True
                    assignment.append(labs)
                    break

            if not lab_added:
                assignment.pop()

        else:
            assignment.append(course)

        result = recursive_backtracking(assignment, courses)

        if result != 'Fail':
            return result

        assignment.pop()

    return 'Fail'
```

Figure 4: Implementation of the Constraint Satisfaction Problem with back-tracking for CS course data from Lous' List .

The assignment is complete when a valid schedule of n credit hours is found where n can be set as any number greater than 0. Upon assignment completion, the algorithm is terminated and the valid assignment is returned, just like in the example algorithm.

Even if the algorithm is not able to find a valid schedule of n credit hours, the user would still like to see a schedule filling as many credits as possible. Therefore, the schedule algorithm stores two additional fields: max_credits and max_assignment. These fields represent the maximum number of credits that the algorithm has managed to successfully fit into a schedule so far, and the assignment which generates that number of credits. After checking for assignment completion, these fields are updated. If the current assignment credit hours is higher than max_credits, then max_credits and max_assignment are set to the current assignment. Then later if the algorithm returns failure, the user will at least be able to see the maximum number of credits that the scheduler was able to fit, and what the corresponding assignment was.

In the actual backtracking algorithm: two constraints need to be checked. The course to be added cannot have a time conflict with any course in the current assignment, and the course to be added cannot be the same course number as any course in the current assignment. The second scenario arises because Lou's List has many sections of the same course.

Another important constraint is the inclusion of courses with labs. Labs must be taken in the same semester as the corresponding course to which they are assigned.

The lab constraint is satisfied by considering both the lab and its corresponding course together. When a course is added to an assignment, the algorithm attempts to find an available time slot which matches the lab time slots. Once a suitable time slot is found, the lab is inserted into the assignment in that time slot. If no suitable time slot can be found, the course cannot be added to the assignment.

Results

The collected data from Lou's List and the backtracking scheduler algorithm was packaged together into a course scheduling application written in Python. The application takes input from the user, who specifies which courses he or she has taken already. Based on this input, the application then displays an optimal schedule to the user which satisfies at least 15 credits.

The number n = 15 was chosen because it represents a full semester course load. It is not likely that one will choose to take more than 15 credits worth of computer science courses in one semester, as students also need to fulfill College of Arts and Sciences general requirements. Because 15 credits consists of on average only 5 courses, the application can return an optimal schedule in less than 1 second.

Sample generated schedules can be viewed in Figures 5, 6, 7, and 8.

```
d-172-25-203-100:ai sujinpark$ python3 backtracking.py
Please enter the courses you have taken : 1110
Didn't find 15 credits of CS courses
-------------------------------------------------------
  Course Number  Meeting Days Start Time End Time  Credits
1        CS2102  [Mo, We, Fr]    10:00am  10:50am        3
2        CS2110  [Mo, We, Fr]    11:00am  11:50am        3
0        CS1501          [Mo]     1:00pm   1:50pm        1
```

Figure 4: Generated class schedule for a student who has only

taken CS1110. Only three classes are scheduled because all other classes have prerequisites that have not yet been fulfilled.

```
d-172-25-203-100:ai sujinpark$ python3 backtracking.py
Please enter the courses you have taken : 1110,2102
Didn't find 15 credits of CS courses
------------------------------------------------------------
  Course Number  Meeting Days Start Time End Time  Credits
0        CS2150  [Mo, We, Fr]    10:00am  10:50am        3
2        CS2110  [Mo, We, Fr]    11:00am  11:50am        3
1        CS2150          [Tu]  _  8:00am   9:15am        0
```

Figure 5: Generated class schedule for a student who has taken CS1110 and CS2102. Only two classes and a lab are scheduled because all other classes have prerequisites that have not yet been fulfilled.

```
d-172-25-203-100:ai sujinpark$ python3 backtracking.py
Please enter the courses you have taken : 1110,2102,2110,21
Found 15 credits of CS courses
------------------------------------------------------------
  Course Number  Meeting Days Start Time End Time  Credits
6        CS4720  [Mo, We, Fr]    10:00am  10:50am        3
0        CS4730      [Mo, We]    11:00am  11:50am        3
2        CS3330      [Mo, We]     2:00pm   3:15pm        3
3        CS3330          [We]     3:30pm   4:45pm        0
4        CS4102      [Mo, We]     5:00pm   6:15pm        3
5        CS3710  [Mo, We, Fr]     9:00am   9:50am        3
1        CS4730          [Th]  _  9:30am  10:45am        0
```

Figure 6: Generated class schedule for a student who has taken CS1110, CS2102, CS2110, and CS2150.

```
d-172-25-203-100:ai sujinpark$ python3 backtracking.py
Please enter the courses you have taken : 1110,2102,2110,2150,4760,4
Found 15 credits of CS courses
------------------------------------------------------------
  Course Number  Meeting Days Start Time End Time  Credits
6        CS4720  [Mo, We, Fr]    10:00am  10:50am        3
0        CS4730      [Mo, We]    11:00am  11:50am        3
2        CS3330      [Mo, We]     2:00pm   3:15pm        3
3        CS3330          [We]     3:30pm   4:45pm        0
4        CS4102      [Mo, We]     5:00pm   6:15pm        3
5        CS3710  [Mo, We, Fr]     9:00am   9:50am        3
1        CS4730          [Th]  _  9:30am  10:45am        0
```

Figure 7: Generated class schedule for a student who has taken CS1110, CS2102, CS2110, CS2150, CS4760, CS4750.

Conclusion

This project explores the possibility of using CSP to generate an optimal course schedule fulfilling the most credit hours. The courses considered are computer science courses offered at the University of Virginia, available to undergraduates pursuing a B.A. Computer Science degree. Data about these courses was scraped from Lou's List, and a scheduling algorithm based on recursive backtracking was developed in order to produce the optimal schedule.

Next steps for this project include transforming the program into a web application and hosting it online so that UVA students can use it to schedule their courses for this upcoming semester. Adding other features such as including class preferences and the inclusion of other majors or courses would also further enhance Artificial Scheduling. This application will be useful for both new and returning students, as it helps to alleviate stress when selecting courses at the beginning of each semester.

Sources

1. Feng, Lu (2019). Constraint Satisfaction Problems.
2. Kumar, V. (1992). Algorithms for constraint-satisfaction problems: A survey. AI magazine, 13(1), 32-32.
3. Sadeh, N., & Fox, M. S. (1996). Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. Artificial intelligence, 86(1), 1-41.
4. Sadeh, N., Sycara, K., & Xiong, Y. (1995). Backtracking techniques for the job shop scheduling constraint satisfaction problem. Artificial intelligence, 76(1-2), 455-480.