CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

AN EVOLUTIONARY ALGORITHM FOR TIMETABLING
PROBLEMS: CSUN’S MATH SCHEDULE

A thesis submitted in partial fulfillment of the requirements
For the degree of Master of Science in
Applied Mathematics

by

Maninder Kaur

August 2013
The thesis of Maninder Kaur is approved:

__________________________________________  ____________________________
Dr. Csaba Toth                          Date

__________________________________________  ____________________________
Dr. Mary Rosen                           Date

__________________________________________  ____________________________
Dr. Jorge Balbás, Chair                   Date
Dedication

My wonderful family.
Acknowledgements

I would like to thank my advisor Dr. Jorge Balbas for his tremendous support and commitment towards my thesis research. I would like to special thank all of the faculty members at CSUN and my committee members Dr. Mary Rosen and Dr. Casaba Toth. Thank you so much for helping me to improve my work.

I would personally like to thank my husband, family and friends for supporting me throughout my thesis work. With their support and encouragement, I was able to complete my thesis.
Table of Contents

1 Scheduling Problem 1
   1.1 Introduction ............................................. 1
   1.2 Previous Work ......................................... 1
       1.2.1 Mathematical Matrix Method ....................... 2
       1.2.2 Graph Theory Method .............................. 2
       1.2.3 Integer Linear Programming Method ............... 2
       1.2.4 Lagrangian Relaxation Method ..................... 2
       1.2.5 Genetic Algorithm and Hybrid Algorithms .......... 2
       1.2.6 Simulated Annealing Method ....................... 3

2 Description of Algorithms used in Our Problem 4
   2.1 Genetic Algorithm (GA) ................................ 4
       2.1.1 GA Definitions .................................... 4
   2.2 HereBoy Algorithm .................................... 5
       2.2.1 Adaptive Mutation ................................ 5
   2.3 Simulated Annealing Algorithm ......................... 8
   2.4 Comparison among HereBoy Algorithm and Simulated Annealing Algorithm 8

3 Math Department’s Time Tabling Problem 9
   3.1 Math Department Time Tabling Model .................. 9
   3.2 List of Preferences ................................... 9
   3.3 Simulated Annealing Algorithm applied to Math Departments Timetabling Problem .................................. 10
   3.4 Constraints ............................................. 11
   3.5 Fitness Score ........................................... 12
   3.6 Implementation ......................................... 13
       3.6.1 TimeBlock Class .................................. 13
       3.6.2 CoursePreference Class ............................ 13
       3.6.3 Course Class ....................................... 14
       3.6.4 Professor Class .................................... 14
       3.6.5 Classroom Class .................................... 14
       3.6.6 GeneticAlgorithms Class ......................... 15
       3.6.7 ScheduleSolver Class ............................... 22

4 Results and Conclusions 23
4.1 Examples ................................................................. 23
  4.1.1 Example 1. ..................................................... 23
  4.1.2 Example 2. ..................................................... 24
  4.1.3 Example 3. ..................................................... 26
  4.1.4 Example 4. ..................................................... 27
4.2 Analysis of Examples .................................................. 29
4.3 Future Work ............................................................ 30
ABSTRACT

AN EVOLUTIONARY ALGORITHM FOR TIMETABLE PROBLEMS: CSUN’S MATH SCHEDULE

By

Maninder Kaur

Master of Science in Applied Mathematics

The Mathematics’s Department Timetabling Problem (MDTP) at CSUN is currently solved manually by one faculty member using heuristic methods. This is a daunting task that takes several weeks every semester. Depending on the size of the problem, however, timetabling problems can often be solved with evolutionary algorithms.

In this work we explore three of these algorithms: a purely Genetic algorithm, HereBoy, and Simulated Annealing, and we combine them to develop a hybrid algorithm to solve the MDTP more efficiently. The proposed algorithm combines the genetic algorithm with HereBoy and amounts to a Simulating Annealing algorithm that we apply to help solve the MDTP.

The proposed algorithm is implemented using the Python programming language, and we present it along with several sample schedules that it created. These sample schedules are feasible. Our algorithm would be a valuable tool for creating the Math Department’s semester schedule more efficiently.
Chapter 1
Scheduling Problem

1.1 Introduction

Scheduling is the procedure of deciding how to commit resources between a variety of possible tasks. There are many scheduling problems such as airline fleet scheduling, work-shift scheduling, school bus routing and many more.

The Timetable problem is a well-known NP complete combinatorial problem that consists of scheduling events over a period of time. It is a challenging, complex task that is often seen in education, transportation, health care, sports, and manufacturing [12]. For our purpose, we will focus on the University Course Timetable scheduling Problem (UCTS) for the California State University Northridge (CSUN) Mathematics Department.

UCTS problem has been proved to be non-deterministic polynomial hard problem (NP). Which implies that there is no known polynomial time algorithm that can guarantee finding the best solution [2]. The UCTS problem consists of the following inputs: finite number of professors, courses and time slots in classrooms. Each set of inputs have set of constraints on them. In general, the nature of the constraints varies between different instances of the UCTS problem, but most problems share the following conditions [12].

- No faculty is scheduled to teach two courses at the same time.
- Each professor is teaching during his or her preferred time.

For large UCTS problems, finding optimal solutions are time intensive hence we consider trial and error experiments. Currently, the math department’s schedule is generated heuristically by hand. Creating a university course schedule for any department can be a daunting task. In this work we will formulate a simplified UCTS for the CSUN’s Mathematics Department using less constraints. We will use python to implement Mathematics Department timetabling problem (MDTP).

Our work is organized as follows: in the next section we present an overview of previous attempts to solve timetabling problems, and in Chapter 2 we discuss the details of existing algorithms that we believe are best suited for our particular problem. In Chapter 3 we discuss the details of our algorithm and its implementation, and in Chapter 4 we present several sample schedules created with this implementation.

1.2 Previous Work

Since 1950’s many researchers have been working on alternative methods to solve the timetabling problems. It can be defined as the optimization of allocation of given activities,
actions or events to a set of objects in space-time matrix (search space-matrix) to satisfy a set of desirable constraints [12].

1.2.1 Mathematical Matrix Method

Kuhn’s paper was among of the first papers to be published about solving timetable problem (TTP) in 1955. Kuhn used the Hungarian method to solve TTP, this method is based on matrix reduction [6].

In 1964, Csima and Gotlied suggested a method for constructing timetables, using an iterative process involving Boolean matrices. In limited tests this method supposedly produced successful timetables on every attempt [4].

1.2.2 Graph Theory Method

In 1972, McDiarmid used graph theory to solve the teacher-class TTP of producing such a timetable, which also minimizes the rooms required [11].

1.2.3 Integer Linear Programming Method

In 1969, Lawrie used integer linear programming to formulate TTP. This timetabling problem was based on larger units of departments, groups of pupils and layouts [9].

In 1984, LaPorte and DesRoches used integer linear programming to solve examination TTP. The system was implemented at École des Hautes Etudes Commerciales de Montreal [7].

1.2.4 Lagrangian Relaxation Method

In 1984 Tripthy formulated a UCTS as a large integer linear programming problem. The solution is based on Lagrangian relaxation joined with subgradient optimization. [14].

1.2.5 Genetic Algorithm and Hybrid Algorithms

In 1992, Colorni, Dorigo, Maniezo used a genetic algorithm to solve TTP. They compared their results with simulated annealing and tabu search. The genetic algorithm produced better timetables than simulated annealing, but slightly worse timetables than tabu search for their experiments [3].

Next, in 2000, Levi proposed a fast evolutionary algorithm called HereBoy. HereBoy is an evolutionary algorithm that combines features from genetic algorithm and simulated annealing [10]. In 2003, Ellingsen and Penaloza used HereBoy to solve the South Dakota School of Mines’s UCTS [5].

In 2009, Wang, Huang and Zou used Multi-Strategy Course Scheduling Algorithm which based on greedy algorithm, genetic algorithm and tabu search [15].
1.2.6 Simulated Annealing Method

In 1991, Abramson used Monte Carlo scheme called simulated annealing is used as an optimization strategy. According to this paper, simulated annealing provided reasonable results. He applied this using parallel computer system and showed that the speed of the algorithm enhanced along with results [1].

We chose simulated annealing algorithm to solve CSUN mathematics department’s TTP due to time constraints, computational and ease of implementation.
Chapter 2
Description of Algorithms used in Our Problem

In this chapter we provide detailed description of the following three evolutionary algorithms.

1. Genetic Algorithm
2. HereBoy Algorithm
3. Simulated Anealing Algorithm

We focus on brief theory and properties of each algorithms.

2.1 Genetic Algorithm (GA)

Genetic algorithms are analogous to the theory that was proposed by Charles Darwin about biological evolution. Genetic algorithms are based on producing genetically fitter individuals from a population. GA generates solutions to the problems by using operators like crossovers, mutations and fitness value.

2.1.1 GA Definitions

2.1.1.1 Chromosomes

Definition 2.1.1 (Biology Chromosome) Chromosome, the microscopic threadlike part of the cell that carries hereditary information in the form of genes.

Definition 2.1.2 (MDTP Chromosome) Chromosome, represents a schedule.

2.1.1.2 Crossover

Definition 2.1.3 (Biology Crossover) In an evolutionary manner, crossover selects genes from parent chromosomes and develops a new child.

Definition 2.1.4 (MDTP Crossover) Crossover, when two schedules exchange parts of their schedules.

2.1.1.3 Mutation

Definition 2.1.5 (Biology Mutation) In a biological manner, mutation takes an chromosome and arbitrarily changes a data within the chromosome.

Definition 2.1.6 (MDTP Mutation) Mutation when a part of schedule is arbitrarily changed.
2.1.1.4 Fitness Value

Definition 2.1.7 (Biology Fitness Value) In a biological manner, fitness value evaluates the quality of a chromosome.

Definition 2.1.8 (MDTP Fitness Value) Fitness Value, evaluates the quality of the schedule based on satisfaction of constraints.[8]

Algorithm 1 Genetic Algorithm [8]

Inputs population of chromosomes.

Steps
1: Given a initial population of chromosomes.
2: Each chromosome in the population is given a fitness score.
3: Select chromosomes with better fitness score call it intermediate population of chromosome.
4: Discard chromosomes with weaker fitness score.
5: if selected chromosomes in step 3 meet the requirement. then
6: We keep the chromosomes with higher fitness values.
7: else if step 3 does not meet the requirement. then
8: crossover or mutation operators are applied to step 3.
9: else
10: end if
11: repeat
12: Go to step 5.
13: until you find a good set of chromosomes.

Output The goal is to find good set of chromosome.

2.2 HereBoy Algorithm

HereBoy uses only single chromosome and it uses only mutation to study the search space.

2.2.1 Adaptive Mutation

HereBoy uses the following scheme to generate the number of bits mutated on a chromosome.

Steps:

1. HereBoy works out the general structure of a chromosome by using high mutation rate.
2. Then lower bit mutation rate is applied to work out the detailed structure of a chromosome.
**Definition 2.2.1 (Mutation Rate)**  *Mutation rate* is a measure of the rate at which various types of mutations occur during some unit of time.

\[
\text{Mutation Bits} = \alpha \ast \beta
\]

\[
\alpha = \text{Maximum Mutation Rate} = \text{user fraction} \ast \text{chromosome bits}
\]

\[
\beta = \frac{\text{Max Fitnes Score} - \text{Max Current Score}}{\text{Max Fitness Score}}
\]

\(\alpha\) governs the number of bits in terms of 0’s and 1’s mutated at every iteration. Each mutation is applied randomly selecting a location on the chromosome and flipping the bit. Maximum mutation rate a measure of the rate at which various types of mutations occur during some unit of time. \(\beta\) is relative change [10]. We will illustrate an example of mutations used in the HereBoy Algorithm.

**Example 1**

*Assume User Fraction \cdot Chromosome bits = 1*

*Assume Maximum Fitness Score = 100*

*Assume current Fitness Score = 65*

*Therefore, \(\alpha_0 = 1\)*

\[
\beta_0 = \frac{100 - 65}{65} = 35\%
\]

*Heuristically, this means that we need to mutate chromosome by 35% to achieve better fitness score next time around.*

Finally, apply Algorithm 3 with respect to mutation rate with respect to \(\alpha_0 \cdot \beta_0\). We can stop running Algorithm 2 using one of the below algorithms.
**Algorithm 2** HereBoy Algorithm Operates as follow [10]

**Inputs:** single chromosome, $\alpha_0, \beta_i$

**Steps**

1: HereBoy initializes a single chromosome call it $C_0$.
2: Mutate $C_0$ at rate of $\alpha_0 \cdot \beta_0$, call mutated $C_0$ to be $C_1$.
3: Evaluate score of $C_0$ and $C_1$.
4: **if** $C_1$ score $> C_0$ score **then**
5: keep $C^* = C_1$
6: compute $\beta_{i+1} = \frac{\max C_1 \text{score} - \max C_0 \text{currentscore}}{\max C_1 \text{score}}$
7: **else if** $C_1$ score $< C_0$ score **then**
8: perform a probability test to keep $C_1$
9: **else**
10: $C^* = C_0$
11: **end if**

**Output** The output is chromosome with higher score and $\beta_i, \alpha_0$

---

**Algorithm 3** Run Algorithm 2 n times

**Inputs** single chromosome, $\alpha_0, \beta_0$

**Steps**

1: set $n = N$
2: **for** $i = 1 \cdots n$ **do**
3: Run Algorithm 2
4: **end for**

---

**Algorithm 4** Run Algorithm 2 in a certain time span

**Steps**

1: let $t_0 = \text{start time}, t_1 = \text{end time}, t^* = \text{current time}$
2: **while** $t_0 < t^* < t_1$ **do**
3: Run Algorithm 2
4: **end while**

---

**Algorithm 5** Run Algorithm 2 until certain score is met

**Steps**

1: set $n = N$, $i = 0$.
2: let $t_0 = \text{start time}, t_1 = \text{end time}, t^* = \text{current time}$
3: initialize the target score
4: **while** $t_0 < t^* < t_1$ and chromosome fitness score $< \text{target score}$ **and** $i < n$ **do**
5: Run Algorithm 2
6: $i = i+1$
7: **end while**
2.3 Simulated Annealing Algorithm

Simulated Annealing algorithms are analogues to the procedure of annealing in metal work. The ultimate goal of this algorithm is to bring the system from an arbitrary initial state to a state with the minimum possible energy.

Algorithm 6 Simulated Annealing Algorithm Operates as follow [10]

Steps
1: Simulated Annealing initializes a single chromosome call it \( C_0 \).
2: Evaluate score of \( C_0 \).
3: Apply 1 bit mutation to \( C_0 \), call mutated \( C_0 \) to be \( C_1 \).
4: Evaluate score of \( C_1 \).
5: Compare score of \( C_0 \) and \( C_1 \).
6: if \( C_1 \) score > \( C_0 \) score then
7: then keep \( C_1 \)
8: else if \( C_1 \) score < \( C_0 \) score then
9: perform a probability test to keep \( C_1 \)
10: else
11: discard it completely
12: end if

2.4 Comparison among HereBoy Algorithm and Simulated Annealing Algorithm

When Simulated Annealing algorithm is compared with HereBoy algorithm, then we can conclude that Simulated Annealing is a HereBoy algorithm for 1 bit mutation case. For instance lets look at the following case:

Let us assume we are applying Algorithm 3 to this case. Then apply one bit mutation randomly to it.

Assume, User Fraction = \( \frac{1}{n} \)

Assume Chromosome Bits = \( n \)

\[ \Rightarrow \alpha_i = \frac{1}{n} \times n \]

let \( \beta_i = 1 \)

\[ \Rightarrow \alpha_i \times \beta_i = 1 \]

Finally, we can conclude that Simulated Annealing is a special case of HereBoy Algorithm specifically, when we have the above case.

In the next chapter, we will discuss about the model for our problem and we will be using Simulated Annealing Algorithm for our specific problem.
Chapter 3
Math Department’s Time Tabling Problem

In general, math department offers about 200 sections of courses and 75 sections of labs, which are taught by 50 professors in 40 classrooms. In math department, it takes about a month for Dr. Rosen to create a schedule for every semester. She has created the schedule heuristically for the last 20 years. She has created a schedule preference sheet to facilitate her and her contemporaries. Therefore, we tried creating MDTP based on Dr. Rosen’s schedule preference sheet, i.e. faculty member constraints. In this chapter we pose a model for MDTP and cover model implementation.

3.1 Math Department Time Tabling Model

The MDTP components are the following:

- K number of courses.
- M number of full time instructors.
- R number of classrooms. Each set of inputs have set of constraints on them see section 3.4.

Output:

Our ultimate goal is to produce a feasible schedule that assigns instructors and courses at specified times satisfying as many constraints as possible with highest fitness score.

The following is the list of preferences.

3.2 List of Preferences

1. dayTimePref: This preference contains blocks of time for the professors to choose.
2. backToBackPref: If any professor wants to teach back to back classes.
3. boardPref: This preference contains chalkboard choice and white board choice.
4. fiveUnitPref: Five unit preference is composed of five unit courses and professors can rank those courses as well.
5. oneHundredLevelPref: This preference provides the list of one hundred level courses and professors can rank these courses as well.
6. twoHundredLevelPref: This preference provides the list of two hundred level courses and professors can rank these courses as well.
7. dayTimeCoursePref: This preference provides the list of day time courses and professors can rank these courses as well.

8. afternoonCoursePref: This preference provides the list of afternoon level courses and professors can rank these courses as well.

9. graduateLevelPref: This preference provides the list of graduate level courses and professors can rank these courses as well.

3.3 Simulated Annealing Algorithm applied to Math Departments Timetabling Problem

First, we will discuss about Simulated Annealing algorithm that we used to solve Math Department’s Timetabling problem.

Algorithm 7 Simulated Annealing Algorithm Operates as follow:

<table>
<thead>
<tr>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Simulated Annealing initializes a single schedule call it ( S_0 ).</td>
</tr>
<tr>
<td>2: Evaluate Fitness Score of ( S_0 ).</td>
</tr>
<tr>
<td>3: Apply 1 bit mutation to ( S_0 ), call mutated ( S_0 ) to be ( S_1 ).</td>
</tr>
<tr>
<td>4: Evaluate Fitness Score of ( S_1 ).</td>
</tr>
<tr>
<td>5: Compare Fitness Score of ( S_0 ) and Fitness Score of ( S_1 ).</td>
</tr>
<tr>
<td>6: if ( S_1 ) Fitness Score &gt; Fitness Score of ( S_0 ) then</td>
</tr>
<tr>
<td>7: keep ( S_1 )</td>
</tr>
<tr>
<td>8: else if ( S_1 ) Fitness Score &lt; ( S_0 ) Fitness Score then</td>
</tr>
<tr>
<td>9: else</td>
</tr>
<tr>
<td>10: discard it completely</td>
</tr>
<tr>
<td>11: end if</td>
</tr>
</tbody>
</table>
In order for this algorithm to work, we had to implement MDTP the following way. *Mutations, Constraints and Fitness Score* played a major role to satisfy the preferences and explore almost entire problem space.

The following are the mutation operators that helped us to solve the MDTP.

<table>
<thead>
<tr>
<th>Mutations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same Day Course Swap</td>
</tr>
<tr>
<td>Same Day Professor Swap</td>
</tr>
<tr>
<td>Same Day Time Hour Swap</td>
</tr>
<tr>
<td>Different Day Course Swap</td>
</tr>
<tr>
<td>Different Day Professor Swap</td>
</tr>
<tr>
<td>Different Day Time Hour Swap</td>
</tr>
<tr>
<td>Classroom Swap</td>
</tr>
</tbody>
</table>

### 3.4 Constraints

- **Fixed Constraint**: Fixed constraint is the constraint with the highest priority. In our case, in order for a schedule to be valid all of the fixed constraints must meet. We have grouped the following preferences as fixed constraints. In my opinion, it is very important for a professor to teach a course during his or her preferred time.

<table>
<thead>
<tr>
<th>Preference List</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day Time Preferences</td>
<td>Every professor is teaching during his or her preferred time.</td>
</tr>
<tr>
<td>Overlap</td>
<td>No faculty is scheduled to teach two courses at the same time</td>
</tr>
</tbody>
</table>

- **Hard Constraint**: Hard constraint has lower priority than the fixed constraint. Hard constraints are more critical than the soft constraints, but less critical than the fixed constraints. For instance, it is important for professors to teach the preferred course rather than teaching in preferred classroom. Hard constraints carry more weight than the soft constraints. Scores of each preferences were calculated according to the rank of each course.

<table>
<thead>
<tr>
<th>Preference List</th>
</tr>
</thead>
<tbody>
<tr>
<td>One Hundred Level Courses</td>
</tr>
<tr>
<td>Two Hundred Level Courses</td>
</tr>
<tr>
<td>Five Unit Courses</td>
</tr>
<tr>
<td>Day Time Courses</td>
</tr>
<tr>
<td>Afternoon Courses</td>
</tr>
<tr>
<td>Graduate Level Courses</td>
</tr>
</tbody>
</table>

Hard Constraint Scores are calculated as follows:
1. For instance if a professor is teaching course 1 from one hundred level preferences and he ranked that to be first then the professor will get 100 points for that course.

2. Next, if a professor is teaching course 2 from two hundred level preference, and that was ranked 2nd, then the professor will get 50 points.

\[ \text{Hard Constraint score} = \frac{100}{2^n}, \text{where } n \text{ is ranking number} \]

The reason, we chose to calculated the hard constraint using the exponential function is because, I wanted to prioritize the first and the second ranked course.

- **Soft Constraint:** Soft constraint is the constraint with the lowest priority. I.e. soft constraint score carry less weight than the hard constraint score.

<table>
<thead>
<tr>
<th>Preference List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back to Back Preference</td>
</tr>
<tr>
<td>Board Preference</td>
</tr>
</tbody>
</table>

Soft constraints scores are calculated as follows:

- If the professor has met the above constraints then 50 points will be added to his or her score function for each constraint or else 0 points will be added.

### 3.5 Fitness Score

*Fitness Score:* Fitness score evaluates the quality of a schedule. Fixed constraints are evaluated as indicator function. The following algorithm helps us to compute the scores of hard constraints and soft constraints.
Algorithm 8 computeSoftHard algorithm operates as follow:

Steps
1: Set total hard constraint score for each professor equal to zero initially.
2: Set total soft constraint score for each professor equal to zero initially.
3: In order to compute the score, we iterate over all the professors.
4: First algorithm evaluates fixed constraints using hasOverlap function and rankScore method based on day time score.
5: Second, algorithm evaluates hard constraint scores.
6: Preferences scores are calculated using rankScore function for each professor.
7: Third, algorithm evaluates soft constraint scores for each professor.
8: Preferences scores are calculated using both cases of backToBackPref and computeBoardScore function.
9: Finally, as we are done iterating over all of the professor
10: Return Total Score = \( \frac{\text{total hard constraint score} + \text{total soft constraint score}}{\text{total number of courses}} \)

3.6 Implementation

To keep track of data used in this program, the data was divided into seven python classes and stored in seven different files. These seven files stored the data about classrooms, courses, professors and time.

3.6.1 TimeBlock Class

Member Variables of TimeBlock Class

<table>
<thead>
<tr>
<th>Member Variables</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>days</td>
<td>enumerated type {MTR, MTWR, MW, MWF, TR, TF}</td>
</tr>
<tr>
<td>startTime</td>
<td>datetime.time</td>
</tr>
<tr>
<td>endTime</td>
<td>datetime.time</td>
</tr>
<tr>
<td>duration</td>
<td>datetime.timedelta</td>
</tr>
</tbody>
</table>

Methods of TimeBlock Class

1. timeDuration() : Input for this function are startTime and endTime. This function calculates the time difference between startTime and endTime.

3.6.2 CoursePreference Class

<table>
<thead>
<tr>
<th>Member Variables</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>descriptor</td>
<td>string or integer or TimeBlock</td>
</tr>
<tr>
<td>rank</td>
<td>integer</td>
</tr>
</tbody>
</table>
Method of CoursePreference Class

1. matches(): Input for this function is coursesTaught. First, this function checks if course title and descriptor are type of basestring. Second, descriptor and section number is type of integer. Lastly, this function checks

3.6.3 Course Class

Member Variables of Course Class

<table>
<thead>
<tr>
<th>Member Variables</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
</tr>
<tr>
<td>sectionNumber</td>
<td>integer</td>
</tr>
<tr>
<td>units</td>
<td>integer</td>
</tr>
<tr>
<td>timeblock</td>
<td>TimeBlock</td>
</tr>
<tr>
<td>professor</td>
<td>Professor</td>
</tr>
</tbody>
</table>

Methods of Course Class: None

3.6.4 Professor Class

Member Variables of Professor Class

<table>
<thead>
<tr>
<th>Member Variables</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>professorId</td>
<td>string</td>
</tr>
<tr>
<td>daytimePref</td>
<td>dictionary {key: string, value: datetime.time}</td>
</tr>
<tr>
<td>backToBackPref</td>
<td>boolean</td>
</tr>
<tr>
<td>boardPref</td>
<td>enumerated type {chalk, white, combo}</td>
</tr>
<tr>
<td>oneHundredLevelPref</td>
<td>list</td>
</tr>
<tr>
<td>twoHundredLevelPref</td>
<td>list</td>
</tr>
<tr>
<td>dayCoursePref</td>
<td>list</td>
</tr>
<tr>
<td>afternoonCoursePref</td>
<td>list</td>
</tr>
<tr>
<td>graduateLevelPref</td>
<td>list</td>
</tr>
<tr>
<td>fiveUnitPref</td>
<td>list</td>
</tr>
<tr>
<td>largeSectionPref</td>
<td>list</td>
</tr>
</tbody>
</table>

3.6.5 Classroom Class

Member Variables of Classroom Class
<table>
<thead>
<tr>
<th>Member Variables</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1    ClassroomId</td>
<td>integer</td>
</tr>
<tr>
<td>2    classroomSize</td>
<td>integer</td>
</tr>
<tr>
<td>3    timeblock</td>
<td>TimeBlock</td>
</tr>
<tr>
<td>4    boardType</td>
<td>enumerated type</td>
</tr>
<tr>
<td>5    schedule</td>
<td>dictionary {key: timeblock.days, value: [courses]}</td>
</tr>
</tbody>
</table>

**Methods of Classroom Class**

1. **addCourse()**: Input for this function is a course. This function adds a course to the schedule dictionary.

2. **getNumberOfCourses()**: This method iterates over all the courses in the classroom incrementing a counter at each iteration and return the total number of courses in the classroom. The return type is an integer.

3. **getprofessorCourseMapper()**: This method generates and returns a dictionary thats maps professors to a list of courses which the professor teaches.

**3.6.6 GeneticAlgorithms Class**

**Member Variables of Classroom Class**: None

**Methods of GeneticAlgorithms Class**

1. **sameDayCourseSwap()**: The input for this method is classroom. This function basically swaps the courses that occur on the same day and have same time duration in one classroom.
Algorithm 9 sameDayCourseSwap()

Input: classroom

Steps
- Given a classroom and make a copy of the given classroom.
- Randomly select a single element out of the set of days of the selected classroom.
- Obtain a list of courses taught in the selected days from step 2.
- Selects two random courses from step 3, refer to it as course 1 and course 2 for remainder of this algorithm.

\[\text{if course 1 time duration is not the same as the course 2 time duration then}\]
\[\text{return none}\]
\[\text{else}\]
\[\text{Swap course 1 with course 2.}\]
\[\text{Swap the time blocks of course 1 with course 2, returning them to their original positions.}\]
\[\text{Swap the professors from course 1 with course 2, returning them to their original positions.}\]
\[\text{Update the schedule in the copy created in Step 1 and return it.}\]
\[\text{end if}\]

Algorithm 10 sameDayProfessorSwap()

Input: classroom

Steps
1. Given a classroom and make a copy of the given classroom.
2. Randomly select a single element out of the set of days of the selected classroom.
3. Obtain a list of courses taught in the selected days from step 2.
4. Selects two random courses from step 3, refer to it as course 1 and course 2 for remainder of this algorithm.
5. Swap the professors of course 1 and course 2
6. Update the schedule in the copy created in Step 1 and return it.

2. sameDayProfessorSwap(): The input for this method is classroom. This method basically swaps the professors that teach during the same days.
3. `sameDayTimeHourSwap()`: The input for this method is classroom. This function swaps the times of courses that are offered during same set of days.

**Algorithm 11** `sameDayTimeHourSwap()`

**Input**: classroom  
**Steps**  
1. Given a classroom, make a copy of the given classroom.  
2. Randomly select a single element out of the set of days of the selected classroom.  
3. Obtain a list of courses taught in the selected days from step 2.  
4. Selects two random courses from step 3, refer to it as course 1 and course 2 for remainder of this algorithm.  
5. **if** course 1 time duration is not the same as the course 2 time duration **then**  
6. return none  
7. **else**  
8. Swap course 1 with course 2.  
9. Update the schedule in the copy created in Step 1 and return it.  
10. **end if**
4. **differentDayCourseSwap()**: The input for this method is classroom. This function swaps the courses that occur on different days in one classroom, but have same time duration. In this case, we only swap the courses not the professors.

**Algorithm 12 differentDayCourseSwap()**

**Input**: classroom  
**Steps**
1: Given a classroom and make a copy of the given classroom.  
2: Get a list of all days that the given classroom is being used.  
3: Randomly select a particular day out of the list from step 2.  
4: Obtain the list of courses being taught on the day selected in step 3.  
5: Select a random course from step 4, refer it to as course 1 for the remainder of this algorithm.  
6: Repeat steps 3 and 4 to obtain course 2.  
7: if course 1 time duration is not the same as the course 2 time duration then
   8: return none  
   9: else
10: Swap course 1 with course 2.  
11: Swap the time blocks of course 1 with course 2, returning them to their original positions.  
12: Swap the professors from course 1 with course 2, returning them to their original positions.  
13: Update the schedule in the copy created in Step 1 and return it.  
14: end if
5. differentDayProfessorSwap(): differentDayProfessorSwap method swaps the professors only for two courses that are being offered during different set of days.

Algorithm 13 differentDayProfessorSwap()

**Input:** classroom

**Steps**
1: Given a classroom and make a copy of the given classroom.
2: Get a list of all days of the given classroom.
3: Randomly select an element out of the list from step 2.
4: Obtain the list of courses being taught on the day selected in step 3.
5: Select a random course from step 4, refer it to as course 1 for the remainder of this algorithm.
6: Repeat steps 3 and 4 to obtain course 2.
7: Swap the professors of course 1 and course 2.
8: Update the schedule in the copy created in Step 1 and return it.
6. `differentDayTimeHourSwap()`: The input for this function is a classroom. This function swaps two courses offered during different set of days.

**Algorithm 14 differentDayTimeHourSwap()**

**Input**: classroom

**Steps**

1. Given a classroom and make a copy of the given classroom.
2. Get a list of all days that the given classroom is being used.
3. Randomly select a particular day out of the list from step 2.
4. Obtain the list of courses being taught on the day selected in step 3.
5. Select a random course from step 4, refer it to as course 1 for the remainder of this algorithm.
6. Repeat steps 3 and 4 to obtain course 2.
7. if course 1 time duration is not the same as the course 2 time duration then
   8. return none
  9. else
10. Swap course 1 with course 2.
11. end if
7. **classroomSwap():** The input for this function is classroom. This function swaps two classrooms with same time duration.

### Algorithm 15 classroomSwap()

**Steps**

1. Given two classrooms and make a copy of the given classrooms.
2. if Classroom 1 time duration is not same as classroom 2 time duration then
   3. return none
4. else
   5. Obtain the schedule of the copy classrooms
   6. Swap classroom 1 schedule with classroom 2 schedule
   7. Update the schedule of copy classroom 1 and copy classroom 2
8. end if

8. **rankScore():** Objective of this function is to develop a score according to professors ranked preferences. For example, if a professor ranks two courses in large-SectionPref, his or her scores will be calculated as following. If he is teaching both courses, for the first ranked course, 100 points will be assign to that professor. For the second ranked, his scores will be calculated by $\frac{100}{2^n}$, where $n = \text{rank number}$. So in our case it will be $\frac{100}{2^1} = 50$. Every preferences scores are calculated as we had explained in an example.

9. **computeBoardScore():** Objective of this function to check if a faculty member is teaching in his or her preferred type of classroom based on his or her board preference. If a professor is teaching in his preferred classroom, he will get 50 points. If his preference is not met 0 points will be added to the score.

10. **hasBackToBack():** This function checks if two courses in a row are assigned to the same faculty member. Some professors prefer to teach them one after another and others do not. This function takes care of both preferences. If a professor wants to teach back to back he will get 50 points otherwise if his preference is not met, then 0 points will be added to the score function.

11. **professorCourseMapper():** Input for this function is dictionary of classrooms. The output for this function is professorCourseMap, professorClassroomMap, numberOfCourses map between professor and courses taught over all classrooms and sum of the number of courses in all the classrooms.

12. **computeScore():** Input is classroom dictionary. Objective of this function is to compute total score of the following method.

13. **computeSoftHardScore():**
Algorithm 16 computeSoftHard algorithm operates as follow:

**Steps**
1: Initialize total hard constraint score for each professor.
2: Initialize total soft constraint score for each professor.
3: In order to compute the score, we iterate over all the professors.
4: First algorithm evaluates fixed constraints using hasOverlap function and rankScore method based on day time score.
5: Second, algorithm evaluates hard constraint scores.
6: Preferences scores are calculated using rankScore function for each professor.
7: Third, algorithm evaluates soft constraint scores for each professor.
8: Preferences scores are calculated using both cases of backToBackPref and computeBoardScore function.
9: Finally, as we are done iterating over all of the professor
10: Return Total Score = \( \frac{\text{total hard constraint score} + \text{total soft constraint score}}{\text{total number of courses}} \)

3.6.7 ScheduleSolver Class

**Member Variables of ScheduleSolver Class:** None

Algorithm 17 ScheduleSolver algorithm operates as follow:

**Steps**
1: Populate initial dictionary of classrooms.
2: for \( i = 1000 \) do
3: Copy the classroom dictionary from step 1.
4: Choose two random classrooms from classroom dictionary from step 1.
5: Randomly select one of mutation operation.
6: Perform mutation operator from step 5 on to the first classroom; if classroom swap is chosen; then apply mutation operation to both classroom chosen in step 4.
7: Take the new classrooms from step 6, and place them into copy dictionary.
8: Compute the fitness score of the copy dictionary.
9: Compare step 8 score with the original dictionary.
10: Select the dictionary with better score as the original dictionary.
11: end for

**Methods of ScheduleSolver Class**
In this chapter we have tested several sample schedules. In order to test our implementation of the program, we have tested several schedules.

4.1 Examples

4.1.1 Example 1.

Sample schedule 1 is consist of the following data.

- 4 Courses
- 4 Instructors
- 2 Classrooms
- Classroom 1 is available from 9 am till 2 pm and contains chalk board.
- Classroom 2 is available from 9 am till 7 pm and contains white board.

- **Fixed Constraints:** Suppose each of the professors have the following time preferences.

<table>
<thead>
<tr>
<th>Professor</th>
<th>MW</th>
<th>TR</th>
<th>MWF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>09:00 - 12:00</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_1$</td>
<td>08:00 - 12:00</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_2$</td>
<td>None</td>
<td>10:00 - 13:00</td>
<td>None</td>
</tr>
<tr>
<td>$P_3$</td>
<td>None</td>
<td>10:00 - 13:00</td>
<td>None</td>
</tr>
</tbody>
</table>

- **Hard Constraints:** Suppose each of the professors has the following course preferences.

1. One hundred level preferences

<table>
<thead>
<tr>
<th>Professor</th>
<th>1st Ranked Course</th>
<th>2nd Ranked Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>Math 103</td>
<td>Math 102</td>
</tr>
<tr>
<td>$P_1$</td>
<td>Math 102</td>
<td>Math 103</td>
</tr>
<tr>
<td>$P_2$</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_3$</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

2. Graduate level course preferences

<table>
<thead>
<tr>
<th>Professor</th>
<th>1st Ranked Course</th>
<th>2nd Ranked Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_1$</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_2$</td>
<td>Math 552</td>
<td>Math 542d</td>
</tr>
<tr>
<td>$P_3$</td>
<td>Math 542d</td>
<td>Math 552</td>
</tr>
</tbody>
</table>
• **Soft Constraints:** None

---

**Results of Example 1**

<table>
<thead>
<tr>
<th>Initial Schedule</th>
<th>Final Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sect</td>
<td>Course</td>
</tr>
<tr>
<td>0</td>
<td>102</td>
</tr>
<tr>
<td>1</td>
<td>103</td>
</tr>
<tr>
<td>2</td>
<td>552</td>
</tr>
<tr>
<td>3</td>
<td>542d</td>
</tr>
</tbody>
</table>

**Red: Same Day Professor Swap**

Number of mutations before converging: Non deterministic
Fitness score of initial schedule: 75
Fitness score of final schedule: 100

In above example, same day professor swap mutation operation occurred. For this example, this was the most feasible solution.

4.1.2 **Example 2.**

Sample schedule 2 is consists of the following data.

• 5 Courses
• 3 Instructors
• 2 Classrooms
• Classroom 1 is available from 9 am till 2 pm and contains chalk board in it.
• Classroom 2 is available from 9 am till 7 pm and contains white board in it.

• **Fixed Constraints:** Suppose each of the professors have the following time preferences.

<table>
<thead>
<tr>
<th>Professor</th>
<th>MW</th>
<th>TR</th>
<th>MTWR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>12:30 - 14:00</td>
<td>12:30 - 14:00</td>
<td>11:00 - 12:30</td>
</tr>
<tr>
<td>$P_1$</td>
<td>11:00 - 14:15</td>
<td>None</td>
<td>11:00 - 12:15</td>
</tr>
<tr>
<td>$P_2$</td>
<td>None</td>
<td>12:00 - 14:00</td>
<td>09:00 - 11:00</td>
</tr>
</tbody>
</table>

• **Hard Constraints:** Suppose each of the professors has the following course preferences.
1. One hundred level preferences

<table>
<thead>
<tr>
<th>Professor</th>
<th>1st Ranked Course</th>
<th>2nd Ranked Course</th>
<th>3rd Ranked Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>None</td>
<td>None</td>
<td>Math 102</td>
</tr>
<tr>
<td>$P_1$</td>
<td>Math 102</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_2$</td>
<td>None</td>
<td>Math 102</td>
<td>None</td>
</tr>
</tbody>
</table>

2. Two hundred level course preferences

<table>
<thead>
<tr>
<th>Professor</th>
<th>1st Ranked Course</th>
<th>2nd Ranked Course</th>
<th>3rd Ranked Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>None</td>
<td>Math 250</td>
<td>None</td>
</tr>
<tr>
<td>$P_1$</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_2$</td>
<td>Math 250</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

3. Five Unit course preferences

<table>
<thead>
<tr>
<th>Professor</th>
<th>1st Ranked Course</th>
<th>2nd Ranked Course</th>
<th>3rd Ranked Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>Math 105</td>
<td>Math 150A</td>
<td>None</td>
</tr>
<tr>
<td>$P_1$</td>
<td>None</td>
<td>Math 150A</td>
<td>Math 105</td>
</tr>
<tr>
<td>$P_2$</td>
<td>None</td>
<td>Math 105</td>
<td>Math 150A</td>
</tr>
</tbody>
</table>

4. Graduate level course preferences

<table>
<thead>
<tr>
<th>Professor</th>
<th>1st Ranked Course</th>
<th>2nd Ranked Course</th>
<th>3rd Ranked Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>None</td>
<td>None</td>
<td>Math 552</td>
</tr>
<tr>
<td>$P_1$</td>
<td>Math 552</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_2$</td>
<td>None</td>
<td>Math 552</td>
<td>None</td>
</tr>
</tbody>
</table>

- **Soft Constraints**: Suppose each of the professor has the following additional preferences.

<table>
<thead>
<tr>
<th>Professor</th>
<th>Back to Back</th>
<th>Board Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>None</td>
<td>Chalk</td>
</tr>
<tr>
<td>$P_1$</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_2$</td>
<td>None</td>
<td>White</td>
</tr>
</tbody>
</table>

Results of Example 2

<table>
<thead>
<tr>
<th>Initial Schedule</th>
<th>Final Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sect</td>
<td>Course</td>
</tr>
<tr>
<td>0</td>
<td>105</td>
</tr>
<tr>
<td>1</td>
<td>150A</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>3</td>
<td>250</td>
</tr>
<tr>
<td>4</td>
<td>552</td>
</tr>
</tbody>
</table>

Red: Different Day Professor Swap
Number of mutations before converging: Non deterministic
Fitness score of initial schedule: 0
Fitness score of final schedule: 80

As we can analyze the initial schedule, professor $P_1$ had overlap condition. $P_1$ is assigned Math 102 on MW 11:00 - 12:15 and Math 150A on MTWR 11:00 - 12:05. Since any faculty member can not teach two courses at same time, therefore initial schedule is invalid with score of 0.

However, different day professor swap mutation operator was able to take care of this issue. We were able to achieve this feasible schedule.

4.1.3 Example 3.

Sample schedule 3 is consists of the following data.

- 8 Courses
- 3 Instructors
- 2 Classrooms

- Classroom 1 is available from 8 am till 2 pm and contains chalk board in it.
- Classroom 2 is available from 8 am till 7 pm and contains white board in it.

- **Fixed Constraints:** Suppose each of the professors have the following time preferences.

<table>
<thead>
<tr>
<th>Professor</th>
<th>MW</th>
<th>TR</th>
<th>MWF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>09:00 - 12:00</td>
<td>09:00 - 14:15</td>
<td>08:00 - 12:00</td>
</tr>
<tr>
<td>$P_1$</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
</tr>
<tr>
<td>$P_2$</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
</tr>
</tbody>
</table>

- **Hard Constraints:** Suppose each of the professors has the following course preferences.

  1. One hundred level preferences

     | Professor | 1st Ranked Course | 2nd Ranked Course |
     |-----------|-------------------|-------------------|
     | $P_0$     | Math 102          | Math 103          |
     | $P_1$     | Math 103          | Math 102          |
     | $P_2$     | None              | None              |

  2. Two hundred level course preferences

     | Professor | 1st Ranked Course | 2nd Ranked Course | 3rd Ranked Course |
     |-----------|-------------------|-------------------|-------------------|
     | $P_0$     | None              | None              | Math 250          |
     | $P_1$     | Math 250          | None              | None              |
     | $P_2$     | None              | None              | None              |
3. Afternoon course preferences

<table>
<thead>
<tr>
<th>Professor</th>
<th>1st Ranked Course</th>
<th>2nd Ranked Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_1$</td>
<td>Math 350</td>
<td>Math 351</td>
</tr>
<tr>
<td>$P_2$</td>
<td>Math 350</td>
<td>Math 351</td>
</tr>
</tbody>
</table>

4. Graduate course preferences

<table>
<thead>
<tr>
<th>Professor</th>
<th>1st Ranked Course</th>
<th>2nd Ranked Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>Math 542b</td>
<td>Math 552</td>
</tr>
<tr>
<td>$P_1$</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>$P_2$</td>
<td>Math 552</td>
<td>Math 542b</td>
</tr>
</tbody>
</table>

- **Soft Constraints:** Suppose each of the professor has the following additional preferences.

<table>
<thead>
<tr>
<th>Professor</th>
<th>Back to Back</th>
<th>Board Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>True</td>
<td>Chalk</td>
</tr>
<tr>
<td>$P_1$</td>
<td>False</td>
<td>White</td>
</tr>
<tr>
<td>$P_2$</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

---

**Results of Example 3**

<table>
<thead>
<tr>
<th>Initial Schedule</th>
<th>Final Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sect</td>
<td>Course</td>
</tr>
<tr>
<td>0</td>
<td>250</td>
</tr>
<tr>
<td>1</td>
<td>102</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>3</td>
<td>103</td>
</tr>
<tr>
<td>4</td>
<td>350</td>
</tr>
<tr>
<td>5</td>
<td>351</td>
</tr>
<tr>
<td>6</td>
<td>542B</td>
</tr>
<tr>
<td>7</td>
<td>552</td>
</tr>
</tbody>
</table>

Red: Different Day Professor Swap

Number of mutations before converging | Non deterministic
Fitness score of initial schedule     | 84.38
Fitness score of final schedule       | 96.88

4.1.4 Example 4.

Sample schedule 4 is consists of the following data.

- 6 Courses
• 4 Instructors
• 2 Classrooms

• Classroom 1 is available from 8 am till 2 pm and contains chalk board in it.
• Classroom 2 is available from 8 am till 7 pm and contains white board in it.

• **Fixed Constraints:** Suppose each of the professors have the following time preferences.

<table>
<thead>
<tr>
<th>Professor</th>
<th>MW</th>
<th>TR</th>
<th>MWF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
</tr>
<tr>
<td>$P_1$</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
</tr>
<tr>
<td>$P_2$</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
</tr>
<tr>
<td>$P_3$</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
<td>08:00 - 12:00</td>
</tr>
</tbody>
</table>

• **Hard Constraints:** Suppose each of the professors has the following course preferences.

1. One hundred level preferences

<table>
<thead>
<tr>
<th>Professor</th>
<th>1st Ranked Course</th>
<th>2nd Ranked Course</th>
<th>3rd Ranked Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>Math 102</td>
<td>Math 103</td>
<td>Math 104</td>
</tr>
<tr>
<td>$P_1$</td>
<td>Math 102</td>
<td>Math 103</td>
<td>Math 104</td>
</tr>
<tr>
<td>$P_2$</td>
<td>Math 102</td>
<td>Math 103</td>
<td>Math 104</td>
</tr>
<tr>
<td>$P_3$</td>
<td>Math 102</td>
<td>Math 103</td>
<td>Math 104</td>
</tr>
</tbody>
</table>

2. Two hundred level course preferences

<table>
<thead>
<tr>
<th>Professor</th>
<th>1st Ranked Course</th>
<th>2nd Ranked Course</th>
<th>3rd Ranked Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>Math 250</td>
<td>Math 262</td>
<td>None</td>
</tr>
<tr>
<td>$P_1$</td>
<td>Math 250</td>
<td>Math 262</td>
<td>None</td>
</tr>
<tr>
<td>$P_2$</td>
<td>Math 250</td>
<td>Math 262</td>
<td>None</td>
</tr>
<tr>
<td>$P_3$</td>
<td>Math 262</td>
<td>None</td>
<td>Math 250</td>
</tr>
</tbody>
</table>

3. **Soft Constraints:** Suppose each of the professor has the following additional preferences.

<table>
<thead>
<tr>
<th>Professor</th>
<th>Back to Back</th>
<th>Board Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>True</td>
<td>Chalk</td>
</tr>
<tr>
<td>$P_1$</td>
<td>False</td>
<td>Chalk</td>
</tr>
<tr>
<td>$P_2$</td>
<td>True</td>
<td>None</td>
</tr>
<tr>
<td>$P_3$</td>
<td>False</td>
<td>None</td>
</tr>
</tbody>
</table>
In example 3, $P_2$ ranked Math 552 as his or her first priority. However, instead of teaching Math 552, he or she was teaching Math 542b initially, which was second ranked. At the same time, $P_0$ ranked Math 542b as his or her first priority. However, instead of teaching Math 542b, he or she was teaching Math 552 initially, which was second ranked.

As different day professor mutation operator occurred, the above issue was taken care of by swapping $P_2$ with $P_0$. Therefore, we were able to achieve the most feasible solution to this example.

### Results of Example 4

<table>
<thead>
<tr>
<th>Initial Schedule</th>
<th>Final Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sect</td>
<td>Course</td>
</tr>
<tr>
<td>0</td>
<td>102</td>
</tr>
<tr>
<td>1</td>
<td>250</td>
</tr>
<tr>
<td>2</td>
<td>103</td>
</tr>
<tr>
<td>3</td>
<td>255A</td>
</tr>
<tr>
<td>4</td>
<td>104</td>
</tr>
<tr>
<td>5</td>
<td>103</td>
</tr>
<tr>
<td>6</td>
<td>262</td>
</tr>
<tr>
<td>7</td>
<td>250</td>
</tr>
</tbody>
</table>

**Red:** Different Day Course Swap  
**Blue:** Different Day Professor Swap

Number of mutations before converging | Non deterministic  
Fitness value of initial schedule | 81.25  
Fitness value of final schedule | 103.13

Example 4 was little more complex than previous examples. For instance, $P_2$ was assigned Math 103 on MW and Math 262 on TR initially. Both of the courses were secondly ranked on his preference list. At the same time $P_3$ was assigned Math 250, which was ranked third on his or her preference list. After SA algorithm was applied to this initial schedule, two of the mutation operators occurred. we were able to achieve most feasible schedule.

### 4.2 Analysis of Examples

- By using SA, we were able to find feasible solutions for the above examples. We tried to test how many iterations it would take to converge to the feasible schedule. Sometimes, it took 20 iterations and sometimes the program required 500 iterations.

- Different test data will affect our results. For instance, if the initial schedule was designed that did not satisfy many of professors preferences, then for the final schedule we would get higher fitness value compared to a better designed initial schedule.

- Also, depending on the situation, different mutation operators take place. For example, if professors wanted their ranked courses, then then the professor swap mutation would occur more frequently than any other operations.
4.3 Future Work

1. Large section Course preference still needs to be implemented. One can treat this to be an individual problem.

2. Courses like, math 140 still needs to be implemented.

3. Initial schedule generator needs to be implemented.

4. For back to back preference, there needs to be 15 minute difference implemented.

5. One may want to implement the probability test to compare the schedules.
References


