The Pennsylvania State University
The Graduate School

COURSE SCHEDULING WITH PREFERENCE OPTIMIZATION

A Thesis in
Computer Science
by
Siddharth Dahiya

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2015

The thesis of Siddharth Dahiya was reviewed and approved* by the following:

Thang N. Bui
Associate Professor of Computer Science
Chair, Mathematics and Computer Science Programs
Thesis Advisor

Omar El Ariss
Assistant Professor of Computer Science

Jeremy J. Blum
Associate Professor of Computer Science

Sukmoon Chang
Associate Professor of Computer Science

Linda M. Null
Associate Professor of Computer Science
Graduate Coordinator

*Signatures are on file in the Graduate School.

Abstract

The university course timetabling problem is a well-researched NP-hard problem where the goal is to create a course timetable with a given number of professors, courses, and time slots. There are certain constraints that need to be followed to ensure that professors are not over booked and that courses, which may be scheduled by a student in the same semester, are not scheduled during overlapping time slots. A new perspective of preferences for courses and professors is presented and upon it a new problem is introduced called course scheduling with preference optimization. The focus of this problem is, given sets of courses, professors, time slots, and preferences in terms of time of day for each professor and each course, a schedule is generated where maximum possible preferences are satisfied. To solve this problem, a hybrid genetic algorithm called Course Scheduling Algorithm is also presented. The Course Scheduling Algorithm returns a schedule where the preferences for professors and courses are maximized, and the difference between the number of credits that may be assigned to a professor and the number of credits actually assigned to a professor is kept to a minimum.

Table of Contents

List of Figures

List of Algorithms

List of Tables

Acknowledgements

In Star Wars, Yoda is the key for Luke's success in the rebellion against the Empire. I would like to thank my Yoda, Dr. Thang N. Bui, for guiding me as I learned about Genetic Algorithms and for being the one to destroy the walls when I hit them. I will always be grateful to him for ensuring that I always had my I's dotted and my T's crossed, and for fixing my grammar and spellings when I failed (which was quite frequent). I also want to thank my thesis committee; without their support and feedback, I would not have been able to perfect and improve my work to the current level. A special note of thanks to Dr. Linda Null for going through my thesis and utilizing her English skills to improve my writing.

For the regular supply of wonderful brownies and ensuring that I worked 'distraction free' on my thesis, I want to thank Lisa Hipkins. A hearty thanks also goes to my friend Tyler Derr, for always being present for me to bounce ideas off. Sonam Gupta, Janelle Heiserman, and Jason Reynolds, thank you for helping me with my grammar. Writing a thesis can get very frustrating at times. A note of appreciation is also a must for Viplav Patel and Thomas Zanis for joking around with me and calming me down, more times than I should mention, and Isaac Polinsky for just listening to my ramblings when I got frustrated.

I want to acknowledge the support I got from my other friends and family. Thank you, all of you. And last, but not the least, I want to acknowledge the Imgur community. They've made me laugh when I needed it the most.

Dedicated to my parents,
Shri Bhagwan Dahiya and Suprabha Dahiya.

Chapter 1

Introduction

The timetabling problem is the problem of allocating events, while abiding by constraints, into a small number of time slots with the aim of optimizing a set of goals. The university course timetabling problem (UTP) is an interesting and well researched [1, 3, 4, 8, 10, 11, 12, 13, 14] version of the timetabling problem that is tackled by educational institutions on a continuous basis. In this NP-hard problem [4, 7, 8], the goal is to create a course timetable by using a given number of professors, courses and time slots. Some of the general constraints are preventing professors from being scheduled for more than one course during overlapping time slots and preventing scheduling of courses that can be taken by a student simultaneously in overlapping time slots.

A common factor faced while solving UTP is the availability of faculty. For example, some professors prefer to teach early in the morning, while others are limited to working in the evenings. Another common factor is the time of the day when certain courses are scheduled. An example would be that universities sometimes prefer to schedule certain classes in the evening to accommodate students who work full-time during the day.

Although the above mentioned factors are useful in determining if a given schedule is good or not, they are not the sole deciding factors. For any given schedule, the basic constraints that must be satisfied include: courses are taught by the correct professors, courses are scheduled for the correct duration, and, as mentioned earlier,

professors are not scheduled for multiple courses at the same time.

When determining good schedules, the *preference* of time of the day at which professors are available and courses should be scheduled provides a new perspective. A good schedule in this perspective would schedule courses at times that match the highest preference for professors, as well as the preference for courses themselves. With the goal to find such good schedules, the new problem of *course scheduling with preference optimization* (CSPOp) is presented in this thesis. In this problem, given a set of courses, professors, time slots, and preferences for each course and each professor, the objective is to generate a schedule such that the time assignment for each course and each professor is in accordance with the preferences provided for them. Each course and each professor assign a unique preference value to morning, afternoon, and evening. The preference value that is assigned to each time of the day is chosen from the list of *not available*, *least preferred*, *preferred*, and *highly preferred*. It is also possible that a professor or a course may have no preference for time of day, in which case all three times of day, morning, afternoon, and evening, are given a value of *no preference*.

To solve the course scheduling with preference optimization problem, a Course Scheduling Algorithm (CSA) is presented. CSA is the first of its kind to focus on both professor and course preferences relative to the time of the day, namely morning, afternoon, and evening. CSA is a hybrid genetic algorithm, designed to improve the candidate schedule, that performs an additional step of local optimization using the preferences provided for the professors and courses during the evolution phase. CSA takes for input a set of courses, a set of professors, a set of time slots, a set of preferences for professors, a set of preferences for the courses, an initial schedule, a maximum fitness value, and the credit limit tolerance. For each course, a list of courses that cannot be scheduled concurrently is provided. Similarly, for each professor, a limit of the number of credits that may be assigned to the professor is provided.

To test CSA, an extensive list of instances were used with varying parameters. One of the parameters included is sets of professors and courses of varying sizes. Let $\kappa(c)$ represent the credit value for a course c and let $\pi(p)$ represent the credit limit for a professor P. A related parameter is the relationship between the sum of credit values for each course c in the set of courses $\mathcal{C}$, $\sum_{c \in \mathcal{C}} \kappa(c)$ and the sum of the

number of credits available for each professor p in the set of professors $\mathcal{P}$, $\sum_{p \in \mathcal{P}} \pi(p)$ to schedule. This relationship varies from being overload, where $\sum_{c \in \mathcal{C}} \kappa(c) > \sum_{p \in \mathcal{P}} \pi(p)$, to being underloaded, where $\sum_{c \in \mathcal{C}} \kappa(c) < \sum_{p \in \mathcal{P}} \pi(p)$. It equalizes when both of these credit values are equal, creating a balanced input. Another parameter for the instance is the initial schedule that was provided. The resultant schedules have been found to extensively accommodate the given preferences.

The thesis is organized as follows. Chapter 2 presents a formal definition of the CSPOp problem, a brief summary of genetic algorithms, and other work related to timetabling problems. The CSA algorithm, its input, output, and inner workings are described in Chapter 3. In Chapter 4, the results of this algorithm are analyzed in detail with different sections for each combination of input parameters. Finally, Chapter 5 presents the conclusion of the work along with ideas for future work.

Chapter 2

Problem Description

In this chapter, the Course Scheduling with Preference Optimization (CSPOp) problem is presented in its general form. Also provided is terminology for the parameters and constraints of the problem. Since this thesis uses a hybrid genetic algorithm, a brief overview of genetic algorithms is also presented in this chapter.

2.1 General Formulation of CSPOp

In this thesis, a *time slot* is a period of time that is used to represent the weekly meeting time for courses over the span of one or more days. The time represented by the time slot may also vary for each day. A time slot can be categorized into morning, afternoon, or evening, based on the general start and end time of the meetings. For example, a time slot representing a meeting time of Monday, Wednesday, Friday from 10:00 AM to 10:50 AM is considered morning, while another time slot representing the meeting time Wednesday from 6:00 PM to 8:45 PM is considered evening. If a time slot spans multiple categories, the start time of the time slot is used to categorize it. For example, a time slot representing a meeting time on Monday from 11:30 AM to 1:00 PM is be categorized as morning based on its starting time. The total number of hours in a time slot corresponds to the number of credit hours it represents.

The understanding of time slot is important, as the time of day by which a time slot is classified plays a crucial role for the preferences of both professors and courses. Time slots also play an important role in ensuring *incompatible* courses are scheduled separately. Two courses are considered *incompatible* if they cannot be

scheduled during overlapping time slots.

In the course scheduling with preference optimization problem, each course and each professor has a set of time of day preferences, and the goal is to schedule them in time slots that best fit the given preferences. The set of preferences identifies the time of day that has the highest preference, the time of day with normal preference, and the time of day with the least preference. The input for CSPOp is a collection of courses, a collection of time slots, a collection of professors, a list of preferences of the time of day for each professor and each course, and an initial schedule to use, which may be empty, partial, or complete. Since the schedule of courses do not change much between semesters, a schedule from a previous semester can be passed as an initial schedule to help course scheduling algorithm in generating an initial population of the genetic algorithm.

All courses in the collection contain information about the course, including the credit value, the list of incompatible courses to avoid, and a list of professors that can teach the course. It is possible that a course may have multiple sections, but for generality this thesis considers each section as an individual course. Similarly, each professor in the collection of professors contains the number of credits the professor may be scheduled for and the list of courses the professor can teach. Each time slot in the input contains information regarding the course credit value that may be scheduled, the time of the day they fall in, and other time slots with which it conflicts.

When this information is provided to the course scheduling algorithm, it returns a valid schedule where courses and professors are scheduled at time slots that best match the preferences they set, while staying within the limitations imposed on the schedule. The next section describes some terminology used in this thesis.

2.1.1 Terminology

Every schedule must satisfy a set of constraints. A constraint is said to be *hard* if it is required to be satisfied to ensure the validity of the schedule. An example of a hard constraint would be not scheduling a professor for two separate courses during overlapping time slots. Contrary to hard constraints, a *soft* constraint is an optional limitation that hints at how desirable a schedule is. The more soft constraints are

satisfied, the better the schedule is. Using these definitions, a *valid* schedule is one that satisfies all hard constraints. Violating a hard constraint renders a schedule *invalid*.

Another important hard constraint is avoiding scheduling two incompatible courses at the same time. Two courses are *incompatible* if a student is allowed to schedule both of these courses during a given semester. On another note, each professor has a *credit limit* which is used to represent the number of credits for which professor should be scheduled. However, to allow for some flexibility in scheduling courses, a *tolerance* value of Δ is used. The value of Δ may be 0 or more. Scheduling a professor for credits that are outside of Δ range of the credit limit renders a schedule invalid. A professor is considered to be *overloaded* if the number of credits the professor is scheduled for exceeds the credit limit within the tolerance. Similarly, a professor is considered to be *underloaded* if the professor is scheduled for fewer credits than the credit limit.

2.1.2 Problem Constraints

For CSPOp, the hard constraints and soft constraints are as follows.

Hard Constraints

1. No two incompatible courses may be scheduled during overlapping time slots.
2. No professor may be scheduled for two different courses during overlapping time slots.
3. No professor may be assigned courses more or less than Δ of their respective credit limit.
4. Each course must be assigned to a professor that can teach it.
5. All courses must be scheduled at a time slot that has the same number of credit hours as required by the course.

Soft Constraints

1. A professor should be scheduled for the exact number of credits as stated in the limits.
2. A course should be scheduled in a time slot that falls in the highest preference category for the course.
3. A professor should be scheduled at a time slot that falls in the highest preference category for the professor.
4. A professor should not be scheduled for more than two consecutive time slots. Two time slots are *consecutive* if, for any day of the week in which they have both occur, their start and end times differ by 15 minutes or less.
5. A professor should not be scheduled for two time slots where one time slot is classified as a morning time slot while the other is classified as evening on the same day of the week.

2.1.3 Course Scheduling with Preference Optimization

The formal input and output for the Course Scheduling with Preference Optimization problem is as follows.

Input:

- $\mathcal{C} = \{c_1, \ldots, c_d\}$ is a set of d courses,
- $\mathcal{P} = \{p_1, \ldots, p_e\}$ is a set of e professors, and
- $\mathcal{T} = \{t_1, \ldots, t_f\}$ is a set of f time slots.
- $\Gamma = \{\gamma_1, \ldots, \gamma_e\}$ is the set of preferences for the professors. For each professor, γ represents a 3-tuple that gives a weight for morning, afternoon, and evening preferences. Each weight is a number from the set $\{0, 1, 2\}$ with 2 representing the weight for the most preferred time. Hence, for a professor whose most preferred time is morning, then the afternoon, and the least preferred time is evening, γ is represented as $\gamma = (2, 1, 0)$.
- $\Upsilon = \{\upsilon_1, \ldots, \upsilon_d\}$ is the set of preferences for the courses. For each course, υ represents a 3-tuple that gives a weight for morning, afternoon, and

evening preferences, similar to γ for professor. Each weight is a number from the set $\{0, 1, 2\}$ with 2 representing the weight for the most preferred time. Hence, for a course that is most preferred time is in the evening, then the morning, and the least preferred time is in the afternoon, then υ is represented as $\upsilon = (1, 0, 2)$.

- $\omega = \{(c_1, p_x, t_y), \ldots, (c_z, p_u, t_w)\}$ is an initial schedule, that may be a complete, partial, or empty schedule, containing 3-tuples of the form (c_i, p_j, t_k) representing that the course c_i is assigned professor p_j in time slot t_k.
- The credit function, $\kappa(c)$ where $c \in \mathcal{C}$ is used to represent the credit values for courses, and $\kappa(t)$ where $t \in \mathcal{T}$ represent the credit values for time slots.
- The function $\pi(p)$ where $p \in \mathcal{P}$ represents the credit limit for a professor.
- Given two courses, $c, c' \in \mathcal{C}$, the incompatibility function, $\psi(c, c')$, is true if c and c' are incompatible, and false otherwise.
- The preference satisfaction function, $\sigma(c, t)$ where $c \in \mathcal{C}$ and $t \in \mathcal{T}$, represents the satisfaction of the time preference of c by time slot t. If t is the most preferred time for c, then $\sigma(c, t) = 2$; if t is the least preferred time for c, then $\sigma(c, t) = 0$; and if t is the normal preferred time for c, then $\sigma(c, t) = 1$.
- Similarly, $\sigma(p, t)$ where $p \in \mathcal{P}$ and $t \in \mathcal{T}$, represents the satisfaction of the time preference of p by time slot t. If t is the most preferred time for p, then $\sigma(p, t) = 2$; if t is the least preferred time for p, then $\sigma(p, t) = 0$; and if t is the normal preferred time for p, then $\sigma(p, t) = 1$.
- The credit limit tolerance value, Δ.
- The maximum fitness value, f_{max}.

Output:

A valid schedule, $\omega = \{(c_1, p_x, t_y), \ldots, (c_d, p_u, t_w)\}$ such that $\sum_{c \in \mathcal{C}, t \in \mathcal{T}} \sigma(c, t)$ and $\sum_{p \in \mathcal{P}, t \in \mathcal{T}} \sigma(p, t)$ are maximized. Let $\overline{\pi}(p)$ represent the number of credits assigned to $p \in \mathcal{P}$. Then, the value of $|\pi(p) - \overline{\pi}(p)|$, $\forall p \in \mathcal{P}$ is also minimized.

2.2 Background

In this section, a brief overview of Genetic Algorithms (GA) is presented. This provides a general understanding of the working of genetic algorithms. This is followed by a brief summary of some work in the field of University Timetabling Problems that are related to this thesis.

2.2.1 Genetic Algorithms

Genetic algorithms were first introduced by John Holland at the University of Michigan in 1975 [9]. In its general form, a GA mimics Darwinian evolution. It consists of problems encoding, selection methods, and genetic operators such as crossover and mutation. A GA typically has four stages; selection, recombination (crossover), mutation, and replacement. Algorithm 2.1 is an example of a general genetic algorithm [6].

---

begin
  INITIALIZE population p with random candidate solution EVALUATE each candidate **repeat**
    SELECT parents from p
    CROSSOVER pairs of parents to produce offspring
    MUTATE offspring
    EVALUATE the offspring
    REPLACEMENT of an individual in p by selecting individual among the current individuals and the new offspring to form the population for the next generations
  until *TERMINATION CONDITION* is satisfied;

Algorithm 2.1: Simple Genetic Algorithm

First, a population of randomly generated solutions is formed. Then, two parents are selected using a selection function such as random selection, fitness proportional selection, ranking selection, or tournament selection [6]. After parents have been selected, they are recombined to produce offspring. There are different crossover methods that may be applied such as, k-point crossover, uniform crossover, partially mapped crossover, edge crossover, cycle crossover, etc. [6]. The crossover operation then produces one or more children that can now be mutated using any one of the

various mutation operations. Once a mutated offspring is obtained, it is compared with the existing solutions in the population. If the offspring is better than an individual in the population, it replaces that individual and joins the population in the replacement phase.

The steps from selection to replacement are repeated until the stopping criteria are satisfied. At that point, the loop is exited and the best individual of the population is returned. This individual represents the best solution that was discovered by the algorithm.

As mentioned earlier, this is a very simple GA. This algorithm can further be improved to find an even better solution by introducing a local optimization method before replacement. The local optimization method ensures that the child is as good as it can be before it is introduced to the population. Also, note that at every iteration of the evolution, only one individual is being replaced. This type of GA is referred to as a *steady state* genetic algorithm.

2.2.2 Related Work

As mentioned earlier, the University Timetabling Problem has been the focus of a number of studies in the past. In this section, some of those studies are discussed to provide background information on UTP.

In 1998, Carter and Laporte [4] presented an overview of algorithms involved in course timetabling. They listed several papers that discussed solving the timetabling problem at that time. They did note, however, that during the 1960s and 1970s, there was a large number of implementations of algorithms to solve the problem. However, only a few new methods had been implemented and were being used by institutions.

The next year, in 1999, Burke and Newall [3] used a multistage evolutionary algorithm to tackle the timetable problem. They broke the difficult timetabling problem into phases to reduce search time to find a solution, as well as to improve the solution. The method was a hybrid of heuristic sequencing and evolutionary methods that was able to outperform heuristic sequencing and evolutionary methods individually. It worked by taking the most difficult event and searching for the best placement for it with respect to other events. They repeated this process, focusing on one event at a time, in decreasing order of difficulty. They also implemented a

look-ahead approach to ensure that scheduling an event would not prevent future events from being placed in a valid time slot.

Subsequently, a genetic algorithm to solve the timetabling problems was presented by Sigl, Golub and Mornar [12]. The encoding they used consisted of a list of binary variables, x_{cdtrgi}, which represented instructor i teaching the class c in room r for group g on day d and time t. They also presented a graphical user interface that allowed for input of days, times, rooms for class placement, number of rooms required by a class, groups of students attending the class, and the instructors that could teach the class. They were only able to solve small scale problems and concluded that further algorithmic improvements were required to solve the full scale problem.

Another genetic algorithm to solve course scheduling problems was presented the same year by Wang [14]. This paper presented a new perspective of professor preference when scheduling courses as well as the minimum and maximum number of hours a professor can be assigned. The author demonstrated that GAs could significantly reduce the time spent on scheduling as well as improve teaching quality by taking professor preference into account.

Lastly, Turabieh and Abdulla [13] examined the timetabling problem. They proposed a hybrid approach to solve the problem by incorporating heuristic operators with the Great Deluge Algorithm. The Great Deluge algorithm introduced by Dueck [5], is an algorithm applied to optimization problems similar to simulated annealing. In their paper, the authors attempted to avoid the local optima by calculating a decay rate in determining the level within the Great Deluge Algorithm, and moving the sample points towards a high quality solution. They divided their problem into two parts. The first part did not consider the room capacity requirement while constructing a timetable. In the second part they took into account many real world constraints including room capacity. They found that their approach produced the best results compared to others from the literature, and that their approach could be easily adapted with new constraints.

All of these papers presented a different approach to tackling the timetabling problem. However, none of these papers focused on both professor and course preferences from the time of day perspective, as is being done by the CSPOp problem. Because of this difference, we are not able to compare the results given by the course scheduling algorithm discussed here. In the next chapter, the the course scheduling

algorithm is described for solving the CSPOp problem.

Chapter 3

Course Scheduling Algorithm

This chapter discusses the proposed algorithm to solve CSPOp, the Course Scheduling Algorithm (CSA). All of the inner workings of the algorithm are described, along with explanations of any salient features.

3.1 Encoding

One of the biggest tasks to tackle when using any genetic algorithm is to have a good encoding of the solution for the individuals comprising the population. In CSA, the solution is a schedule, $\omega = \{(c, p, t) \mid \forall c \in \mathcal{C},\, p \in \mathcal{P},\, t \in \mathcal{T}\}$. For the schedule comprising this set, there are three possible indices to identify each element: time slot, professor, and course.

c_1	c_2	c_3		c_{d-1}	c_d
(p_a, t_x)	(p_b, t_y)	(p_c, t_z)	.....	$(p_{\alpha-1}, t_{\beta-1})$	(p_α, t_β)

Figure 3.1: CSA Encoding

CSA uses an array-based encoding with *course* as an index. The array element consists of pairs of professors and time slots. To facilitate easy processing of data, courses, professors, and time slots are all represented by numerical identifiers. Hence, courses are numbered from 1 to d, professors are numbered 1 to e, and time slots are numbered from 1 to f.

3.2 Population Generation

The input for CSA contains a schedule ω_i that may be provided to help in the initialization of the population, Ω. The given schedule, ω_i, may be one of the following.

- *Complete Schedule*, where a professor and time slot pair is provided for each course. This initial schedule may or may not be valid.
- *Partial Schedule*, where professor and time slot pairs are provided for some of the courses. The partial schedule may be valid or invalid.
- *Empty Schedule*, where no course is provided with a professor and time slot pair.

If ω_i is a *Complete Schedule*, the first individual schedule of the population is created using ω_i. Then, all of the other individual schedules of the population are generated by replicating the first individual, and mutating and repairing the resultant individual schedule. In the instance where ω_i is a *Partial Schedule*, the first individual schedule of the population is again initialized using ω_i. The difference is that a compatible professor and time slot are assigned at random to the courses that do not have any assigned. The algorithm proceeds to generate the population as if ω_i were a complete schedule as described before.

Finally, if ω_i is empty, then all individual schedules that form Ω are randomly generated by assigning each individual schedule a random professor and time slot for each course. These individuals are neither mutated nor repaired. This form of initialization of the initial population allows CSA to account for various real-world scenarios and handle them accordingly. Some of these scenarios may be an increase in the number of sections for a course, an introduction of a new faculty member, the removal of a previous faculty member, changing of time slots for courses, etc. Note that the initial population, Ω, may contain both valid and invalid solutions.

3.3 Fitness Function

The fitness function, $F(\omega)$, where $\omega \in \Omega$, is one of the most crucial functions of CSA because it provides a way to differentiate between a good and a bad solution. The fitness function uses the hard and soft constraints as parameters where each

constraint is assigned a weight value. The larger the fitness, the better an individual is considered. The parameters and the weights are defined in Table 3.1. The fitness function uses these weights and a starting weight f_{max} to calculate the fitness using Eq. (3.1).

$$F(\omega) = \underbrace{f_{max}}_{(1)} - \underbrace{\sum_{i=1}^{m} (Penalty(H_i) \times Count(H_i))}_{(2)} - \underbrace{\sum_{j=1}^{n} (Penalty(S_i) \times Count(S_i))}_{(3)} \quad (3.1)$$

Where,

- $F(\omega)$ is the fitness for schedule ω,
- m is the number of hard constraints, n is the number of soft constraints,
- f_{max} is the maximum fitness,
- $Penalty(H_i)$ is the penalty for the hard constraint H_i, $1 \leq i \leq m$,
- $Count(H_i)$ is the number of violations of the hard constraint H_i, $1 \leq i \leq m$,
- $Penalty(S_j)$ is the penalty for the soft constraint S_i, $1 \leq j \leq n$,
- $Count(S_j)$ is the number of violations of the soft constraint S_i, $1 \leq j \leq n$.

For a schedule ω, the fitness is first assigned a maximum fitness value f_{max} in part (1) of Eq. (3.1). This is the benchmark fitness for the optimal solution that CSA aims to obtain. Then the penalty for all the hard constraints is computed as shown in part (2) of Eq. 3.1. This is followed by the computation of the penalty for all the soft constraints as shown in part (3) of Eq. 3.1. Finally, parts (2) and (3) are subtracted from the maximum fitness in part (1) to obtain the fitness of the schedule.

As mentioned in Table 3.1, one of the constraints on the schedules is to schedule professors with the proper number of credits, as dictated by their credit limit. However, to provide CSA with some flexibility in moving courses among professors, a Δ value is used to allow tolerance to the violation of this rule. Then each professor is required to be scheduled such that $|\pi(p) - \overline{\pi}(p)| \leq \Delta$, where, $\overline{\pi}(p)$ is the actual number of credits assigned to the professor. That is, each professor's credit load must be within a Δ value of the credit limit, $\pi(p)$. In terms of penalty, $0 < |\pi(p) - \overline{\pi}(p)| \leq \Delta$ is considered to be a soft constraint violation, while $|\pi(p) - \overline{\pi}(p)| > \Delta$ is considered to be a violation of a hard constraint.

Parameter	Constraint	Penalty
H_1	No two incompatible courses may be scheduled at the same time.	10
H_2	No professor may be scheduled for two different courses at the same time.	10
H_3	Professor's credit constraint must be met within tolerance	20
S_1	A professor should be scheduled for the exact number of credits as stated in the limits.	5
S_2	A course should not be scheduled at a time slot that falls in the least preference category for the course.	10
S_3	A professor should not be scheduled at a time slot that falls in the least preference category for the professor.	10
S_4	A course should not be scheduled at a time slot that falls in the normal preference category for the course.	5
S_5	A professor should not be scheduled at a time slot that falls in the normal preference category for the professor.	5
S_6	A professor should not be scheduled for more than two consecutive time slots.	5
S_7	A professor should not be scheduled for spread-out time slots. Time slots are considered to be *spread-out* if one is considered a morning time slot, while the other is an evening time slot.	5

Table 3.1: Constraint Parameters and Their Weights

The Course Scheduling Algorithm was tested with different constraint penalties. The penalty values listed in Table 3.1 are the values that were able to distinguish between the schedules the most.

3.4 CSA Overview

The Course Scheduling Algorithm (CSA) follows the general structure of a hybrid genetic algorithm. The pseudocode presented in Algorithm 3.1 performs the operations on the population Ω of μ schedules, with the input:

- $\mathcal{C}$, set of d courses,

- $\mathcal{P}$, set of e professors,
- $\mathcal{T}$, set of f time slots,
- Γ, set of preferences for the professors,
- Υ, set of preferences for the courses,
- ω_i, an initial schedule,
- f_{max}, the maximum fitness value,
- Δ, the credit limit tolerance,
- μ, the number of schedules in the population,
- g, number of generations for evolution, and
- m_p, the mutation probability.

Input: $\langle \mathcal{C}, \mathcal{P}, \mathcal{T}, \Gamma, \Upsilon, \omega_i, f_{max}, \Delta, \mu, g, m_p \rangle$
Output: $\omega^\star$, the best schedule in the population
if $\omega_i \neq$ NULL **then**
  $\Omega \longleftarrow$ `GeneratePopulation`(ω, μ)
else
  $\Omega \longleftarrow$ `GeneratePopulation`$(\mathcal{C}, \mathcal{P}, \mathcal{T}, \mu)$
while currentGeneration $\leq g$ ***or*** lastReplacement $<$ MaximumWait **do**
  $\omega_{parent} \longleftarrow$ `RandomSelection`(Ω)
  $\omega_{child} \longleftarrow \omega_{parent}$
  `ScheduleMutation`(ω_{child})
  `ScheduleRepair`(ω_{child})
  `LocalOptimization`(ω_{child})
  `Replace`$(\omega_{parent}, \omega_{child}, \Omega)$
  currentGeneration ++
`/* Find the best schedule in the population,` $\omega^\star$ `*/`
$\omega^\star \longleftarrow$ `Best`(Ω)
return $\omega^\star$

Algorithm 3.1: CSA Overview

CSA starts by generating an initial population, Ω. The initial population is based on the initial schedule, ω_i, if the initial schedule is not empty, and random if it is. This is then followed by the evolution of the population. In the evolution phase,

CSA attempts to improve the population, as shown in Algorithm 3.1. It begins by selecting a random `parent` from the population using fitness scaling based on Eq. (3.2). This equation is based on the weighted scaling equation presented in [2].

$$F_i = (F_i - F_w) + (F_b - F_w)/3, \tag{3.2}$$

where,

- F_w: fitness of the worst individual of the population,
- F_b: fitness of the best individual of the population,
- F_i: fitness of the current individual.

A copy of the `parent` is created as the `child`. The `child` is mutated by using Algorithm 3.2. At this point, the `child` might be invalid. Hence, the mutated individual is then repaired using Algorithms 3.3 and 3.4. To improve the `child` even further, it is optimized using Algorithm 3.5. Finally, using Algorithm 3.6, CSA attempts to introduce the `child` into the population by replacing an existing schedule in the population, and continue the evolution of the population. The evolution phase of CSA ends when either no new schedule has been introduced into the population for a `MaximumWait` period, or if the algorithm has run for the maximum number of generations. At the end, the best schedule is found in the population by performing a search for the valid schedule with the highest fitness and is returned by the algorithm. In case no valid schedules are found, the algorithm returns the schedule with the highest fitness.

The mutation probability is decremented by 1% for every $\frac{g}{(m_p \times 100)}$ generations, where m_p is the mutation probability and g is the total number of generations. The period of $\frac{g}{(m_p \times 100)}$ generations allows for a consistent decrease based on the mutation probability. With the assumption that CSA is nearing an optimal solution, this reduction in mutation probability ensures that the mutation is decreased near the end of evolution and CSA focuses on improving the best solution.

3.5 Schedule Mutation

In Algorithm 3.2, the schedule ω is mutated on each course c with a probability of m_p. The algorithm begins by selecting a random number uniformly in $(0, 1)$. If the selected number is less than m_p, a new time slot $t' \in \mathcal{T}$, such that it is not the time

slot currently assigned to c and t' belongs to the list of time slots during which c can be scheduled. Then the current time slot for the course is changed to this new time slot, t'. By selecting t' from the list of possible time slots, a course may be restricted to certain time slots as well. If t' causes any conflicts, then a conflict-free t' is selected from the list of possible time slots by performing an iterative search and selecting the first time slot that is conflict-free. This ensures that the mutation performs its necessary functions while being efficient in terms of time.

Similarly, another random number is uniformly selected from $(0, 1)$. If the selected number is less than m_p, a new professor $p' \in \mathcal{P}$ such that if p is the currently assigned professor, then $p \neq p'$, is randomly selected from the list of compatible professors for the given course. Then for the given course, the current professor, p, is replaced with the new professor, p'. Once these two steps are completed, the iteration of the loop continues until there has been an attempt to mutate all of the courses in the schedule.

```
Input: ⟨ω, m_p⟩, Schedule to mutate and mutation probability
Output: ω', Schedule with mutations
ω' ⟵ ω
foreach c ∈ C do
    m ⟵Random(0,1)
    if m ≤ m_p then
        t' ⟵ ListOfPossibleTimeSlots(c)
        if t' causes conflict then
            foreach t' ∈ListOfPossibleTimeSlots(c) do
                /* Find a valid t'.                          */
        ω'.SetTime(c, t')
    m ⟵Random(0,1)
    if m ≤ M_p then
        p' ⟵ ListOfPossibleProfessors(c)
        ω'.SetProfessor(c, p')
return ω'
```

Algorithm 3.2: ScheduleMutation

3.6 Schedule Repair

The repair algorithm attempts to repair a schedule so that there are no hard constraint violations. The Repair Algorithm, Algorithm 3.3, starts by repairing the professor credit loads using the Professor Credit Constraint Repair Algorithm, Algorithm 3.4. Once all of the professor assigned credit loads have been repaired, it ensures that the professor for a given course is not changed. This helps in ensuring that the professors do not get assigned a course load that is not on par with their credit limits.

3.6.1 Time Slot Constraint Repair

The time slot conflict resolution phase starts by iterating through all of the courses in the schedule ω, using the index c to identify the first course and then iterating again with the index, c', to identify the second course to compare. Now, if the two selected courses are incompatible, that is $\psi(c, c')$ is true, or if they have the same professors, the time slot allocated to the courses needs to be changed to resolve the conflict. The repair algorithm does this by adding the current time slot of c to the tabu list, and then changing its time slot to an alternate that is not in the tabu list, and may be assigned to c.

It must be noted here that occasionally a schedule has too many constraint violations, making it impossible to repair. In such cases, the repair algorithm attempts to repair a schedule for a set number of tries, referred to in Algorithm 3.3 as Max_Repair_Tries. After the number of repair attempts exceeds this limit, the repair algorithm abandons the attempts to repair that schedule.

3.6.2 Professor Credit Constraint Repair

One of the hard constraints is to ensure that no professor is assigned a credit load that is beyond the Δ tolerance of their credit limits, that is, for each $p \in \mathcal{P}$, $|\pi(p) - \overline{\pi}(p)| \leq \Delta$. If a professor is assigned a course load more than the given credit limit, that professor is considered to be *overloaded*. If the professor's course load is less than the credit limit, the professor is then considered to be *underloaded*. If the schedule contains a professor who has been assigned a credit limit such that $|\pi(p) - \overline{\pi}(p)| > \Delta$,

```
Input: ω, schedule to repair
Output: ω′, repaired schedule
ω′ ⟵ ω
for try = 0 to Max_Repair_Tries do
    if isValid(ω′) then
        break
    ProfessorCreditConstraintRepair(ω′)
    foreach c ∈ C do
        foreach c′ ∈ (C − {c}) do
            if ψ(c, c′)  or  ω′.Professor(c) == ω′.Professor(c′) then
                if ω′.GetTimeSlot(c) == ω′.GetTimeSlot(c′) then
                    AddToTabuList(GetTimeSlot(c))
                    if ∃potentialTimes ∈ (AvailableSlots(c) − TabuList) then
                        ω′.SetTimeSlot(c, potentialTimes)
return ω′
```

Algorithm 3.3: ScheduleRepair

then the schedule is considered to be invalid. In the Professor Credit Constraint Repair Algorithm, Algorithm 3.4, focus is given to balancing assigned credit loads to prevent underloading or overloading professors.

The algorithm iterates through all professors to find a professor $p \in \mathcal{P}$ that is overloaded or underloaded beyond the Δ threshold which makes the schedule invalid. If such a professor p is found, and that professor is overloaded, then a list of professors that may be able to teach a course currently assigned to p is searched to find a professor p' that is underloaded. Then a course that is currently assigned to p is transferred to p', provided p' can teach the course. Similarly, if p is underloaded, then a list of professors that are currently assigned courses that p may teach is searched to find a professor p' that is overloaded. Then a course that is currently assigned to p' is transferred to p, provided p can teach the course. In either case, if such a p' is not found, a depth-first search is done to find a p', that could take a course if p was over-loaded or that could give a course if p was under-loaded. The first p' is selected to ensure that the algorithm does not get into a cycle. Once a p' is found, a course is transferred through the chain of professors. This is repeated until either p has been assigned a valid course load or the list of possible p' has been exhausted.

The algorithm loops through this process until either no professor is causing the

schedule to be invalid during the current iteration of the loop, signifying all the professors have met their respective credit constraints within tolerance, or credit constraints for more professors were repaired in the current iteration than the previous iteration, which would signify that the professor credit constraint violations were increased.

Input : ω, where $\exists p \in \mathcal{P}$ such that $|\pi(p) - \overline{\pi}(p)| > \Delta$
Output: ω', such that $\forall p \in \mathcal{P}, |\pi(p) - \overline{\pi}(p)| \leq \Delta$

$\omega' \longleftarrow \omega$
while previouslyRepaired $>$ currentlyRepaired ***and*** currentlyRepaired $\neq 0$ **do**
    previouslyRepaired $\leftarrow$ currentlyRepaired
    currentlyRepaired $\leftarrow 0$
    for $p \in \mathcal{P}$ **do**
        if $|\pi(\omega'.p) - \overline{\pi}(\omega'.p)| > \Delta$ **then**
            currentlyRepaired ++
            while $|\pi(\omega'.p) - \overline{\pi}(\omega'.p)| > \Delta$ **do**
                `/* Search to find another professor that can give or take a course, and then shift courses on the path. */`
return ω'

Algorithm 3.4: ProfessorCreditConstraintRepair

3.7 Local Optimization

One of the goals of CSA is to optimize the preferences for professors and courses. As such, the evolution process includes *Local Optimization*. In the Local Optimization Algorithm, Algorithm 3.5, an attempt is made to swap professors between courses to try and maximize the satisfaction of preferences for both professors and courses. The algorithm starts by iterating through all the courses, using `course` as the index to identify the current course, c. If there is more than one professor that can teach c, and p is the professor currently assigned to teach c, then an alternate professor, p' is selected such that p' can also teach c.

If p' is teaching a course, c', and p can teach c, p and p' are swapped. Let $F(\omega)$ be the fitness prior to the swap and $F'(\omega)$ be the fitness after the swap. If

$F'(\omega) > F(\omega)$, then the optimization for the course c is considered successful. In case $F'(\omega) < F(\omega)$, then the swap is undone. The algorithm tries again to schedule c with a different professor p'. When all possible alternate professors have been tried, c is declared unoptimizable and the local optimization continues to the next course, c'.

```
Input: ω, Unoptimized Schedule
Output: ω', Optimized Schedule
ω' ⟵ ω
for c ∈ 𝒞 do
    if Number of CanTeach(c) > 1 then
        p ← ω'.AssignedProfessor(c)
        for p' ∈ CanTeach(c) do
            if ∃c' ∈ ω'.CurrentlyAssigned(p') such that p can teach c' then
                F(ω) ← ω'.Fitness()
                ω'.SwapProfessor(c, c')
                F'(ω) ← ω'.Fitness()
                if F(ω) > F'(ω) then
                    ω'.SwapProfessor(c', c)
                else
                    break
return ω'
```

Algorithm 3.5: LocalOptimization

3.8 Replacement

Once selection, mutation, repair, and optimization of a child ω_c is complete, it is necessary to check if the child would be a viable replacement for a schedule in the population, Ω. There are two phases of evolution, exploration and exploitation. The goal of evolution, during exploration, is to have a large variety of schedules. In the exploitation phase, the goal changes to improving the schedules. These two phases also determine how the replacement is performed.

During the exploration phase, the population can contain valid and invalid schedules to increase the variety of schedules in the population. First, the child is compared with the parent schedule, ω_p. If the child has a better fitness than the parent, the

parent is replaced by the child. Otherwise, the child is compared with the weakest schedule of the population, ω_w. Again, if the child is better than ω_w, it is replaced with the child. If the child cannot be added to the population, the current generation counter is incremented and the evolution continues.

During the exploitation phase, however, the population is cleared of all invalid schedules. Hence, if the child is invalid, it is discarded and replacement ends with the increment of the generation counter. If the population contains invalid schedules, the child replaces the first invalid schedule it finds. Otherwise, the child replaces the parent if the parent is weaker than the child. In case the parent is better than the child, the child replaces the worst individual, provided the child is better than the worst individual. If the child is not better than the parent or the worst individual, the generation counter is incremented and the evolution continues.

Input: $\langle \Omega, \omega_c, \omega_p \rangle$, Population, Child Schedule and Parent Schedule
Output: Ω', Post Replacement Population
$\Omega' \longleftarrow \Omega$
if *Exploration Phase* **then**
    if $F(\omega_c) > F(\omega_p)$ **then**
        Ω'.`Replace`(ω_p, ω_c)
    else if $F(\omega_c) > F(\omega_w)$ **then**
        Ω'.`Replace`(ω_w, ω_c)
        $\omega_w \longleftarrow$ `FindNewWorst`(Ω')
else if `IsValid`(ω_c) **then**
    if $\exists \omega_{invalid} \in \Omega'$ **and** $F(\omega_c) > F(\omega_{invalid})$ **then**
        Ω'.`Replace`($\omega_{invalid}$, ω_c)
        $\omega_{invalid} \longleftarrow$ `FindNewWorstInvalid`(Ω')
    else if $F(\omega_c) > F(\omega_p)$ **then**
        Ω'.`Replace`(ω_p, ω_c)
    else if $F(\omega_c) > F(\omega_w)$ **then**
        Ω'.`Replace`(ω_w, ω_c)
        $\omega_w \longleftarrow$ `FindNewWorst`(Ω')
return Ω'

Algorithm 3.6: Replacement

Chapter 4

Results

To ensure that the Course Scheduling Algorithm was tested with various types of input, CSA was provided with a set of randomly generated data. This chapters discusses the inputs and the results that were seen with the given inputs.

4.1 Random Input Instances

Let a *data set* consist of a list of professors and their preferences, a list of courses and their preferences, and a list of time slots. Let an *input instance* consist of the following.

- A data set,
- A credit limit tolerance, Δ, and
- a valid initial schedule, ω_i.

The CSA was given 27 unique data sets, 3 unique credit limit tolerance values, $\Delta = \{0, 1, 2\}$, and 3 types of initial schedules (complete schedule, partial schedule, and empty schedule). In the complete initial schedule, there existed an assignment of professor-time slot pair for every course. In the partial initial schedule, only 75% of courses had the professor-time slot pair assigned for every course. Finally, in the empty initial schedule no professor-time slot pairs were assigned to any course. This provided CSA with a total of 243 unique input instances over all. Each instance was run 50 times to get sufficient statistical data. A point to note here is that for all

the data sets, a total of 34 time slots was used. The distribution of credit limit for professors for each data set is available in Table 4.1.

Credit Limit	Percentage of Total Professors
3.0	2%
6.0	8%
9.0	75%
12.0	15%

Table 4.1: Professor Credit Limit Distribution

The data set consisted of three variations of the number of professors: 100, 200, and 300. For each number, there were three different data sets for each relationship between the total credits for courses to schedule and the total credits available for professors in the initial schedules. An *overloaded* initial schedule occurs when the total number of credits for courses to schedule exceeds the total number of credits available for professors to schedule. An *underloaded* initial schedule occurs if the total number of credits for courses to schedules is less than the total number of credits available for professors to schedule. Finally, a *balanced* initial schedule is when the total number of credits for courses to schedule is equal to the total number of credits available for professors to schedule. For each combination of number of professors and credit relationship, three unique data sets were generated. The statistics about the input data sets are discussed in Appendix A.1. Also, the preference level satisfactions for the input data sets are discussed in Appendix A.2.

An overview of the mean penalty per course is shown in Table 4.2. From the table, it can be seen that when CSA was given test instances with $\Delta = 0$, there were no changes in the penalties. With $\Delta = 1$, CSA was able to decrease the penalties per course by 3% to 16%. However, when CSA was given test instances with $\Delta = 2$, significant decrease in penalties, of upto 26%, were shown. Hence, it can be seen that as the Δ value is increased, CSA is able to decrease the penalties per course as well. For each Δ value, the penalties increase with an increase in the number of professors and the number of courses. This increase in penalties can be accredited to the constant number of time slots. With an increase in the number of professors and number of courses, it becomes harder for CSA to be able to improve a schedule as the number of conflicts to avoid is also increased.

Δ Value	# of Professors	Input Type	Mean # Of Courses	Mean Penalty Per Course		
				Input (ω_i)	Output ($\omega^\star$)	Percentage Improvement
$\Delta = 0$	100	Balanced	296	10.38	10.38	0%
	200	Balanced	601	10.02	10.02	0%
	300	Balanced	896	10.11	10.11	0%
$\Delta = 1$	100	Balanced	296	10.38	9.38	10%
		Overload	302	10.22	8.60	16%
	200	Balanced	601	10.02	9.69	3%
		Overload	598	10.59	10.06	5%
	300	Balanced	896	10.11	9.84	3%
		Overload	917	10.33	9.81	5%
$\Delta = 2$	100	Balanced	296	10.38	8.31	20%
		Overload	302	10.21	8.00	22%
		Underload	282	11.20	8.23	26%
	200	Balanced	601	9.47	8.76	8%
		Overload	598	10.56	9.07	14%
		Underload	564	11.02	9.05	18%
	300	Balanced	896	10.11	9.08	10%
		Overload	917	10.26	9.04	12%
		Underload	847	10.91	9.26	15%

Table 4.2: Mean Penalty Per Course For Random Input Instances

4.1.1 Analysis of Results with Tolerance Value 0

For $\Delta = 0$, CSA was not provided test instances with overloaded or underloaded initial schedules. In the scenario where the input had a balance between the total number of course credits and the total number of available professor credits, the results of meeting preference levels are shown in Tables 4.3 and 4.4, for courses and professors, respectively. Each row represents the values for the number of professor and mean number of courses in the input.

The results show that the preference satisfaction distribution remain similar even when the number of professors changes. A possible cause for this lies in the fact that the total number of time slots in the input was kept constant. This, along with

# of Courses	# of Professors	% of Courses Meeting Preference Levels		
		Highest	Normal	Least
296	100	30	40	30
598	200	30	38	31
896	300	31	39	30

Table 4.3: Comparison of Course Preference Satisfaction for $\Delta = 0$

# of Courses	# of Professors	% of Professors Meeting Preference Levels		
		Highest	Normal	Least
296	100	31	39	30
598	200	35	34	31
896	300	34	36	30

Table 4.4: Comparison of Professor Preference Satisfaction for $\Delta = 0$

the constant distribution of credits among courses, adversely affected the preference satisfactions. Another observation that was made in the results is that for instances where the initial schedule was partial or empty, a valid solution could not be found. The mean CPU time in seconds for each run is noted in Table 4.5.

# of Courses	# of Professors	Mean CPU Time (in Seconds)
296	100	69
598	200	165
896	300	277

Table 4.5: Mean CPU Time in Seconds Used Per Instance for $\Delta = 0$

4.1.2 Analysis of Results with Tolerance Value 1

For $\Delta = 1$, CSA was able to find valid solutions for balanced and overloaded initial schedule. The input covered all three types of initial schedules, that is, complete, partial, and empty. However, in the data set that contained less total course credits than total available professor credits, CSA was unable to find any valid solutions. This follows the mathematical reasoning that the total number of courses being

scheduled was less than the number of professors by a factor greater than Δ itself. The satisfaction result for these inputs is shown in Tables 4.6 and 4.7, for courses and professors, respectively. The parent row represents the values for the number of professors and the mean number of courses in the input. The child row represents the type of input that was provided, namely, balanced or overloaded schedule. In each of the following cells, a 3-tuple is used to show the satisfaction when the initial schedule, ω_i is a complete schedule, partial schedule, and empty schedule respectively.

Mean # of Courses	# of Professors	Professor Credit Relationship	% of Courses Meeting Preference Levels		
			Highest	Normal	Least
294	100	Balanced	(33, 32, 32)	(47, 49, 50)	(20, 19, 18)
		Overload	(34, 33, 33)	(50, 51, 51)	(15, 15, 15)
588	200	Balanced	(30, 30, 30)	(45, 49, 50)	(25, 20, 19)
		Overload	(32, 30, 30)	(48, 50, 50)	(20, 20, 20)
887	300	Balanced	(30, 31, 29)	(48, 49, 52)	(21, 20, 20)
		Overload	(31, 30, 30)	(48, 49, 50)	(21, 21, 20)

† Each triple represents the results when the initial schedule is complete, partial and empty, respectively.

Table 4.6: Comparison of Course Meeting Preference Level for $\Delta = 1$ Over a Complete, Partial, and Empty Initial Schedule

As is seen in the tables, the results show a slight increase in the preference satisfaction for both professor and courses when the initial schedule is overloaded, and a complete or partial initial schedule is provided in the input instance. However, since the instance with an empty initial schedule does not have a valid starting point for CSA, this increase in preference satisfaction does not hold for the empty initial schedules. The mean CPU time in seconds for each run is noted in Table 4.8.

Mean # of Courses	# of Professors	Professor Credit Relationship	% of Professors Meeting Preference Levels		
			Highest	Normal	Least
294	100	Balanced	(36, 37, 39)	(41, 43, 42)	(22, 19, 18)
		Overload	(44, 44, 44)	(39, 39, 39)	(17, 16, 16)
588	200	Balanced	(38, 38, 38)	(38, 41, 42)	(24, 21, 19)
		Overload	(36, 36, 38)	(42, 43, 42)	(22, 21, 19)
887	300	Balanced	(36, 37, 39)	(42, 42, 41)	(22, 22, 20)
		Overload	(38, 37, 37)	(41, 41, 42)	(21, 21, 21)

† Each triple represents the results when the initial schedule is complete, partial and empty, respectively.

Table 4.7: Comparison of Professor Meeting Preference Level for $\Delta = 1$ Over a Complete, Partial, and Empty Initial Schedule

# of Professors	Professor Credit Relationship	Mean # of Courses	Mean CPU Time (in Seconds)		
			Complete ω_i	Partial ω_i	Empty ω_i
100	Balanced	296	40	35	34
	Overload	302	95	88	82
200	Balanced	598	95	96	102
	Overload	603	98	94	98
300	Balanced	896	195	196	201
	Overload	917	204	206	210

Table 4.8: Mean CPU Time in Seconds Used Per Instance for $\Delta = 1$

4.1.3 Analysis of Results with Tolerance Value 2

Finally, for $\Delta = 2$, CSA was able to find valid solutions for all input with all variations of initial schedules (complete, partial and empty initial schedules). Since the value of Δ was sufficiently large to accommodate shifting of courses among professors, it was able to have significantly lower run time. The results, however, are similar to the $\Delta = 0$ and $\Delta = 1$, as shown in Tables 4.9 and 4.10, for courses and professors, respectively. The tables show that the preference satisfaction again remains consistent, with an increase in the number of professors and changes in the relationship between total professor and course credits. The parent row represents the values for the number of professors and mean number of courses in the input. The child row represents the type of input that was provided, namely, balanced, overload, and underload schedule. In each of the following cells, a 3-tuple is used to show the satisfaction when the initial schedule, ω_i is a complete schedule, partial schedule, and empty schedule respectively. From the mean CPU time shown in Table 4.11, it can be seen

Mean # of Courses	# of Professors	Professor Credit Relationship	% of Courses Meeting Preference Levels		
			Highest	Normal	Least
294	100	Balanced	(37, 38, 38)	(49, 49, 49)	(13, 13, 13)
		Overload	(37, 36, 36)	(50, 51, 51)	(13, 13, 13)
		Underload	(38, 38, 38)	(49, 49, 49)	(13, 13, 13)
588	200	Balanced	(35, 35, 35)	(49, 49, 50)	(16, 16, 16)
		Overload	(34, 34, 34)	(50, 50, 50)	(16, 16, 16)
		Underload	(33, 33, 33)	(51, 51, 51)	(15, 15, 15)
887	300	Balanced	(34, 33, 33)	(49, 50, 49)	(17, 17, 17)
		Overload	(33, 33, 33)	(49, 50, 50)	(17, 17, 17)
		Underload	(33, 33, 33)	(50, 50, 50)	(17, 17, 17)

† Each triple represents the results when the initial schedule is complete, partial and empty, respectively.

Table 4.9: Comparison of Course Meeting Preference Level for $\Delta = 2$ Over a Complete, Partial, and Empty Initial Schedule

that increasing the value of δ decreases the run time of the algorithm. One reason

Mean # of Courses	# of Professors	Professor Credit Relationship	% of Professors Meeting Preference Levels		
			Highest	Normal	Least
294	100	Balanced	(44, 44, 46)	(40, 40, 40)	(15, 15, 14)
		Overload	(47, 46, 48)	(38, 38, 38)	(15, 15, 14)
		Underload	(46, 45, 47)	(41, 41, 39)	(14, 13, 14)
588	200	Balanced	(42, 42, 42)	(40, 41, 42)	(17, 17, 16)
		Overload	(41, 41, 42)	(42, 42, 42)	(17, 17, 16)
		Underload	(42, 42, 43)	(41, 42, 41)	(16, 16, 16)
887	300	Balanced	(41, 40, 41)	(41, 42, 42)	(18, 18, 17)
		Overload	(41, 41, 41)	(41, 41, 41)	(18, 18, 17)
		Underload	(41, 41, 41)	(41, 41, 41)	(18, 18, 18)

† Each triple represents the results when the initial schedule is complete, partial and empty, respectively.

Table 4.10: Comparison of Professor Meeting Preference Level for $\Delta = 2$ Over a Complete, Partial, and Empty Initial Schedule

for this is that with a larger δ value, CSA has more flexibility to balance professors, allowing CSA to attain a valid population very quickly.

# of Professors	Professor Credit Relationship	Mean # of Courses	Mean CPU Time (in Seconds)		
			Complete ω_i	Partial ω_i	Empty ω_i
100	Balanced	296	19	17	16
	Overload	302	19	17	15
	Underload	282	23	20	23
200	Balanced	598	100	87	84
	Overload	603	87	88	78
	Underload	564	100	86	79
300	Balanced	896	200	188	200
	Overload	917	246	244	258
	Underload	847	247	210	202

Table 4.11: Mean CPU Time in Seconds Used Per Instance

4.2 Real World Schedule

A real world data set was also provided to CSA for testing. The statistics for this data set are in Table A.19. The data set contained a complete initial schedule and was tested for Δ values 1 and 2. Table 4.13 shows a comparison of the initial schedule and the results for $\Delta = \{1, 2\}$ for courses meeting preference levels and Table 4.14 shows a comparison for professors meeting preference levels. The schedule was not tested for $\Delta = 0$ as the total number of credits for courses exceeded the total number of credits for professors, making it mathematically impossible to provide a schedule where the professor credit constraints were met precisely. Overall, from Table 4.12, it can be seen that a significant improvement is seen for both $\Delta = 1$ and $\Delta = 2$ for penalties per course.

Δ Value	# of Professors	Mean # Of Courses	Mean Penalty Per Course		
			Input (ω_i)	Output ($\omega^\star$)	Percentage Improvement
$\Delta = 1$	37	85	3.47	1.12	68%
$\Delta = 2$	37	85	3.12	1.00	68%

Table 4.12: Mean Penalty Per Course For Real Schedule

Preference Levels	Initial Schedule	Results	
		$\Delta = 1$	$\Delta = 2$
Highest	61%	68%	69%
Normal	25%	31%	29%
Least	14%	1%	2%

Table 4.13: % of Courses Meeting Preference Levels

Preference Levels	Initial Schedule	Results	
		$\Delta = 1$	$\Delta = 2$
Highest	55%	71%	71%
Normal	38%	23%	25%
Least	8%	6%	4%

Table 4.14: % of Professors Meeting Preference Levels

As can be seen in the Tables 4.13 and 4.14, the initial schedule was already very good. Since the provided input was overloaded, there was a slight increase in the courses that were scheduled in their highest and normal preferred time periods. However, a significant reduction is seen in the courses that were scheduled during their least preferred time periods. Increasing the Δ value showed only a small improvement of the overall schedule from the perspective of courses. From the professor's perspective, the schedule was improved significantly by CSA. However, the increase of Δ value saw only slight changes. A detailed comparison of schedules is provided in the Appendix A.3. The mean CPU time used is presented in Table 4.15.

Δ	CPU Time in Seconds
1	5.22169
2	0.86632

Table 4.15: Mean CPU Time in Seconds Used Per Instance

Chapter 5

Conclusion

In this thesis, a new variation of the university timetabling problem, called course scheduling with preference optimization, was presented. The focus of CSPOp was to maximize the preference satisfaction for both courses and professors. To solve this NP-hard problem, a hybrid genetic algorithm, Course Scheduling Algorithm, was presented. The CSA was able to take into account the various preferences and constraints provided to it and return a valid schedule where the preferences for both courses and professors had been maximized. It was also able to ensure that the difference between a professor's credit limit and the total number of credits assigned to the professor is kept to the minimum.

CSA was tested with 243 different input instances. From the results, it was noted that given a sufficient tolerance value for Δ, CSA is able to create a good schedule from either a complete or partial initial schedule. It is also able to create a schedule if no initial schedule is given. In all the resultant schedules, the preference constraints are optimized to ensure that all schedules are valid as per hard constraint violations. CSA is also able to ensure that the resulting scheduling minimizes penalties for soft constraint violations. It was also noted that the increase in the number of courses and professors has an adverse effect on the run time of the algorithm.

As mentioned earlier, the tolerance for the difference between the credit limit for each professor and the number of credits each is assigned can play a major role in determining the efficiency and the final run time for the algorithm. Overall, CSA was able to provide valid results, except when the Δ was 0 and the data set consisted of sets of professors and courses such that the total number of credits of courses was

different than the total number of credits for professors. As can be understood, if the number of credits between professors and courses is different, it will not be possible to assign each professor an exact number of credits.

In the future, a GUI to help with the generation of the input and viewing of the results would be very helpful. A GUI would enable CSA to be deployed and be used by regular end users. With the help of the GUI, the users could enter the various details relating to professors, courses and time slots in a user-friendly, graphical application. Also, the GUI would be able to provide a timetable view of a schedule returned by CSA. This would allow various educational institutions to schedule courses easily, while utilizing CSA to ensure the schedules satisfy the most preferences for both courses and professors.

CSA currently uses a small set of soft constraints that were presented in Table 3.1. These soft constraints may be changed to change the focus of the algorithm very easily. In the future, a possible modification to the local optimization may be able to use the soft constraints to increase the efficiency.

Appendix A

Data Analysis

A.1 Statistics

The input instances that CSA was tested on contained data sets that are categorized in this appendix by the number of professors they had. The various numbers of professors were 100, 200, and 300. For each data set that follows, a table is presented that contains the total number of courses in that data set, the maximum fitness f_{max}, the mean number of courses each course is incompatible with, the mean number of courses that can be taught by a professor, and the mean number of professors that can teach a course.

For each different set of professors, there were three unique data sets that were generated for each relationship between total number of credits for courses and total number of available credits for professors. These unique data sets are labeled A, B, and C.

A.1.1 Professor Count of 100

	A	B	C
Total Courses	297	297	295
Difference Between Total Credits	0	0	0
Maximum Fitness (f_{max})	445,800	453,360	446,360
Range of Incompatible Courses per Course	[0, 10]	[0, 10]	[0, 10]
Range of # of Professors per Course	[6, 23]	[7, 22]	[6, 24]
Range of # of Courses per Professor	[29, 56]	[29, 56]	[29, 56]

Table A.1: Statistics for Data Sets of 100 Professors With Balanced Course Load

	A	B	C
Total Courses:	300	306	301
Difference Between Total Credits	3	9	8
Maximum Fitness (f_{max})	457,890	485,400	470,830
Range of Incompatible Courses per Course	[0, 10]	[0, 10]	[0, 10]
Range of # of Professors per Course	[6, 24]	[6, 24]	[5, 27]
Range of # of Courses per Professor	[30, 57]	[30, 58]	[30, 57]

Table A.2: Statistics for Data Sets of 100 Professors With Course Overload

	A	B	C
Total Courses:	281	281	284
Difference Between Total Credits	52	50	52
Maximum Fitness (f_{max})	403,560	416,650	421,710
Range of Incompatible Courses per Course	[0, 10]	[0, 10]	[0, 10]
Range of # of Professors per Course	[7, 25]	[5, 24]	[5, 26]
Range of # of Courses per Professor	[28, 53]	[28, 53]	[28, 53]

Table A.3: Statistics for Data Sets of 100 Professors With Course Underload

A.1.2 Professor Count of 200

	A	B	C
Total Courses:	594	601	598
Difference Between Total Credits	0	0	0
Maximum Fitness (f_{max})	1,867,500	1,910,800	1,893,530
Range of Incompatible Courses per Course	[0, 10]	[0, 10]	[0, 10]
Range of # of Professors per Course	[15, 50]	[6, 45]	[10, 48]
Range of # of Courses per Professor	[59, 112]	[60, 114]	[59, 113]

Table A.4: Statistics for Data Sets of 200 Professors With Balanced Course Load

	A	B	C
Total Courses:	600	604	606
Difference Between Total Credits	18	23	25
Maximum Fitness (f_{max})	1,908,490	1,929,590	1,940,260
Range of Incompatible Courses per Course	[0, 10]	[0, 10]	[0, 10]
Range of # of Professors per Course	[12, 44]	[14, 48]	[15, 44]
Range of # of Courses per Professor	[60, 114]	[60, 114]	[60, 115]

Table A.5: Statistics for Data Sets of 200 Professors With Course Overload

	A	B	C
Total Courses:	564	565	564
Difference Between Total Credits	100	101	100
Maximum Fitness (f_{max})	1,698,230	1,704,380	1,696,380
Range of Incompatible Courses per Course	[0, 10]	[0, 10]	[0, 10]
Range of # of Professors per Course	[13, 44]	[14, 42]	[16, 46]
Range of # of Courses per Professor	[56, 107]	[56, 107]	[56, 107]

Table A.6: Statistics for Data Sets of 200 Professors With Course Underload

A.1.3 Professor Count of 300

	A	B	C
Total Courses:	897	898	894
Difference Between Total Credits	0	0	0
Maximum Fitness (f_{max})	4,212,590	4,222,940	4,187,660
Range of Incompatible Courses per Course	[0, 10]	[0, 10]	[0, 10]
Range of # of Professors per Course	[26, 63]	[26, 66]	[25, 67]
Range of # of Courses per Professor	[89, 170]	[89, 170]	[89, 169]

Table A.7: Statistics for Data Sets of 300 Professors With Balanced Course Load

	A	B	C
Total Courses:	921	920	911
Difference Between Total Credits	50	65	47
Maximum Fitness (f_{max})	4,431,200	4,421,580	4,337,740
Range of Incompatible Courses per Course	[0, 10]	[0, 10]	[0, 10]
Range of # of Professors per Course	[27, 70]	[23, 61]	[23, 62]
Range of # of Courses per Professor	[92, 174]	[92, 174]	[91, 173]

Table A.8: Statistics for Data Sets of 300 Professors With Course Overload

	A	B	C
Total Courses:	852	845	845
Difference Between Total Credits	151	150	152
Maximum Fitness (f_{max})	3,818,760	3,757,810	3,758,020
Range of Incompatible Courses per Course	[0, 10]	[0, 10]	[0, 10]
Range of # of Professors per Course	[23, 63]	[24, 60]	[27, 64]
Range of # of Courses per Professor	[85, 161]	[84, 160]	[84, 160]

Table A.9: Statistics for Data Sets of 300 Professors With Course Underload

A.2 Initial Schedule Preference Statistics

This section provides details about the preference level satisfaction for the initial schedules provided in Appendix A.1. The satisfactions are only presented for the complete schedules. Each cell represents the number of courses satisfying the preference level indicated by the column title.

A.2.1 Professor Count of 100

	Preference	A	B	C	Mean	Percentile
Number of Courses		297	297	295	296	
Courses	Highest	96	106	91	98	33%
	Normal	116	100	122	113	38%
	Least	85	91	82	86	29%
Professors	Highest	92	86	87	88	30%
	Normal	117	120	122	120	40%
	Least	88	91	86	88	30%

Table A.10: Preference Statistics for 100 Professors with Balanced Course Load

	Preference	A	B	C	Mean	Percentile
Number of Courses		300	306	301	302	
Courses	Highest	95	88	104	96	32%
	Normal	119	111	123	118	39%
	Least	86	107	84	92	31%
Professors	Highest	100	105	89	98	32%
	Normal	107	113	118	113	37%
	Least	93	88	94	92	30%

Table A.11: Preference Statistics for 100 Professors with Course Overload

	Preference	A	B	C	Mean	Percentile
Number of Courses		281	281	284	282	
Courses	Highest	105	101	96	101	36%
	Normal	105	96	102	101	36%
	Least	71	84	86	80	28%
Professors	Highest	72	83	97	84	30%
	Normal	118	116	99	111	39%
	Least	91	82	88	87	31%

Table A.12: Preference Statistics for 100 Professors with Course Underload

A.2.2 Professor Count of 200

	Preference	A	B	C	Mean	Percentile
Number of Courses		600	600	602	601	
Courses	Highest	212	214	210	212	35%
	Normal	211	212	215	213	35%
	Least	177	174	177	176	29%
Professors	Highest	178	178	199	185	31%
	Normal	243	229	242	238	40%
	Least	179	193	161	178	30%

Table A.13: Preference Statistics for 200 Professors with Balanced Course Load

	Preference	A	B	C	Mean	Percentile
Number of Courses		594	601	598	598	
Courses	Highest	228	195	188	204	34%
	Normal	194	229	226	216	36%
	Least	178	180	192	183	31%
Professors	Highest	196	187	217	200	33%
	Normal	220	245	228	231	39%
	Least	184	172	161	172	29%

Table A.14: Preference Statistics for 200 Professors with Course Overload

	Preference	A	B	C	Mean	Percentile
Number of Courses		564	565	564	564	
Courses	Highest	198	210	202	203	36%
	Normal	218	205	202	208	37%
	Least	148	150	160	153	27%
Professors	Highest	177	156	171	168	30%
	Normal	234	247	234	238	42%
	Least	153	162	159	158	28%

Table A.15: Preference Statistics for 200 Professors with Course Underload

A.2.3 Professor Count of 300

	Preference	A	B	C	Mean	Percentile
Number of Courses		897	898	894	896	
Courses	Highest	294	310	293	299	33%
	Normal	351	324	334	336	38%
	Least	252	264	267	261	29%
Professors	Highest	269	285	272	275	31%
	Normal	348	357	347	351	39%
	Least	280	256	275	270	30%

Table A.16: Preference Statistics for 300 Professors with Balanced Course Load

	Preference	A	B	C	Mean	Percentile
Number of Courses		921	920	911	917	
Courses	Highest	296	313	335	315	34%
	Normal	338	341	309	329	36%
	Least	287	266	267	273	30%
Professors	Highest	322	289	294	302	33%
	Normal	357	368	345	357	39%
	Least	242	263	272	259	28%

Table A.17: Preference Statistics for 300 Professors with Course Overload

	Preference	A	B	C	Mean	Percentile
Number of Courses		852	845	845	847	
Courses	Highest	292	297	302	297	35%
	Normal	317	308	303	309	37%
	Least	243	240	240	241	28%
Professors	Highest	264	255	258	259	31%
	Normal	342	348	325	338	40%
	Least	246	242	262	250	30%

Table A.18: Preference Statistics for 300 Professors with Course Underload

A.3 Real World Schedule

The statistics for the real world schedule data set are in Table A.19.

Total Courses	85
Total Professors	37
Total Time slots	34
Total Credits for Courses	279.5
Total Credits for Professors	273
Maximum Fitness Value	$23,185$
Mean Penalty per Course	258
Range of Incompatible Courses per Course	$[0, 4]$
Range of # of Professors per Course	$[1, 10]$
Range of # of Courses per Professor	$[1, 7]$

Table A.19: Input Data Statistics

The comparisons of schedules for $\Delta = \{1, 2\}$ are provided in Tables A.20 and A.21. Professors without defined preferences are marked with a '–'. The names of the professors have been changed to protect their identities.

Table A.20 Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 1$

Input Schedule			Preferences		Result			Preferences	
Course	Professor	Timeslot	Prof	Course	Course	Professor	Timeslot	Prof	Course
math026(1)	Ada	MWF(11:15AM-12:05PM)	H	H	math026(1)	Ada	MWF(11:15AM-12:05PM)	H	H
math026(2)	Ada	TR(9:45AM-11:00AM)	H	H	math026(2)	Ada	TR(9:45AM-11:00AM)	H	H
math220(1)	Ada	MW(9:05AM-9:55AM)	H	H	math220(3)	Ada	MW(9:05AM-9:55AM)	H	H
math220(3)	Ada	MW(3:35PM-4:25PM)	N	N	stat200(3)	Ada	MTWF(8:00AM-8:50AM)	H	H
stat200(3)	Ada	M(1:25PM-2:15PM) TR(1:00PM-2:15PM)	N	N	cmpsc121(2)	Alonzo	MWF(11:15AM-12:05PM)	–	H
cmpsc121(1)	Alonzo	MWF(10:10AM-11:00AM)	–	H	cmpsc121(3)	Alonzo	MWF(1:25PM-2:15PM)	–	N
cmpsc221(1)	Alonzo	TR(4:15PM-5:30PM)	–	N	cmpsc221(1)	Alonzo	MWF(12:20PM-1:10PM)	–	N
cmpsc412(1)	Babbage	M(2:30PM-4:20PM)	H	H	cmpsc412(1)	Babbage	M(2:30PM-4:20PM)	H	H
cmpsc421(1)	Babbage	R(6:00PM-9:00PM)	L	H	cmpsc421(1)	Babbage	TR(2:30PM-3:45PM)	H	N
cmpsc444(1)	Babbage	M(6:00PM-9:00PM)	L	H	cmpsc444(1)	Babbage	M(6:00PM-9:00PM)	L	H
cmpsc462(1)	Babbage	TR(1:00PM-2:15PM)	H	H	cmpsc462(1)	Babbage	MWF(1:25PM-2:15PM)	H	H
math140(8)	Borg	MTWF(8:00AM-8:50AM)	H	H	math022(2)	Borg	MWF(9:05AM-9:55AM)	H	H
math430(1)	Borg	MWF(9:05AM-9:55AM)	H	H	math430(1)	Borg	TR(9:45AM-11:00AM)	H	H
math475w(1)	Borg	TR(2:30PM-3:45PM)	N	H	math475w(1)	Borg	TR(2:30PM-3:45PM)	N	H
math022(5)	Cantor	MWF(9:05AM-9:55AM)	H	H	math022(5)	Cantor	MWF(9:05AM-9:55AM)	H	H
math250(1)	Cantor	MTWF(1:25PM-2:15PM)	N	H	math141(3)	Cantor	MTWF(10:10AM-11:00AM)	H	H
math250(2)	Cantor	MTWF(2:30PM-3:20PM)	N	H	math250(1)	Cantor	M(1:25PM-2:15PM) TR(1:00PM-2:15PM)	N	H
math140(3)	Courant	MTWF(8:00AM-8:50AM)	H	H	math140(1)	Courant	MTWF(10:10AM-11:00AM)	H	H
math141(3)	Courant	MTWF(10:10AM-11:00AM)	H	H	math140(3)	Courant	MTWF(8:00AM-8:50AM)	H	H
math004(2)	Cray	MW(6:00PM-7:15PM)	–	N	math004(2)	Cray	MWF(11:15AM-12:05PM)	–	H
math140(1)	Dahl	MTWF(10:10AM-11:00AM)	N	H	math250(2)	Dahl	M(4:40PM-5:30PM) TR(4:15PM-5:50PM)	H	H
math140(4)	Dahl	MTWF(1:25PM-2:15PM)	H	N	math250(3)	Dahl	MTWF(2:30PM-3:20PM)	H	H

Continued on next page

Table A.20 Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 1$ - continued from previous page

Input Schedule			Preferences		Result			Preferences	
Course	Professor	Timeslot	Prof	Course	Course	Professor	Timeslot	Prof	Course
math250(3)	Dahl	MTWF(2:30PM-3:20PM)	H	H					
math022(3)	Dijkstra	MWF(10:10AM-11:00AM)	N	H	math022(3)	Dijkstra	TR(9:45AM-11:00AM)	N	H
math041(1)	Dijkstra	MWF(12:20PM-1:10PM)	H	N	math041(1)	Dijkstra	MWF(12:20PM-1:10PM)	H	N
math141(2)	Eckert	MWRF(10:10AM-11:00AM)	H	H	math141(2)	Eckert	MWRF(10:10AM-11:00AM)	H	H
math141(4)	Eckert	MWRF(1:25PM-2:15PM)	N	N	math141(4)	Eckert	MTWF(8:00AM-8:50AM)	H	H
math220(2)	Eckert	WF(9:05AM-9:55AM)	H	H	math220(2)	Eckert	WF(9:05AM-9:55AM)	H	H
					math220(4)	Eckert	MW(3:35PM-4:25PM)	N	N
comp505(1)	Erdos	T(6:00PM-9:00PM)	–	H	comp505(1)	Erdos	T(6:00PM-9:00PM)	–	H
cmpsc200(1)	Euclid	TR(4:15PM-5:30PM)	–	N	cmpsc200(1)	Euclid	TR(4:15PM-5:30PM)	–	N
math110(2)	Euler	M(4:40PM-5:30PM) TR(4:15PM-5:30PM)	H	N	math110(2)	Euler	F(1:25PM-2:15PM) TR(1:00PM-2:15PM)	H	N
math140(9)	Euler	MWRF(10:10AM-11:00AM)	N	H	math140(4)	Euler	M(4:40PM-5:30PM) TR(4:15PM-5:50PM)	H	N
math230(2)	Euler	MTWF(2:30PM-3:20PM)	H	L	math140(8)	Euler	MTWF(8:00AM-8:50AM)	N	H
math140(2)	Fermat	TR(6:00PM-8:00PM)	–	L	math140(2)	Fermat	MWRF(1:25PM-2:15PM)	–	N
stat200(5)	Fourier	TR(9:45AM-11:00AM) M(10:10AM-11:00AM)	H	H	stat200(5)	Fourier	MTWF(8:00AM-8:50AM)	H	H
stat401(1)	Fourier	MWF(2:30PM-3:20PM)	N	H	stat401(1)	Fourier	MWF(2:30PM-3:20PM)	N	H
stat460(1)	Fourier	MWF(1:25PM-2:15PM)	N	H	stat460(1)	Fourier	MWF(11:15AM-12:05PM)	H	N
math140(6)	Frege	MTWF(2:30PM-3:20PM)	H	N	math140(6)	Frege	MTWF(10:10AM-11:00AM)	N	H
stat200(6)	Frege	TR(9:45AM-11:00AM) F(10:10AM-11:00AM)	N	H	stat200(6)	Frege	M(4:40PM-5:30PM) TR(4:15PM-5:50PM)	H	N
stat200(7)	Frege	MTWF(8:00AM-8:50AM)	N	H	stat200(7)	Frege	M(1:25PM-2:15PM) TR(1:00PM-2:15PM)	H	N

Continued on next page

Table A.20 Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 1$ - continued from previous page

Input Schedule			Preferences		Result			Preferences	
Course	Professor	Timeslot	Prof	Course	Course	Professor	Timeslot	Prof	Course
math022(2)	Galois	MWF(9:05AM-9:55AM)	–	H	math140(7)	Galois	TR(9:45AM-11:00AM) M(10:10AM-11:00AM)	–	H
math140(7)	Galois	MTWF(10:10AM-11:00AM)	–	H	math140(9)	Galois	M(4:40PM-5:30PM) TR(4:15PM-5:50PM)	–	N
cmpsc101(1)	Gauss	MWF(2:30PM-3:20PM)	H	N	cmpsc101(1)	Gauss	MWF(2:30PM-3:20PM)	H	N
math110(1)	Gauss	TR(9:45AM-11:00AM) M(10:10AM-11:00AM)	N	H	math110(3)	Gauss	F(1:25PM-2:15PM) TR(1:00PM-2:15PM)	H	N
math110(3)	Gauss	F(1:25PM-2:15PM) TR(1:00PM-2:15PM)	H	N	math110(4)	Gauss	M(4:40PM-5:30PM) TR(4:15PM-5:50PM)	H	N
math110(4)	Godel	M(4:40PM-5:30PM) TR(4:15PM-5:50PM)	–	N	math110(1)	Godel	MTWF(8:00AM-8:50AM)	–	H
math141(1)	Godel	MTWF(10:10AM-11:00AM)	–	H	math230(1)	Godel	MTWF(1:25PM-2:15PM)	–	L
math230(3)	Godel	MWRF(2:30PM-3:20PM)	–	L	math230(3)	Godel	TR(6:00PM-8:00PM)	–	H
cmpsc472(1)	Grace	TR(2:30PM-3:45PM)	H	H	cmpsc472(1)	Grace	MWF(1:25PM-2:15PM)	H	H
comp519(1)	Grace	M(6:00PM-9:00PM)	L	H	comp519(1)	Grace	M(6:00PM-9:00PM)	L	H
comp594(1)	Grace	W(6:00PM-9:00PM)	L	H	comp594(1)	Grace	W(6:00PM-9:00PM)	L	H
math021(5)	Hamming	MWF(11:15AM-12:05PM)	N	L	math140(5)	Hamming	MTWF(2:30PM-3:20PM)	H	N
math140(5)	Hamming	MTWF(2:30PM-3:20PM)	H	N	math141(1)	Hamming	MTWF(10:10AM-11:00AM)	N	H
math435(1)	Hamming	MWF(1:25PM-2:15PM)	H	H	math435(1)	Hamming	MWF(1:25PM-2:15PM)	H	H
cmpsc121(2)	Hardy	R(6:00PM-9:00PM)	–	L	cmpsc121(1)	Hardy	MWF(9:05AM-9:55AM)	–	H
math026(3)	Hilbert	MWF(11:15AM-12:05PM)	–	H	math026(3)	Hilbert	MWF(9:05AM-9:55AM)	–	H
math003(1)	Noether	TR(4:15PM-5:30PM)	–	H	math003(1)	Noether	TR(4:15PM-5:30PM)	–	H
math021(3)	Noyce	W(6:00PM-9:00PM)	–	H	math021(6)	Noyce	W(6:00PM-9:00PM)	–	H
cmpsc487w(1)	Nygaard	W(6:00PM-9:00PM)	–	H	cmpsc487w(1)	Nygaard	W(6:00PM-9:00PM)	–	H
cmpsc122(1)	Pascal	TR(1:00PM-2:15PM)	N	N	cmpsc122(1)	Pascal	TR(1:00PM-2:15PM)	N	N
cmpsc470(1)	Pascal	TR(4:15PM-5:30PM)	N	H	cmpsc470(1)	Pascal	MWF(12:20PM-1:10PM)	N	H

Continued on next page

Table A.20 Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 1$ - continued from previous page

Input Schedule			Preferences		Result			Preferences	
Course	Professor	Timeslot	Prof	Course	Course	Professor	Timeslot	Prof	Course
comp516(1)	Pascal	R(6:00PM-9:00PM)	H	H	comp516(1)	Pascal	R(6:00PM-9:00PM)	H	H
math004(1)	Peano	TR(9:45AM-11:00AM)	–	H	math004(1)	Peano	MWF(10:10AM-11:00AM)	–	H
math022(4)	Peano	MWF(2:30PM-3:20PM)	–	N	math022(4)	Peano	MWF(2:30PM-3:20PM)	–	N
cmpsc121(3)	Perlis	MWF(1:25PM-2:15PM)	H	N	math021(1)	Perlis	MWF(12:20PM-1:10PM)	H	N
math021(2)	Perlis	MWF(10:10AM-11:00AM)	N	L	math021(3)	Perlis	TR(2:30PM-3:45PM)	H	N
math021(6)	Perlis	MWF(9:05AM-9:55AM)	N	L	math021(4)	Perlis	MWF(1:25PM-2:15PM)	H	N
math220(4)	Riemann	MW(3:35PM-4:25PM)	–	N	math220(1)	Riemann	MW(9:05AM-9:55AM)	–	H
math230(1)	Riemann	MTWF(1:25PM-2:15PM)	–	L	math230(2)	Riemann	TR(6:00PM-8:00PM)	–	H
math449(1)	Riemann	MWF(2:30PM-3:20PM)	–	H	math449(1)	Riemann	MWF(2:30PM-3:20PM)	–	H
stat200(1)	Shannon	TR(9:45AM-11:00AM) M(10:10AM-11:00AM)	–	H	stat200(1)	Shannon	MTWF(10:10AM-11:00AM)	–	H
stat200(2)	Shannon	MW(6:00PM-8:00PM)	–	L	stat200(2)	Shannon	M(4:40PM-5:30PM) TR(4:15PM-5:30PM)	–	N
stat200(4)	Shannon	F(1:25PM-2:15PM) TR(1:00PM-2:15PM)	–	N	stat200(4)	Shannon	F(1:25PM-2:15PM) TR(1:00PM-2:15PM)	–	N
math022(1)	Ulam	MW(6:00PM-7:15PM)	–	L	math022(1)	Ulam	TR(4:15PM-5:30PM)	–	N
math315(1)	Wilkes	MWF(10:10AM-11:00AM)	H	H	math315(1)	Wilkes	MWF(10:10AM-11:00AM)	H	H
math318(1)	Wilkes	MWF(11:15AM-12:05PM)	H	H	math318(1)	Wilkes	MWF(9:05AM-9:55AM)	H	H
cmpsc360(1)	Wilkinson	TR(2:30PM-3:45PM)	N	H	cmpsc360(1)	Wilkinson	TR(2:30PM-3:45PM)	N	H
cmpsc360(2)	Wilkinson	TR(9:45AM-11:00AM)	H	L	cmpsc360(2)	Wilkinson	MWF(1:25PM-2:15PM)	N	H
math200(1)	Zeno	TR(9:45AM-11:00AM)	–	H	math200(1)	Zeno	MWF(10:10AM-11:00AM)	–	H
math041(2)	Zermelo	MWF(12:20PM-1:10PM)	–	N	math041(2)	Zermelo	MWF(12:20PM-1:10PM)	–	N
math017(1)	Zorn	MWF(12:20PM-1:10PM)	–	H	math017(1)	Zorn	MWF(12:20PM-1:10PM)	–	H
math017(2)	Zorn	T(6:00PM-9:00PM)	–	N	math017(2)	Zorn	MWF(1:25PM-2:15PM)	–	H
math021(1)	Zorn	MWF(9:05AM-9:55AM)	–	L	math021(2)	Zorn	R(6:00PM-9:00PM)	–	H

Continued on next page

Table A.20 Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 1$ - continued from previous page

Input Schedule			Preferences		Result			Preferences	
Course	Professor	Timeslot	Prof	Course	Course	Professor	Timeslot	Prof	Course
math021(4)	Zorn	MW(6:00PM-7:15PM)	–	H	math021(5)	Zorn	MW(6:00PM-7:15PM)	–	H
math035(1)	Zorn	MWF(10:10AM-11:00AM)	–	H	math035(1)	Zorn	MWF(10:10AM-11:00AM)	–	H

Table A.20: Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 1$

Table A.21 Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 2$

Input Schedule			Preferences		Result			Preferences	
Course	Professor	Timeslot	Prof	Course	Course	Professor	Timeslot	Prof	Course
math026(1)	Ada	MWF(11:15AM-12:05PM)	H	H	math026(1)	Ada	MWF(11:15AM-12:05PM)	H	H
math026(2)	Ada	TR(9:45AM-11:00AM)	H	H	math026(2)	Ada	TR(1:00PM-2:15PM)	N	N
math220(1)	Ada	MW(9:05AM-9:55AM)	H	H	math220(3)	Ada	MW(9:05AM-9:55AM)	H	H
math220(3)	Ada	MW(3:35PM-4:25PM)	N	N	stat200(5)	Ada	TR(9:45AM-11:00AM) M(10:10AM-11:00AM)	H	H
stat200(3)	Ada	M(1:25PM-2:15PM) TR(1:00PM-2:15PM)	N	N					
cmpsc121(1)	Alonzo	MWF(10:10AM-11:00AM)	–	H	cmpsc121(1)	Alonzo	TR(4:15PM-5:30PM)	–	N
cmpsc221(1)	Alonzo	TR(4:15PM-5:30PM)	–	N	cmpsc121(3)	Alonzo	TR(1:00PM-2:15PM)	–	N
					cmpsc221(1)	Alonzo	MWF(9:05AM-9:55AM)	–	H
cmpsc412(1)	Babbage	M(2:30PM-4:20PM)	H	H	cmpsc412(1)	Babbage	M(2:30PM-4:20PM)	H	H
cmpsc421(1)	Babbage	R(6:00PM-9:00PM)	L	H	cmpsc421(1)	Babbage	TR(2:30PM-3:45PM)	H	N
cmpsc444(1)	Babbage	M(6:00PM-9:00PM)	L	H	cmpsc444(1)	Babbage	MWF(1:25PM-2:15PM)	H	N
cmpsc462(1)	Babbage	TR(1:00PM-2:15PM)	H	H	cmpsc462(1)	Babbage	TR(1:00PM-2:15PM)	H	H
math140(8)	Borg	MTWF(8:00AM-8:50AM)	H	H	math022(2)	Borg	MWF(9:05AM-9:55AM)	H	H
math430(1)	Borg	MWF(9:05AM-9:55AM)	H	H	math140(1)	Borg	MTWF(10:10AM-11:00AM)	H	H
math475w(1)	Borg	TR(2:30PM-3:45PM)	N	H	math475w(1)	Borg	TR(4:15PM-5:30PM)	N	H
math022(5)	Cantor	MWF(9:05AM-9:55AM)	H	H	math140(7)	Cantor	MTWF(10:10AM-11:00AM)	H	H
math250(1)	Cantor	MTWF(1:25PM-2:15PM)	N	H	math141(1)	Cantor	MTWF(8:00AM-8:50AM)	H	H
math250(2)	Cantor	MTWF(2:30PM-3:20PM)	N	H	math250(1)	Cantor	MTWF(1:25PM-2:15PM)	N	H
math140(3)	Courant	MTWF(8:00AM-8:50AM)	H	H	math140(5)	Courant	MTWF(10:10AM-11:00AM)	H	H
math141(3)	Courant	MTWF(10:10AM-11:00AM)	H	H	math141(3)	Courant	M(4:40PM-5:30PM) TR(4:15PM-5:50PM)	N	N
math004(2)	Cray	MW(6:00PM-7:15PM)	–	N	math004(2)	Cray	T(6:00PM-9:00PM)	–	N

Continued on next page

Table A.21 Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 2$ - continued from previous page

Input Schedule			Preferences		Result			Preferences	
Course	Professor	Timeslot	Prof	Course	Course	Professor	Timeslot	Prof	Course
math140(1)	Dahl	MTWF(10:10AM-11:00AM)	N	H	math250(2)	Dahl	M(4:40PM-5:30PM) TR(4:15PM-5:50PM)	H	H
math140(4)	Dahl	MTWF(1:25PM-2:15PM)	H	N	math250(3)	Dahl	MWRF(2:30PM-3:20PM)	H	H
math250(3)	Dahl	MTWF(2:30PM-3:20PM)	H	H					
math022(3)	Dijkstra	MWF(10:10AM-11:00AM)	N	H	math022(1)	Dijkstra	MWF(12:20PM-1:10PM)	H	N
math041(1)	Dijkstra	MWF(12:20PM-1:10PM)	H	N	math041(1)	Dijkstra	MWF(2:30PM-3:20PM)	H	N
math141(2)	Eckert	MWRF(10:10AM-11:00AM)	H	H	math141(2)	Eckert	TR(9:45AM-11:00AM) M(10:10AM-11:00AM)	H	H
math141(4)	Eckert	MWRF(1:25PM-2:15PM)	N	N	math141(4)	Eckert	MTWF(8:00AM-8:50AM)	H	H
math220(2)	Eckert	WF(9:05AM-9:55AM)	H	H	math220(4)	Eckert	WF(9:05AM-9:55AM)	H	H
comp505(1)	Erdos	T(6:00PM-9:00PM)	–	H	comp505(1)	Erdos	T(6:00PM-9:00PM)	–	H
cmpsc200(1)	Euclid	TR(4:15PM-5:30PM)	–	N	cmpsc200(1)	Euclid	M(6:00PM-9:00PM)	–	H
math110(2)	Euler	M(4:40PM-5:30PM) TR(4:15PM-5:30PM)	H	N	math140(4)	Euler	MWRF(1:25PM-2:15PM)	H	N
math140(9)	Euler	MWRF(10:10AM-11:00AM)	N	H	math140(9)	Euler	MWRF(10:10AM-11:00AM)	N	H
math230(2)	Euler	MTWF(2:30PM-3:20PM)	H	L	math230(3)	Euler	MWRF(2:30PM-3:20PM)	H	L
math140(2)	Fermat	TR(6:00PM-8:00PM)	–	L	math140(2)	Fermat	MTWF(8:00AM-8:50AM)	–	H
stat200(5)	Fourier	TR(9:45AM-11:00AM) M(10:10AM-11:00AM)	H	H	stat200(6)	Fourier	TR(9:45AM-11:00AM) F(10:10AM-11:00AM)	H	H
stat401(1)	Fourier	MWF(2:30PM-3:20PM)	N	H	stat401(1)	Fourier	MWF(11:15AM-12:05PM)	H	N
stat460(1)	Fourier	MWF(1:25PM-2:15PM)	N	H	stat460(1)	Fourier	MWF(1:25PM-2:15PM)	N	H
math140(6)	Frege	MTWF(2:30PM-3:20PM)	H	N	math140(6)	Frege	MTWF(2:30PM-3:20PM)	H	N
stat200(6)	Frege	TR(9:45AM-11:00AM) F(10:10AM-11:00AM)	N	H	stat200(4)	Frege	TR(9:45AM-11:00AM) F(10:10AM-11:00AM)	N	H
stat200(7)	Frege	MTWF(8:00AM-8:50AM)	N	H	stat200(7)	Frege	MTWF(8:00AM-8:50AM)	N	H

Continued on next page

Table A.21 Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 2$ - continued from previous page

Input Schedule			Preferences		Result			Preferences	
Course	Professor	Timeslot	Prof	Course	Course	Professor	Timeslot	Prof	Course
math022(2)	Galois	MWF(9:05AM-9:55AM)	–	H	math022(5)	Galois	MWF(9:05AM-9:55AM)	–	H
math140(7)	Galois	MTWF(10:10AM-11:00AM)	–	H	math140(3)	Galois	MTWF(8:00AM-8:50AM)	–	H
cmpsc101(1)	Gauss	MWF(2:30PM-3:20PM)	H	N	cmpsc101(1)	Gauss	MWF(2:30PM-3:20PM)	H	N
math110(1)	Gauss	TR(9:45AM-11:00AM) M(10:10AM-11:00AM)	N	H	math110(1)	Gauss	TR(9:45AM-11:00AM) M(10:10AM-11:00AM)	N	H
math110(3)	Gauss	F(1:25PM-2:15PM) TR(1:00PM-2:15PM)	H	N	math110(3)	Gauss	MTWF(1:25PM-2:15PM)	H	N
math110(4)	Godel	M(4:40PM-5:30PM) TR(4:15PM-5:50PM)	–	N	math110(2)	Godel	M(4:40PM-5:30PM) TR(4:15PM-5:30PM)	–	N
math141(1)	Godel	MTWF(10:10AM-11:00AM)	–	H	math110(4)	Godel	MTWF(10:10AM-11:00AM)	–	H
math230(3)	Godel	MWRF(2:30PM-3:20PM)	–	L	math230(1)	Godel	TR(6:00PM-8:00PM)	–	H
cmpsc472(1)	Grace	TR(2:30PM-3:45PM)	H	H	cmpsc472(1)	Grace	MWF(12:20PM-1:10PM)	H	H
comp519(1)	Grace	M(6:00PM-9:00PM)	L	H	comp519(1)	Grace	M(6:00PM-9:00PM)	L	H
comp594(1)	Grace	W(6:00PM-9:00PM)	L	H	comp594(1)	Grace	W(6:00PM-9:00PM)	L	H
cmpsc487w(1)	Hamming	W(6:00PM-9:00PM)	–	H	cmpsc487w(1)	Hamming	MW(6:00PM-7:15PM)	–	H
math021(5)	Hardy	MWF(11:15AM-12:05PM)	N	L	math140(8)	Hardy	M(4:40PM-5:30PM) TR(4:15PM-5:30PM)	H	N
math140(5)	Hardy	MTWF(2:30PM-3:20PM)	H	N	math430(1)	Hardy	MWF(9:05AM-9:55AM)	N	H
math435(1)	Hardy	MWF(1:25PM-2:15PM)	H	H	math435(1)	Hardy	TR(2:30PM-3:45PM)	H	H
cmpsc121(2)	Hilbert	R(6:00PM-9:00PM)	–	L	cmpsc121(2)	Hilbert	TR(9:45AM-11:00AM)	–	H
math026(3)	Noether	MWF(11:15AM-12:05PM)	–	H	math026(3)	Noether	MWF(11:15AM-12:05PM)	–	H
math003(1)	Noyce	TR(4:15PM-5:30PM)	–	H	math003(1)	Noyce	TR(4:15PM-5:30PM)	–	H
math021(3)	Nygaard	W(6:00PM-9:00PM)	–	H	math021(1)	Nygaard	M(6:00PM-9:00PM)	–	H
cmpsc122(1)	Pascal	TR(1:00PM-2:15PM)	N	N	cmpsc122(1)	Pascal	T(6:00PM-9:00PM)	H	L
cmpsc470(1)	Pascal	TR(4:15PM-5:30PM)	N	H	cmpsc470(1)	Pascal	TR(4:15PM-5:30PM)	N	H

Continued on next page

Table A.21 Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 2$ - continued from previous page

Input Schedule			Preferences		Result			Preferences	
Course	Professor	Timeslot	Prof	Course	Course	Professor	Timeslot	Prof	Course
comp516(1)	Pascal	R(6:00PM-9:00PM)	H	H	comp516(1)	Pascal	R(6:00PM-9:00PM)	H	H
math004(1)	Peano	TR(9:45AM-11:00AM)	–	H	math004(1)	Peano	TR(9:45AM-11:00AM)	–	H
math022(4)	Peano	MWF(2:30PM-3:20PM)	–	N	math022(4)	Peano	MWF(11:15AM-12:05PM)	–	H
cmpsc121(3)	Perlis	MWF(1:25PM-2:15PM)	H	N	math021(2)	Perlis	TR(2:30PM-3:45PM)	H	N
math021(2)	Perlis	MWF(10:10AM-11:00AM)	N	L	math021(3)	Perlis	TR(4:15PM-5:30PM)	H	N
math021(6)	Perlis	MWF(9:05AM-9:55AM)	N	L	math021(6)	Perlis	TR(1:00PM-2:15PM)	H	N
math220(4)	Riemann	MW(3:35PM-4:25PM)	–	N	math220(1)	Riemann	WF(9:05AM-9:55AM)	–	H
math230(1)	Riemann	MTWF(1:25PM-2:15PM)	–	L	math220(2)	Riemann	MW(3:35PM-4:25PM)	–	N
math449(1)	Riemann	MWF(2:30PM-3:20PM)	–	H	math230(2)	Riemann	MW(6:00PM-8:00PM)	–	H
					math449(1)	Riemann	MWF(2:30PM-3:20PM)	–	H
stat200(1)	Shannon	TR(9:45AM-11:00AM) M(10:10AM-11:00AM)	–	H	stat200(1)	Shannon	MWRF(10:10AM-11:00AM)	–	H
stat200(2)	Shannon	MW(6:00PM-8:00PM)	–	L	stat200(2)	Shannon	MTWF(2:30PM-3:20PM)	–	N
stat200(4)	Shannon	F(1:25PM-2:15PM) TR(1:00PM-2:15PM)	–	N	stat200(3)	Shannon	M(4:40PM-5:30PM) TR(4:15PM-5:50PM)	–	N
math022(1)	Ulam	MW(6:00PM-7:15PM)	–	L	math022(3)	Ulam	TR(2:30PM-3:45PM)	–	N
math315(1)	Wilkes	MWF(10:10AM-11:00AM)	H	H	math315(1)	Wilkes	MWF(10:10AM-11:00AM)	H	H
math318(1)	Wilkes	MWF(11:15AM-12:05PM)	H	H	math318(1)	Wilkes	MWF(11:15AM-12:05PM)	H	H
cmpsc360(1)	Wilkinson	TR(2:30PM-3:45PM)	N	H	cmpsc360(1)	Wilkinson	TR(2:30PM-3:45PM)	N	H
cmpsc360(2)	Wilkinson	TR(9:45AM-11:00AM)	H	L	cmpsc360(2)	Wilkinson	MWF(12:20PM-1:10PM)	N	H
math200(1)	Zeno	TR(9:45AM-11:00AM)	–	H	math200(1)	Zeno	TR(9:45AM-11:00AM)	–	H
math041(2)	Zermelo	MWF(12:20PM-1:10PM)	–	N	math041(2)	Zermelo	MWF(10:10AM-11:00AM)	–	H
math017(1)	Zorn	MWF(12:20PM-1:10PM)	–	H	math017(1)	Zorn	MWF(12:20PM-1:10PM)	–	H
math017(2)	Zorn	T(6:00PM-9:00PM)	–	N	math017(2)	Zorn	MWF(1:25PM-2:15PM)	–	H
math021(1)	Zorn	MWF(9:05AM-9:55AM)	–	L	math021(4)	Zorn	MW(6:00PM-7:15PM)	–	H

Continued on next page

Table A.21 Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 2$ - continued from previous page

Input Schedule			Preferences		Result			Preferences	
Course	Professor	Timeslot	Prof	Course	Course	Professor	Timeslot	Prof	Course
math021(4)	Zorn	MW(6:00PM-7:15PM)	–	H	math021(5)	Zorn	TR(1:00PM-2:15PM)	–	N
math035(1)	Zorn	MWF(10:10AM-11:00AM)	–	H	math035(1)	Zorn	MWF(10:10AM-11:00AM)	–	H

Table A.21: Comparison of Complete Input Schedule and Resultant Schedule for $\Delta = 2$

References

[1] Abramson, D. and J. Abela. A Parallel Genetic Algorithm for Solving The School Timetabling Problem, *15th Australian Computer Science Conference*, pp. 1–11. 1992.

[2] Bui, T. N. and B. R. Moon. Genetic Algorithm and Graph Partitioning. *IEEE Transactions on Computers*, 45(7), pp. 841–855, 1996.

[3] Burke, E.K. and J.P. Newall. A Multistage Evolutionary Algorithm for the Timetable Problem. *IEEE Transactions on Evolutionary Computation*, 3(1), pp. 63–74. IEEE, 1999.

[4] Carter, M.W. and G. Laporte. Recent developments in practical course timetabling. *Practice and Theory of Automated Timetabling II*, pp. 3–19. Springer Berlin Heidelberg, 1998.

[5] Dueck, G. New Optimization Heuristics: The Great Deluge Algorithm and The Record-to-Record Travel. *Journal of Computational Physics*, 104(1), pp. 86–92. Elsevier, 1993.

[6] Eiben, A.E. and J.E. Smith. Introduction to Evolutionary Computing. Springer, New York, 2003.

[7] Even, S., A. Itai and A. Shamir. On the Complexity of Timetable and Multicommodity Flow Problems. *Foundations of Computer Science, 16th Annual Symposium*, pp. 184–193. IEEE, 1975.

[8] Ghaemi, S., M.T. Vakili, and A. Aghagolzadeh. Using a genetic algorithm optimizer tool to solve University timetable scheduling problem. *9th International Symposium on Signal Processing and Its Applications*, pp. 1–4. IEEE, 2007.

[9] Holland, J. H. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. U Michigan Press, 1975.

[10] Irene, H. S. F. and S. Deris and S. Z. M. Hashim. University Course Timetable Planning Using Hybrid Particle Swarm Optimization. *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pp. 239–246. ACM, 2009.

[11] Raghavjee, R. and N. Pillay. An Informed Genetic Algorithm for the High School Timetabling Problem. *SAICSIT*. ACM, 2010.

[12] Sigl, B., M. Golub, and V. Mornar. Solving Timetable Scheduling Problem Using Genetic Algorithms. *Proceedings of the 25th International Conference on Information Technology Interfaces*, pp. 519–524. 2003.

[13] Turabieh, H. and S. Abdulla. An Integrated Hybrid Approach to The Examination Timetabling Problem. *Omega*, 39(6), pp. 598–607. Elsevier, 2011.

[14] Wang, YZ. Using Genetic Algorithm Methods to Solve Course Scheduling Problems. *Expert Systems with Applications*, 25(1) pp. 39–50. Elsevier, 2003.