Derek Alexander Edrich

Dylan Cockerham

Rodrigo Hernandez

Tu Le Phu Hoang

# Class Scheduling Optimization

# Problem Statement

Universities have thousands of students enrolled and offer hundreds of courses from different schools. However, universities have limited classrooms to hold lectures and discussion for these courses. They are tasked with assigning a number of courses to classrooms, taking into consideration certain constraints, such as the size of a classroom and conflicting times. The class scheduling problem is a nonlinear, multivariable problem.

For our approach, we present a function that determines the effectiveness of our current schedule by calculating a cost. Cost is defined as the number of students that either are in a class that does not fit in a schedule due to conflicting times with another class or do not fit in a classroom because of insufficient space in the room the class was assigned to.

In order to have an effective class schedule, we need to minimize our cost function so that we can achieve in scheduling classes so that there are the minimum amount of students possible that are not in a conflicted class with the given information on the classes and rooms. Simulated annealing will be used to optimize our cost function, as it provides a way for optimizing a function without the need of gradients

**Previous Work**

Class scheduling (CS) problem is one of the most difficult problems in operational research. Given courses, classrooms, professor's schedule, time slots and other parameters, one has to assign courses to classroom and maximize its utility. For example, some professor like to work in the morning. Others like to work in the afternoon. A good schedule should try to fit the professor's preference as much as possible, especially it's not enough to only assign a course to a classroom with higher capacity. It does not make sense to put a class with 45 students to a hall supporting 300 students. However, putting a 50 student to a classroom of 80 can have some benefits. For instance, it reduces a chances of students' cheating. There are many models and formulations that are used to solve this problem.

In Wasfy and Aloul, their approach to the class scheduling problem makes use of local search techniques that try to find a good solution ,such as simuled annealing, genetic algorithm. The problem with theses algorithms is that it does not yield an optimal solution. However, some researches also introduce solution to CS by using linear programming technique combined with Boolean Satisfiability (SAT) solving. One advantage of this approach is that it can tell if problem has a solution or not. SAT problem isformulated in CNF form below.

where (+) represents logical OR, a period (.) represents logical AND and*a,b,c* are Boolean variables with domain {0,1}

The problem is to find a solution that makes the function*f*become 1. Unfortunately, the SAT problem is NP-Complete. Finding solutions for it makes use of backtracking search with some heuristic functions. SAT solvers has been optimized with algorithm such as to minimize the conflict. A disadvantage of the SAT solver is that the problem is formulated as conjunctive normal form (CNF), which restricts its usage. Some researcherpropose pseudo-

where*a[i], b* arepositive integer and *x[i]* are Boolean variables. This proposal extends the its usage drastically by removing exponential constraint. Now, to formulate CS with Linear programming and SAT problem: Given *n* courses and *m* classrooms

Define

With the following constraints:

1. Each course is assigned to one class room
2. Each classroom fits up to one course
3. Each classroom must have capacity greater than course's enrollment

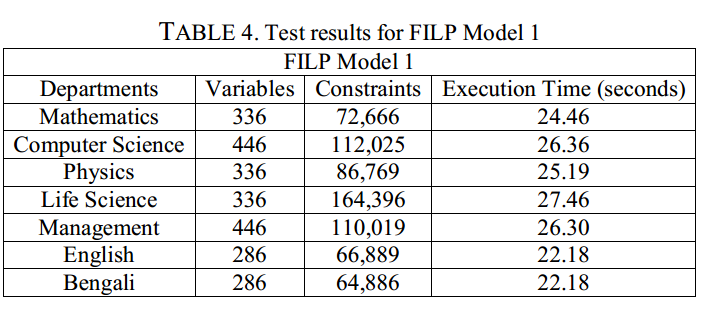
Discussing about utility, it needs t avoid putting low-capacity to big-capacity classroom.

the problem becomes minimizing

with*c[i][j]* equal to the capacity of classroom *j* divided by the number of students in course*i.* Using some Integer Linear Programming solver, like CPLEX, it could solve 100 courses with optimal results in 0.3 seconds on a computer with 4GB of RAM, and 3GHz for CPU.

In the paper by Pupeikiene, et al., they attempt to solve the class scheduling problem in a high school setting, rather than a university. This problem is slightly different in that there are different constraints. For example, a student cannot have a gap in their schedule. Their approach to solving the CS problemis through simulated annealing. This formulation pays attention to the making of a class schedule for students and teachers in high school. The objective function emphasizes reduction in the gap between each class. It should try to reduce the gap as much as possible for the teachers and eliminate the gap for students all together. The solver for class scheduling combines three methods: Monte Carlo, Simulated Annealing, and Based heuristic approach. Some experiments were done with 350 students and 66 teachers. After 10000 iterations, it found a fairly good schedule. However, when the data was increased to include students of around 3000, it takes around 2 hours to generate a good schedule.

Another variant of the class scheduling problem is the examination table problem (ETP). The problems is defined as an assignment of courses to be examined, candidates to time periods and examination rooms that utilized the facilities. There are several constraints, such as having avoiding a high capacity classroom to a course with not many students enrolled. The difference between this approach is that the constraints are divided into soft and hard constraints. The soft constraints can be relaxed. Of course, there is no unique algorithm to apply to all of these variant ETPs. However, there exists some algorithms applicable to variety of problems. Some research also introduced the concept of a graph to the ETP where an arc joins two vertices only if they cannot be scheduled at same time. The problem is solved using fuzzy integer linear programming. It is implemented in Microsoft Visual C++ 6 under window Xp on 2GHz Intel Core 2 Duo with 512 RAM and 1 GB of memory. Here are the results.

****

As we can see, previous works attack problems with different viewpoint. The first one focuses on reducing the capacity of classroom over number of students in a university, while the second focuses on reducing the gap in the student's’ schedule in a high school. The other previous work approaches the class scheduling problem in a different way, constructing a graph to help solve the problem and separating constraints into soft and hard constraints.

# Decomposition

The work for this project seemed to be composed of three different parts: the code, the write up, and the poster.

The code itself was divided into 4 major parts:

1. *Code that simulated the problem:*  We introduced classes, majors, rooms, and timeslots, each with the properties that mimicked that of real life, such as the classes and rooms having different sizes. Initially these numbers were randomized. Eventually, however, they were modeled to reflect that of the University of California Irvine.
2. *New data structures*: Since a class has many properties, we decided to introduce a new Python class, called "Class", that would contain the information of a class, such as the major it is under, the number of students enrolled in the course, its course number, and the timeslot it is assigned. For similar reasons, we also made a class, called "Classroom" for each room. The Classroom class contains the following information of each classroom: the size of the room (number of seats), the number of timeslots that the rooms has, the ID of the classroom, and the assigned timeslots to the classroom.
3. *Function to Optimize:* In order to measure the effectiveness of our class scheduling algorithm, we introduced a function that calculated the cost of our current schedule. As mentioned previously, the cost is simply the number of students that are in a class with a conflicting schedule or that do not have space in the classroom due to insufficient space.
4. *Optimization Code*: Once we have this function, we can optimize it to compute an effective class schedule. We use the simulated annealing method for optimizing our cost function, as it makes things easier since it requires no derivatives. Since one of our homework assignments was to implement simulated annealing, the majority of the code was reused, with only a few minor adjustments.

The decomposition for the project write up followed the structure of the write up in the stated in the project requirements. This resulted in the write up being split into the following partitions: problem statement, citation to previous work, decomposition of the project, experience in coding, experience and results, conclusions from our results, and the bibliography. Since the poster was just a summary of our project, it was simple enough that no further decomposition was needed.

As far as responsibilities go, Derek was the major contributor for the code section, doing everything except for coding up the Python class "Class", which was done by Tu. The write up was left for Rodrigo Hernandez, except for citations to previous work, in which Tu and Dylan researched previous work in class scheduling and wrote the citations for them. The poster was done by Tu, as he has had experience in making poster before. Tu was also responsible for creating the graphs in the results Dylan was the one responsible for gathering data about all the classes and classrooms in UCI so that we can test our class scheduler with real-world data.

**Coding Experience/ Milestones**

After the initial brainstorming of how to approach the problem and getting a general idea of what needed to be done, we set our first milestone. Our first milestone was simple, have a working simulator and optimizer. We decided to use an optimizer that didn't rely on derivatives, so we decided on simulated annealing. Since a previous assignment was to implement simulated annealing, we already had the majority of our code for that part. As for the simulator, we decided to start small scale and without much constraints until later. Our first version of the code was very crude as many things were hard-coded as well as the optimizer not working as well as we had hoped, but it provided a foundation for us to work with.

It was after this initial version of the class scheduling problem in which we realized that a class that encapsulated all the properties of a course and a class that encapsulated the properties of a classroom would simplify things. This resulted in our second milestone: Write two classes, one that simulates a course and another that has the properties of a classroom, as well was rewrite out code so that it implements these new classes and working on the optimizer so that it produced better results. It was also after completing our first milestone that we decided to split the group so that the code and the write up are being worked on in conjunction. This resulted in the write up being partially written by the time the new classes were rewritten and implemented into the code. Writing the classes wasn't hard as it served mostly as a container of information. Once it was written and implemented, however, it made our code much more manageable and easier to change.

With our second milestone finished, our classroom/course generator was more automated, as well as the code being more readable and easier to change because of the implementation of the two classes. A new function for swapping rooms, courses, and time slots resulted in our optimizer to produce better results. However, our code still had minor issues. The problem generator was created classes and rooms with infeasible or illogical sizes/enrollment, such as classrooms with space for only one student, or classes with 3 students. At this point the write up had the problem statement written, as well as the decomposition, and some of the coding experience. We decided to continue working on the write up and code separately since it was working well so far. Our next milestone was to have a problem generator that modeled that of a real University, namely the UC Irvine and to finish the write up, except for the result and conclusion section, which could not be completed until the actual code was finished.

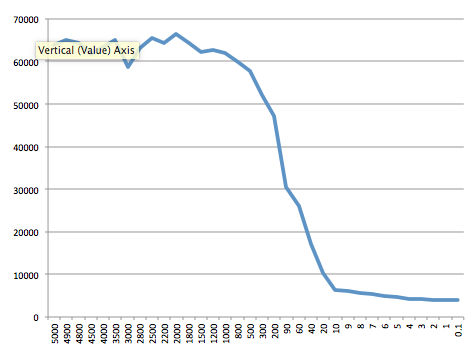
Once the entire UCI class information for the winter quarter was collected , the code was ran with this data. Now that we had numerical results, we constructed a graph to show the effects of our simulated annealing, and wrote down the results in our write up, finishing up this project.

**Results**

Using the classroom and course data from UCI for the 2016 winter quarter, we ran our code. UCI had 1347 different classes (discussion included) along with 131 available classrooms, where each classroom had 15 different time slots available for scheduling classes.

We plotted a graph of the temperature and cost, as well as the table showing which classes were assigned which classrooms (the data for this is not in this write up as it is too big. It is in the file named "classScheduleResutls.txt"

**Simulated Temperature and Cost**

**

*(Vertical: Cost – Horizontal: temperature)*

Looking at our results, we can see that once the temperature dropped down to around 10, our graph started to converge. Our final schedule gave a cost of 3514, with 11 dropped classes, and took around 9360 seconds (160 minutes) to run.

**Conclusion**

While our class scheduler came up with good results, it did not match UCI's system of scheduling classes. This is because 11 classes had to be dropped due to conflicts, which means 3514 students were not able to get a class that they signed up for. As seen in the simulated annealing graph, it converges at around the 100 mark. This convergence is a slow one, giving the range of the temperature (from 5000 to 0 ). As the temperature decreases from the 100 mark, the cost is not reduced much, which means our constraints were strict and made it difficult for our cost to be reduced.

Moreover, let’s take a look on our result (a part of result) like the one below. The column on the left is the size of room and any subsequent columns are the timeslots, with the value in the columns representing the number of students in the class that take up that particular time slot. A "---" appears if that slot has no class using that room during that timeslot. We can see that a lot of slots are not utilized well. For example, the rooms with size 35 in row 2,3,4 have many slots, which are not used at all. We can say that this configuration does not utilized the facilities.

room size: 33 : --- 15 1515 --- --- 15 15 30 3030 32 3232 --- ---

room size: 35 : --- --- --- --- --- --- --- --- 12 12 30 30 --- 35 3535

room size: 35 : --- --- --- --- --- --- --- --- --- --- 24 24 --- --- --- ---

room size: 35 : --- --- --- --- --- --- --- --- --- 31 --- --- --- --- --- ---

room size: 36 : --- --- --- --- 30 3030 --- --- --- --- --- --- --- --- ---

room size: 40 : --- --- --- --- 30 30 35 3535 25 25 40 40 20 20 ---

room size: 40 : --- 30 30 --- 25 2525 40 4040404040404040

room size: 42 : --- --- --- --- --- 25 2525 35 353535 --- --- --- ---

room size: 45 : 44 4444 20 2020 --- 24 24 --- 20 20 --- --- 20 20

room size: 45 : 40 4040 --- --- --- --- 45 4545 35 --- --- --- --- ---

Furthermore, there is still some constraint violation in one of classrooms.

room size: 104 : 100 100100 110 100 100100100100100 110 104 104104 100 100

This classroom allows a maximum of 104 students. However, some slots have 110 students.

**Bibliography**