S	
E1	
E2	
For Office Use Only	

Masters Project Final Report

(MCS)

March 2018

Project Title	Impact of dynamic soft constraints on generation of timetables for UCSC postgraduate courses
Student Name	P. B.K.P. Bogahawatta
Registration No. & Index No.	2014/MCS/009 14440093
Supervisor's Name	Mr. G. P. Seneviratne

For Office Use ONLY

Impact of dynamic soft constraints on generation of timetables for UCSC postgraduate courses

P. B. K. P. Bogahawatta
2018

Impact of dynamic soft constraints on generation of timetables for UCSC postgraduate courses

A dissertation submitted for the Degree of Master of Computer Science

P. B. K. P. Bogahawatta
University of Colombo School of Computing
2018

Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name:

Registration Number:

Index Number:

_____

Signature: Date:

This is to certify that this thesis is based on the work of

Mr./Ms.

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name:

_____

Signature: Date:

Abstract

Course timetabling in general is considered a complex, and time-consuming task to be performed manually due to its NP-hard nature. The problem domain can be defined as assignment of events conducted by a set of lecturers into timeslots and rooms subject to a given set of hard and soft constraints. A timetable that satisfies all hard constraints is known as a feasible schedule. Even though the feasibility of a timetable depends on hard constraints, quality relies on soft constraints the system needs to satisfy. Minimizing the number of soft constraint violations increase the quality of the solution as it would satisfy user requirements to a greater extent. This implies that the quality of a timetable can be predicted with the unmet amount of soft constraints and the total number of soft constraints the system includes. But time is not a factor as such that can be foreseen. The amount of time that a system requires to generate a timetable may increase with the number of soft constraints to be met, for there are more conditions to check. Yet in contrast, it might even lead to a lesser time consumption with more constraints because there can be fixed/known allocations as well. Thus in this research an attempt was made to explore the impact of applying soft constraints on generation of timetables. The research involves an implementation of a software solution that incorporates a genetic algorithm. The genetic algorithm was selected for the implementation due to its evolving and global optimization nature. According to results generated by the system, there was no direct impact found on performance when trying to satisfy soft constraints, and the executions depended solely on hard constraints. Furthermore it was found that the size of initial population can make a great impact on the performance of genetic algorithm. Results showed that setting a higher amount of chromosomes as the initial population could minimize the number of generations to be evolved to find a solution, and also it could prevent premature convergence up to some extent.

Acknowledgement

I would like to express my sincerest gratitude to Mr. G. P. Seneviratne, my supervisor, who has guided me with great kindness over the course of this project.

I would also like to thank all the staff members of Postgraduate unit of University of Colombo, School of Computing for all the help given along the way.

Finally I would like to thank all the authors of references that have used throughout this project.

Table of Contents

List of Figures

List of Abbreviations

NP-hard - Nondeterministic Polynomial time hard

UCSC - University of Colombo School of Computing

PSO - Particle Swarm Optimization

SPSO - Standard Particle Swarm Optimization

SA - Simulated Annealing

GA - Genetic Algorithm

SGA - Simple Genetic Algorithm

HGA - Hybrid Genetic Algorithm

Chapter 1: Introduction

1.1 Overview

Educational administrators face a lot of hardships when generating timetables, especially when there is a considerable amount of classes to be scheduled, as they must ensure a various set of conditions to be fulfilled. Some of these conditions include the availability of adequate teaching resources for subjects, teacher availability in the appropriate classrooms with the appropriate student groups, presence of required hardware resources (computers, projectors etc.) in a classroom etc. Because of this complex nature of the task, planners are not always able to build schedules that are consistent with teaching requirements, and the schedules they come up with may not always satisfy teacher or student needs.

Due to the NP-Hard (Nondeterministic Polynomial time Hard) nature [1] of timetabling problem, there seems to be no shortcut or easy solution for it. Finding a single, simple answer from a general computer program, even with a high performance computer, is merely impossible as the amount of inputs (constraints and resource allocations) get increased. However, a computer system which is intelligent and has constraint optimization capabilities could assist generating a better solution for timetabling problem while consuming a shorter amount of time compared to linear solutions.

1.2 The problem

The problem domain can be defined as assignment of events conducted by a set of lecturers into timeslots and rooms subject to a given set of constraints such as the number of classrooms available, room capacities, number of students, lecturers' preferences and units they teach, available equipment in classrooms etc.

All of these requirements mentioned above can be further broken down or grouped into two categories, which are 'Hard Constraints' and 'Soft Constraints'. Hard constraints are those, which must be strictly satisfied under any circumstances for a schedule to be identified as feasible. Violation of a hard constraint would end up with infeasible timetables. 'Soft Constraints' as the name implies, are desirable, yet do not necessarily have to be satisfied for the timetable to be

feasible. Minimizing the number of soft constraint violations increase the quality of the solution as it would satisfy user requirements to a greater extent. Therefore an equation as below can be derived to identify the relationship between the quality of a timetable and the count of soft constraints.

$$Quality = \frac{no\ of\ satisfied\ soft\ constraints}{no\ of\ total\ soft\ constraints}$$

Even though the quality can be predicted as above, time is not a factor as such that can be foreseen. The amount of time that a system requires to generate a timetable may increase with the number of soft constraints to be met, for there are more conditions to check. Yet in contrast, it might even lead to a lesser time consumption with more constraints because there can be fixed/known allocations as well. Thus the goal of this research is to study whether there is a specific pattern of time variance as per the increase of soft constraint count.

Currently, thousands of automated systems are available to generate feasible time tables, but most of them are capable of satisfying only the hard constraints. The intension of this research is to study the impact of increasing the number of soft constraints in generating feasible and high quality timetables and to develop an intelligent system in order to assist the study by generating timetables for postgraduate courses at UCSC. The system is intended to implement in a way that it would be capable of satisfying as many soft constraints as possible.

1.3 Aims & objective of project

Research Question: What are the impacts of applying soft constraints into generation of feasible time tables?

The aim of this research is to study the impact of increasing the number of soft constraints in generating feasible and high quality timetables. Following objectives have been accomplished to achieve this aim.

- Review existing timetabling systems to gain an insight into how they perform, what type of hard constraints they have met and determine the type of algorithms that have been incorporated in such systems.

- Identify all required hard and soft constraints.
- Design and develop a system to generate timetables for postgraduate courses at UCSC with zero hard constraint violations while satisfying as many soft constraints as possible with the use of most suitable algorithm identified at the literature review.
- Evaluate the system to identify the effect of increasing the amount of soft constraints on feasibility and quality of the timetable generated.
- Prepare necessary documentations.

1.4 Scope of project

Timetabling problem can be viewed in various forms such as school timetabling, examination timetabling and university curriculum based course timetabling. From these types, curriculum based course timetabling has been addressed to narrow down the scope of the project.

Evaluation has been done by studying the effect of increasing the amount of soft constraints on feasibility and quality of the timetable generated, with the use of actual UCSC postgraduate course details. For this purpose postgraduate course details (subject selection by students, availability of lecturers and lecture rooms etc.) of past two semesters (2015 – semester 2 and semester 4) has used.

Following hard constraints have been identified and finalized to establish a solid problem scope.

- Course assignment constraint - all the available courses should be included in the solution.
- Student assignment constraint - one student should not be assigned for two courses at a given time.
- Lecturer assignment constraint - one lecturer should not be assigned for multiple courses at a given time.
- Lecture room capacity constraint – number of seats of a lecture room should be sufficient for the students who will be attending the lecture.

Following are the soft constraints that have used to implement the project.

- Resource assignment constraint - all the resources (like chairs, computers, projectors etc.) should be available in an allocation.

- Max time preference constraint – maximum teaching hours of lecturers should not exceed their preference.
- Room space constraint - large rooms should not be assigned for courses with small number of students.

1.5 Structure of the thesis

The whole report consists of six main chapters, and the organization of chapters are as follows:

Chapter 2 of this report outlines the literature review studies carried out including a definition of the timetabling problem, followed by an extensive research made on several approaches to the timetabling problem. Chapter 3 covers the design aspects of the system, which looks into design strategies that will be used in order to aid with the implementation process. Chapter 4 of the report covers implementation details along with details about how a solution has proposed to timetabling problem in UCSC. Chapter 5 is about evaluation results while Chapter 6 concludes the report with further work than can be done to improve the system.

Chapter 2: Background

2.1 Introduction

This chapter of the report is wholly based on literature review studies that were carried out in order to gain an insight into the course timetabling problem, which includes gaining knowledge on what the timetabling problem is in general, followed by a thorough study on existing solutions for timetabling problem, how they perform, what type of constraints they have met and most importantly about what type of algorithms that have been incorporated in such systems.

2.2 Timetabling problem in general

The timetabling problem may be viewed in various forms such as school timetabling, examination timetabling and University curriculum based course timetabling. The timetabling problem itself in general is considered a hard task to perform manually since it involves in a lot of 'decision making', and therefore it is classed amongst the NP-hard (Nondeterministic Polynomial time hard) problems.

As this project mainly focuses on the curriculum based course timetabling problem, it is first important to understand the key properties on which the solution should be built upon. Consequently, a thorough literature study has been carried out targeting its core constraints that should be met in order to produce a feasible timetable, which can be seen in the following sections.

In the university curriculum based course timetabling problem, each course consists of a fixed number of lectures that should be allocated to distinct time slots, which are attended by student groups and a single lecturer who should not be present in more than one class at a time.

Rooms are identified by specific capacities (seating availability). In this case, rooms should provide sufficient resources for student groups that attend those classes. Correspondingly, rooms should also meet material requirements. For instance, if a lab session is to be held, then the room accommodating that class should be equipped with necessary lab material for all the students.

Thus in brief, a curriculum can be defined as a set of courses which can/may contain students in common. i.e. "Mobile Computing" and "e-Learning Concepts and Technologies" are common for both MCS and MIT students.

2.3 Existing systems

As the timetabling problem has been subject to an extensive research area since the past half century [2], there exists a range of solutions implemented in order to automate the problem by various researchers and individuals using many different algorithms.

The *FET* - Free Timetabling Software is one such open source system written in C++, licensed under GNU GPL; that allows scheduling of school, high school and university curriculum based timetables [3]. The FET project first began on the 31st of October 2002, and it was initially implemented using a Genetic Algorithm. However, considering it performance wise, the algorithm was found to be consuming a lot of time to solve schedules and only had the capability of solving only easy timetables. As a result later in June 2007, a new algorithm had been discovered by FET researchers that is capable of solving difficult timetables and in less amount of time, which they have named it *'recursive swapping'* (a heuristic approach). With the newly implemented recursive swapping algorithm, the FET system is capable of solving complicated problems within a time frame of 5-20 minutes, where simpler timetables would be solved in less than 5 minutes; and much larger, extremely difficult timetables taking a longer time as a matter of hours.

'Automated System for University Timetabling' is another system which is based on an iterative forward search algorithm and developed by Keith Murray and Tomas Muller using the Enterprise Edition of Java 2 (J2EE), Hibernate, and Oracle Database [4]. They have tested the system for large data sets, (*I.e. 800 classes, 50 rooms, 86,000 class requests)* and the system could be able to provide better and stable solutions for each and every test. Authors have stated that it took approximately 10 minutes for the system to come up with a complete and high quality solution, which is a substantial progress over a week of manual work.

Another system developed for Purdue University by Keith Murray, Tomas Muller and Hanna Rudova, provided a solution for course timetabling problem via a search for a complete assignment of times and rooms to classes, taking all hard and soft constraints into account [5]. They have used two types of algorithms namely generic iterative forward search with conflict-based statistics, and branch and bound. The solution is already in use on most of the course timetabling problems encountered each term at Purdue University.

2.4 Algorithms

Following are some of the algorithms that researchers have incorporated in their solutions to timetabling problem.

2.4.1 Graph coloring

Graph coloring is a technique of coloring the entities of a graph in a way that no two adjacent entities are of same color. Different limitations can be set on the graph, or on the way a color is assigned, or even on the color itself. Details of the problem define the structure of the graph. The corresponding graph contains a vertex for every entity and an edge for every conflicting pair of entities.

In 1967, Welsh and Powell [6] highlighted the similarity between timetabling problem and the coloring of the vertices of a graph by taking the vertices to be equivalent to courses and the arcs between them to represent conflicts. In that approach, coloring the graph is equal to placing courses in appropriate periods. A similar algorithm was presented by Broder [7] where vertices are ordered according to degree and the coloring of graph was attempted without using an upper limit on the number of colors.

Another approach that uses a combination of graph coloring and room allocation algorithms was proposed by a group of researchers in the Department of Computer Science at the University of Nottingham [8]. In that solution, the problems of intractability have been overcome by producing a spreadsheet type system that the user can guide in an informed manner. As a result, the users of the solution were given the control of the search and the possibility of backtracking where no realistic solution is found.

2.4.2 Integer linear programming

Integer linear programming is a mathematical optimization technique in which some or all of the variables are restricted to be integers. In this approach the objective function and the constraints are linear and binary decision variables indicate whether an entity is assigned to another or not.

The Department of Electrical and Computer Engineering of the University of Patras in Greece uses a binary linear integer programming model developed by Daskalaki and the group. [9] The objective of the model is to minimize a linear cost function. There are two major terms that the cost function consists of as follows;

- cost of allocating a module to a given time slot
- cost of allocating modules to a given day of the week

Another binary integer programming approach for course timetabling problem was introduced by Schimmelpfeng and Helber [10] and used in the School of Economics and Administration of the University of Hannover in Germany. With the use of an objective function that focuses on minimizing violations of soft constraints, the researchers could achieve good results in a way that 99% of the respondents (lecturers) were satisfied with the new system.

However, the assumptions made in linear programming are unrealistic at times, because a linear relationship assumes that factors never really change, when in reality they do. Finally, limiting the range of the problem also limits the possible solutions that are given in the problem.

2.4.3 Tabu search

Tabu Search is a meta-heuristic local (neighborhood) search algorithm that is used in optimization problems. It was formally introduced by Fred Glover in 1989. [11] As mentioned earlier, it makes use of a *'neighborhood'* search procedure to recursively move from one solution (X) to an improved solution of it (X1) in the neighborhood. To find a feasible solution, this process would take place until a threshold score has been met for the solution, which is usually referred to as the satisfaction of stopping criteria.

In general, all local search procedures comprise of a minor drawback that is, when exploring their neighborhoods, they often get stuck in poor scoring areas which are known as Plateaus and Ridges where no better solution can be found - resulting in them being all equally fit. In order to avoid these anomalies, and explore the rest of the search space that is left unexplored by search procedures, Tabu search makes use of a set of memory structures called the Tabu list, which is a set of rules and recently visited spaces in order to filter future visits to solutions in the neighborhood.

To define further, the memory structures used in Tabu search can be divided into three main sections:

- Short term: list of recently considered solutions, if one of these solutions in the list are to be reconsidered by a potential solution, it will be restricted (cannot be revisited) until it has reached its expiration.
- Immediate term: a set of rules that biases the search towards promising areas.
- Long term: rules that aid in backtracking the search; in other words if an anomaly is met then these rules diverts the search to another route.

In relation to the timetabling problem, Hertz [12] and Schaerf [13] have proposed solutions with the use of Tabu search. In the methods both of them have proposed that a feasible solution could be found by joining the generated potential solutions continuously, rather than a random generation. This enables the neighboring solutions to repair their objective functions effectively which improves the quality of scheduling by keeping the number of conflicts to the minimum.

A major issue with Tabu search is that it works at its best (effective) only in discrete spaces. In order to overcome this problem however, a similarity measure can be implemented to reject solutions that violate the similarity threshold.

Yet once again if a search space is too large in terms of dimensionality, there is a possibility that the search will be limited to a smaller area. Therefore in the implementation of a Tabu search, it is always important to consider the problem in smaller portions rather than as a whole.

2.4.4 Particle swarm optimization

Particle Swarm Optimization (PSO) is an intelligent simulation of birds foraging behavior, which is as the name implies used as an optimization algorithm to discover distributed solutions to complex problems, using interactions between simple agents and their environment. It was developed by Kennedy and Eberhart [14] in 1995. In PSO, each individual, named as a particle, update its own velocity and the position (pbest – personal best) in each iteration. And also they refer to the information from other members of the group (gbest – global best) and then use that information as well to choose the next best foraging sites. As iterations go on, each individual

would be able to choose a best site eventually. At the end of an iteration, the performance of all particles will be evaluated by some predefined cost functions.

A meta-heuristic algorithm, based on the principles of PSO was proposed by Der-Fang [15] to solve the course timetabling problem. In this algorithm both instructors and students are allowed to specify their preferences. For example, instructors have the choice to maximize teaching free hours, and they can state the preferred lecturing format for course sections. Results of this study have demonstrated that the proposed algorithm was better than the approach of Genetic Algorithms proposed in the literature.

A new PSO named standard PSO (SPSO) was proposed by Ruey-Maw Chen and Hsiao-Fang Shih [16] to solve the university course timetabling problem. In their approach the particles were encoded based on timeslots rather than study hours to reduce the computational complexity.

Moreover, an interchange heuristic has used to search for neighborhood solution space and thus enhanced the quality of solution.

PSO is a promising scheme for solving complex problems due to its fast convergence, fewer parameter settings and ability to fit dynamic environmental characteristics.

2.4.5 Simulated annealing

Annealing is a concept of heating physical solid objects and altering their state in order to increase the ductility, ultimately making them more flexible to work around. The process involves in heating objects above their critical temperature, secondly maintaining that temperature in a sustainable manner, and then lastly cooling at a gradual pace. Annealing an object enables it to be stretched by softening the object, which makes it to relieve its internal atomic stresses and refine its structure by making it homogenous.

The concept of Simulated Annealing (SA) was first suggested and applied by David Abramson (1991), where he implemented it on a local search technique that transpired universal concentration keeping the probabilities to the minimal [17]. In the same year, Abramson applied the technique to the timetabling problem as well, where he substituted (simulated) the properties of annealing to the problem as follow.

The actual physical objects (particles) are substituted with the subject elements and the system energy with the cost of the timetable (the cost accumulated to model the timetable). Firstly, an initial allocation is made on subjects, where they are allocated to randomly chosen timeslots. Next, the initial cost is calculated along with a predetermined temperature value. In the application, the cost function comprises of a major role that determines the feasibility or the fitness of the solution; just as in the original process, the system energy function determines the ductility of a particle being annealed. The temperature in this case is used to control the increase of cost in terms of probability and can be linked with a temperature of an actual physical object (particle).

Another researcher, Phillip Kostuch [18] proposed an algorithm with the use of graph coloring heuristics to check for the feasibility and then used Simulated Annealing methodology to satisfy the soft constraints, by ordering time slots and then swapping events between them. He has used cooling schedule criteria to reach the optimum. To be briefer, Kostuch used Simulated Annealing techniques to provide an optimal solution to course timetabling problem by swapping individual elements between time slots.

Ruggero Bellio et al have also proposed a solution for curriculum-based course timetabling problem based on Simulated Annealing techniques. [19] In their approach, classrooms play an important role for the cost of a solution. The cost component was named *'RoomStability'* which states that all lectures of the same course should be conducted in the same room. They have also used a non-geometric cooling scheme for Simulated Annealing, in which the temperature decreases quickly in the beginning of the search and then slowly towards the end, to speed-up the process at high temperatures.

Simulated Annealing is considered to be an iterative concept, and as mentioned above the typical algorithm accepts a new solution if the cost is found to be lower than the cost of the solution it iterated from. Similarly, there is also a probability that new solutions are accepted even at higher costs as long as they are within the margin of a set temperature. This specific acceptance criterion enables the algorithm to overcome anomalies like local minima.

2.4.6 Genetic algorithms

The Genetic Algorithm (GA) in its simplest terms can be explained as a model of machine learning/search heuristic, where its behavior has derived from the process of natural evolution. In heuristic terms, the GA reaches an optimal solution by evolving an initial set of individuals from a population towards a better solution, where new generations (optimized solutions) are created at each iteration of evolution. The candidate solutions or the individuals consist of a set of properties, which are usually altered and mutated in the process of evolution.

GAs have been used to provide solutions for timetabling problem since 1990 [20]. Since then there are a plenty of papers have been published investigating and applying GA methods for the curriculum based course timetabling problem.

One of them is the guided search genetic algorithm proposed by Naseem and Yang. In their approach a guided as well as s local search technique are integrated into a steady state genetic algorithm [21]. In the research paper it has also been mentioned that the authors believe, this is the first such algorithm aimed at the domain of course timetabling problem. They have tested the performance of proposed algorithm, by carrying out some experiments based on a set of benchmark problems to compare it with a set of state-of-the-art methods from the literature. The experimental results have been demonstrated that the proposed solution is competitive and work well across all problem instances in comparison with other approaches studied in their literature.

Enzhe Yu and Ki-Seok Sung have proposed another solution for university weekly course timetabling problem, with the use of a sector-based genetic algorithm [22]. They have introduced the concept of "sector" and applied it to the initialization, crossover, and mutation procedures. A routine named "check-and repair" has been adopted with hard constraints to keep the solutions in a feasible space. According to the experiments that were carried out to evaluate the proposed solution they could achieve promising results even on a university's real data.

2.5 Conclusion

In this chapter, several heuristic approaches are seen with regards to the timetabling problem, with their core objectives being producing feasible timetables; in other words, being able to generate timetables that are clash free for each activity, given those that share the same resources. Every

approach looked at, consists of at least a minor amount of difficulties in being able to produce feasible timetables satisfying all soft and hard constraints. In contrast, GA provides functionalities to maintain a diversity among good solutions while allowing to have multiple solutions. Generally GA is used to solve multi-objective optimization problems, when there is an idea on what a solution looks like but cannot figure out the way to reach that solution. In order to reach the solution GA facilitates traversing the state space in parallel without much of communication delays. Considering all these factors, in this specific problem, the GA would suit to provide a better solution mainly due its evolving and global optimization nature.

Chapter 3: Analysis and Design

3.1 Introduction

From this chapter of the report, it is intended to cover the design aspects of the solution. These aspects include the design process and strategies that have been used for the implementation. Therefore this chapter discusses about some selected algorithms that would be appropriate to generate timetables efficiently, along with the means that the problem can be encoded into each of that algorithm.

3.2 Alternate solutions

As concluded in the previous chapter, the implementation will be consisting on GA to solve the timetabling problem. Even though GA is the selected algorithm, SA and PSO are also good fits because they also are evolutionary algorithms. This section of the chapter explains about how SA and PSO can be used to solve the timetable generation problem.

3.3.1 Particle swarm optimization

In PSO, a bird in a flock is represented as a particle, and the swarm is composed of a group of such particles. The position of a particle is regarded as a candidate solution to an optimization problem. Particles are given a fitness function designed with regard to the problem. When each particle moves to a new position in the search space, it remembers its personal best (Pbest) and exchange information with other particles to remember the global best (Gbest). Then, each particle revises its velocity and direction according to the Pbest and the Gbest in memory to move towards the optimal value and find the optimal solution eventually. This whole process has been depicted in Figure 3.1.

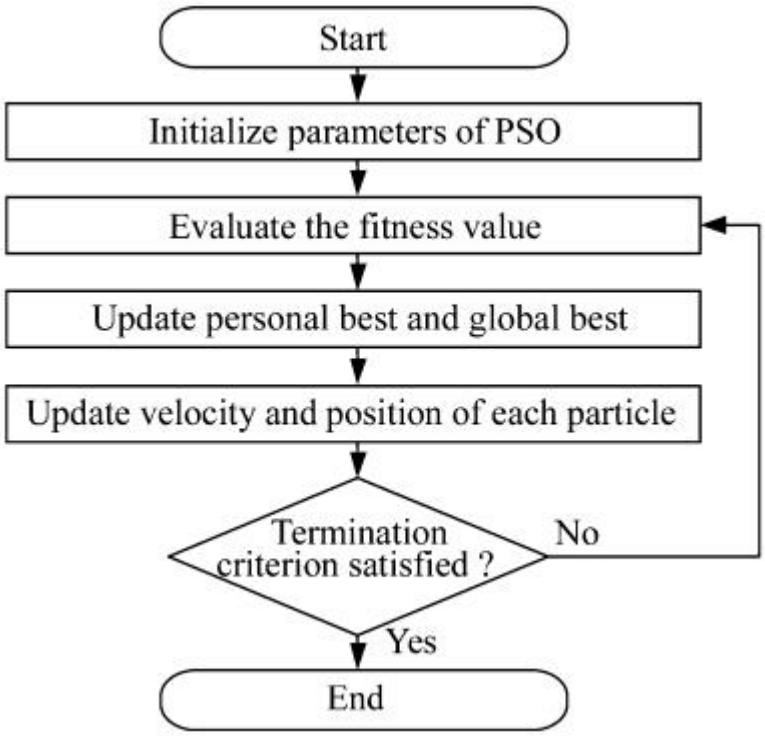


Figure 3.1: Basic steps of PSO

With regard to the timetabling problem, a candidate solution or a feasible timetable has to be considered as a particle. Therefore a particle can be represented with a structure consisting of a timeslot, a room and a class to solve the timetabling problem with the use of PSO.

3.3.2 Simulated annealing

In SA, the first step is to define a starting point together with an initial temperature value assigned to the function. The function is re-evaluated at a new point after making a random move, away from the initial position. If the new value is an improvement (superior value), the details of the movement is preserved in memory. If not, the temperature details will be used to compare with a randomly generated value which is within the range of the maximum and minimum temperatures. If this random generated value is less than the current temperature, a move will be made. Otherwise the move has to be ignored and continue the process. The temperature is reduced by a small value after each iteration for the function to be terminated properly. The diagram below is a graphical representation of this process.

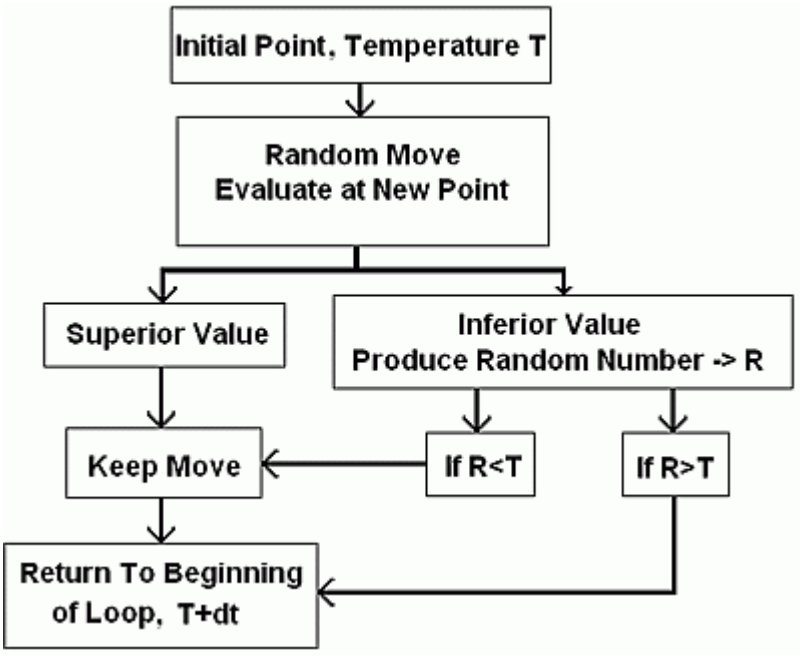


Figure 3.2: Basic steps of SA

A feasible timetable can be represented by a particular position a particle resides after a move. A cooling schedule should also be defined consisting of starting temperature, temperature decrement function, and termination condition.

3.3 Problem representation

Scheduling process of a curriculum based course timetable comprises of five main properties in general, which are:

- Set of time periods
- Room allocations
- Subject allotments
- Groups of students
- Lecturer assignments (to classes/subjects/student groups)

These major properties will be used to encode the problem in to algorithmic representation of GA

The GA process usually starts from a population of randomly selected individuals, and this process is known as *'Initialization'*. These individuals are generally represented by Chromosomes, in a way that they consist of a fixed length sequence, and are usually represented in the form of binary multiple coding (Figure 3.3) with zeros and ones also known as Strings.

0	1	1	1	0	1	0	0	1	0

Figure 3.3: Binary coding representation of a chromosome

However, in order to facilitate the representation according to the problem, it is possible to utilize other forms as well to represent individuals, such as an integer coding representation or a decimal coding representation. Figure 3.4 and Figure 3.5 below depicts these two representations.

1	3	4	-2	9	0	1

Figure 3.4: Integer coding representation of a chromosome

1.3	-4.8	2.1	3.001	9.99

Figure 3.5: Decimal coding representation of a chromosome

All initialized individuals comprise of a fitness value; and in the selection process, based on each individual's fitness function it is decided who get to proceed for Crossover (to reproduce) from their environments. The above-mentioned fitness function that is used to select the fittest individuals may be implemented in two forms. The simplest form being the fitness proportionate selection, where individuals are simply selected through a fitness measure, based on the fitness function and decided who will be the most suitable to undergo GA operations such as Crossover. The other implementation is called the tournament selection, where it is based on a model that randomly selects individuals from a subgroup and makes them compete amongst themselves in order to select the fittest.

In terms of optimizing a GA, mutation can be seen as an important aspect of the evolution process. The purpose of the mutation function is to maintain diversity on generated individuals of a population to the next, and the freedom is all up to the user, how he wishes to manipulate the mutation function; however it is always important to keep the mutation rate or the probability, set

to a low difference. As higher the rate of probability the search will become too random in terms of generating fit population of individuals.

Figure 3.6 below depicts the whole process explained above in detail.

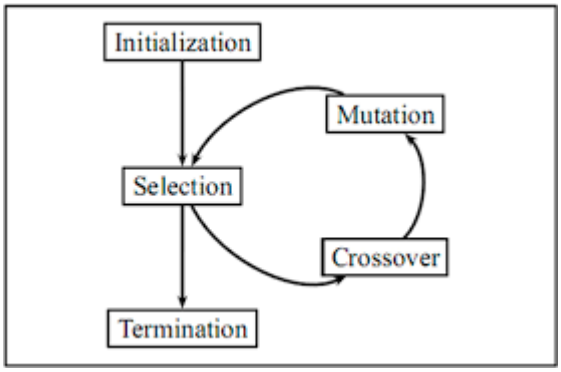


Figure 3.6: Basic steps of GA

Considering the timetabling problem, it is first important to identify the scheduling problem in terms of genes to be able to map it into functionality of the Genetic Algorithms. This mapping of problem properties into genes, and then to the genetic algorithm consists of a number of encoding and decoding procedures. A schedule for a whole week was identified as a possible solution (chromosome) which defines how a curriculum should be arranged including which time slots to place, in which room should be allocated, and which courses should be allotted. Figure 3.7 below depicts the format of a candidate solution for course timetabling problem.

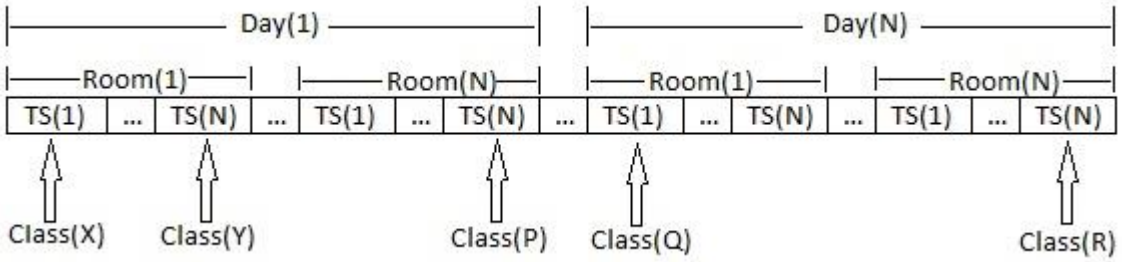


Figure 3.7: Chromosome representation for course timetabling problem

In this diagram TS stand for “Time Slot” and a class is a collection of a specific subject, a lecturer and a student group. Fitness would be calculated according to reward on achievement approach. For every satisfaction of a constraint, individuals will be rewarded with one point. If an individual

breaks a rule at any time slot it will not be rewarded with points. Finally the fitness would be calculated by summing up all the rewards an individual has earned.

3.4 Conclusion

In this chapter it has been discussed about several algorithms that would be appropriate to solve the timetabling problem. Further the chapter explained how the timetabling problem has been encoded into Genetic Algorithm.

Chapter 4: Proposed Solution

4.1 Introduction

As concluded in the previous chapters, the implementation consists on a Genetic algorithm to solve the timetabling problem. With the utilization of GA framework, feasible solutions may be reached or found in a polynomial time, which can be accepted. From this chapter, it is intended to cover the areas of identification and implementation of base classes, how the algorithm has been put in use, and the system's technical aspects etc.

4.2 Algorithm perspective

It is mentioned in the previous chapter how individuals of a population should be represented by chromosomes or genes as an initial step in order to proceed with operations of evolution. The solution starts with a population of randomly generated chromosomes and its quality is evaluated through the fitness function. This fitness evaluation determines whether the chromosome needs to be reserved as an elite object or if it needs to be improved by further operations like crossover and mutation. If all the hard constraints are satisfied, the fitness score will be returned with a value of 1.0, and the algorithm terminates since a solution has found.

4.3 Technology perspective

In the solution, Chromosomes are represented by a List of TimeSlot objects. A TimeSlot object contains details about lecture (module and the lecturer who teaches it), lecture room, start and end time of a lecture and the day of week the lecture is going to be scheduled. During the initialization of chromosomes lectures that are to be scheduled are allocated into randomly selected lecture room and a time. Initialization procedure completes once all the lectures are allocated into available timeslots and the population count becomes the size specified in database. Some meta-data like population count, replace count, crossover and mutation probabilities are stored in the database so that it is easier to check the performance of generating feasible solution for dynamic values of above mentioned variables.

Java method depicted by Figure 4.1 below shows how a random chromosome is initialized.

```
98    public ChromosomeConfig makeNewFromPrototype(ChromosomeConfig chromosomeConfig) {
99        // Make a new chromosome by just copying the current chromosome
100       ChromosomeConfig newChromosome = context.getBean(ChromosomeConfig.class).makeCopy(chromosomeConfig);
101
102       // Re-initialize timetable specific details
103       newChromosome.initData(chromosomeConfig.timetableDetails);
104       int numberOfTimeSlots = newChromosome.slots.size();
105
106       for (Long lectureId : newChromosome.timetableDetails.getLectures()) {
107           Lecture lecture = lectureRepository.findOne(lectureId);
108
109           boolean lectureAllocated = false;
110
111           while (!lectureAllocated) {
112               TimeSlot randomTimeSlot = newChromosome.slots.get(random.nextInt(numberOfTimeSlots));
113               if (randomTimeSlot.getLecture() == null) {
114                   randomTimeSlot.setLecture(lecture);
115                   lectureAllocated = true;
116               }
117           }
118       }
119
120       newChromosome.calculateFitness();
121       return newChromosome;
122   }
```

Figure 4.1: Creation of a random chromosome

Once the initial population of chromosomes is generated, the system performs crossover and mutation operations to generate offsprings. This process continues until the offspring count reaches the value of replace count specified in database or an offspring with fitness value 1.0 is generated. When generating offspring chromosomes with less fitness values get replaced by the ones with higher fitness so that the population always contains better solutions, ensuring the theory of "Survival of the fittest".

As a result of replacing poor solutions with better ones there can be a situation where the population contains chromosomes with same fitness values. Because of this the mean fitness of population might not be increased together with the generation count. This kind of situations are called "Premature Convergence" [23] and to prevent such situations special correction and mutation methodology has been utilized.

Code snippet mentioned below as Figure 4.2 shows how offsprings are generated in the system and how it prevents situations like premature convergence.

```
// Produce offsprings
for(int i = 0; i < replaceCount; i++) {
    int parentIndex = random.nextInt(chromosomes.size());
    ChromosomeConfig parent = chromosomes.get(parentIndex);
    // Perform crossover and mutation
    // Same gene has been used for crossover operation to avoid omission of classes
    ChromosomeConfig crossedOverOffSpring = parent.crossOver();
    ChromosomeConfig mutatedOffSpring = parent.mutate();
    ChromosomeConfig offSpring;

    if(crossedOverOffSpring.getFitness() > mutatedOffSpring.getFitness()) {
        offSpring = crossedOverOffSpring;
    } else {
        offSpring = mutatedOffSpring;
    }

    addOffspringToPopulation(offSpring);

    if(offSpring.getFitness() >= offSpring.getFitnessMargin()) {
        ChromosomeConfig improvedOffSpring = offSpring.improveChromosome();
        if(improvedOffSpring.getFitness() > offSpring.getFitness()) {
            addOffspringToPopulation(improvedOffSpring);
        }
    }
}

currentGeneration++;
```

Figure 4.2: Produce offsprings and prevent premature convergence

4.3.1 Crossover

The Genetic Algorithm's actual evolution process (reproduction) starts from the crossover operation where it combines information (DNA in terms of Biology) of two parents and creates offsprings as a result. In referring to the following code segment of the Crossover operation shown in Figure 10, same chromosome has been used as both the parents to avoid any missing of lectures during crossover operation. Also in this way offspring will be diverged enough from its parent and because of that the population will contain a good set of candidate solutions with a huge variety. The process can be described as follows:

- Pick a random point to divide the "slots" list into two
- Split the list into two portions
- Insert lecture details of second portion from first index of the offspring
- Append the first portion of parent into offspring

- Calculate fitness of the generated offspring

```
124    public ChromosomeConfig crossOver() {
125        // Check probability of crossover operation
126        if (random.nextInt(100) > crossoverProbability) {
127            // No crossover, just return the first parent
128            return this;
129        }
130
131        ChromosomeConfig parent = this;
132
133        ChromosomeConfig offSpring = context.getBean(ChromosomeConfig.class).makeCopy(parent);
134        offSpring.fitness = 0;
135        offSpring.slots.clear();
136
137        int timeSlotIndex = getRandomTimeSlotIndex(parent);
138        int index = 0;
139
140        for (TimeSlot templateTimeSlot : offSpring.timetableDetails.getTimeSlotsTemplate()) {
141            TimeSlot timeSlot = new TimeSlot(index, templateTimeSlot);
142
143            int noOfTimeSlots = parent.slots.size();
144            if (index < noOfTimeSlots - timeSlotIndex) {
145                timeSlot.setLecture(parent.slots.get(timeSlotIndex + index).getLecture());
146            } else {
147                timeSlot.setLecture(parent.slots.get(index - noOfTimeSlots + timeSlotIndex).getLecture());
148            }
149            offSpring.slots.add(timeSlot);
150            index++;
151        }
152
153        offSpring.calculateFitness();
154        return offSpring;
155    }
```

Figure 4.3: Crossover Operation

4.3.2 Mutation

Exchange mutation methodology has been used as the mutation operation, where in each mutation attempt, two random positions from slots are chosen and the lectures in those positions are swapped. Code segment of the mutation operation is shown in Figure 4.4 below.

```java
public ChromosomeConfig mutate() {
    // Check probability of mutation operation
    if (random.nextInt(100) > mutationProbability) {
        // No mutation due to probability, Just copy the first parent
        return this;
    }

    ChromosomeConfig offSpring = context.getBean(ChromosomeConfig.class).makeCopy(this);
    offSpring.fitness = 0;
    int timeSlotIndex1 = getRandomTimeSlotIndex(this);
    Lecture lecture1 = this.slots.get(timeSlotIndex1).getLecture();
    Integer timeSlotIndex2;

    do {
        timeSlotIndex2 = getRandomTimeSlotIndex(this);
    } while (timeSlotIndex1 != timeSlotIndex2);

    Lecture lecture2 = this.slots.get(timeSlotIndex2).getLecture();
    offSpring.slots.get(timeSlotIndex2).setLecture(lecture1);
    offSpring.slots.get(timeSlotIndex1).setLecture(lecture2);

    offSpring.calculateFitness();

    return offSpring;
}
```

Figure 4.4: Mutation Operation

Other than the mutation operation mentioned above, a special mutation methodology has also been utilized in the system to prevent premature convergence. In this method mutation probability has not checked and therefore the chromosome mutates forcefully. The new methodology adhere the rules of insertion mutation operation and a randomly picked up lecture is inserted into a random timeslot where another lecture is not currently available. Figure 4.5 below depicts the code for this operation.

```java
public ChromosomeConfig forceMutation() {
    ChromosomeConfig offSpring = context.getBean(ChromosomeConfig.class).makeCopy(this);
    offSpring.fitness = 0;

    List<Integer> timeSlotIndexesWithLecture = offSpring.slots.stream()
            .filter(timeSlot -> timeSlot.getLecture() != null).map(TimeSlot::getIndex).collect(Collectors.toList());

    List<Integer> timeSlotIndexesWithNoLecture = offSpring.slots.stream()
            .filter(timeSlot -> timeSlot.getLecture() == null).map(TimeSlot::getIndex).collect(Collectors.toList());

    if (!timeSlotIndexesWithNoLecture.isEmpty()) {
        int timeSlotIndexWithLecture = timeSlotIndexesWithLecture
                .get(random.nextInt(timeSlotIndexesWithLecture.size()));
        int timeSlotIndexWithNoLecture = timeSlotIndexesWithNoLecture
                .get(random.nextInt(timeSlotIndexesWithNoLecture.size()));

        Lecture lecture = offSpring.slots.get(timeSlotIndexWithLecture).getLecture();
        offSpring.slots.get(timeSlotIndexWithNoLecture).setLecture(lecture);
        offSpring.slots.get(timeSlotIndexWithLecture).setLecture(null);
    }

    offSpring.calculateFitness();
    return offSpring;
}
```

Figure 4.5: Special Mutation Operation

3.3.3 Fitness function

The next stage of the implementation is to relate a fitness evaluation to chromosomes to determine the feasibility and accuracy of the solution. In this case, when evaluating the fitness function only hard constraints are taken into account, and the rules are as follow. The code snippet for fitness function is depicted in Figure 4.6.

- Check if the same lecture has not been allocated multiple times
- Check for lecture room capacity
- Check for lab equipment requirements
- Check for student overlapping
- Check for lecturer overlapping

```
private void calculateFitness() {
    float score = 0;

    List<TimeSlot> timeSlotsWithLecture = this.slots.stream().filter(timeSlot -> timeSlot.getLecture() != null)
            .collect(Collectors.toList());

    // slots list contain records, more than available lectures
    if (timeSlotsWithLecture.size() > this.timetableDetails.getLectures().size()) {
        return;
    }

    for (TimeSlot currentTimeSlot : timeSlotsWithLecture) {
        Lecture currentLecture = currentTimeSlot.getLecture();

        List<TimeSlot> timeSlotsWithSameLecture = this.slots.stream()
                .filter(timeSlot -> timeSlot.getLecture() != null
                        && timeSlot.getLecture().getId() == currentLecture.getId()
                        && (timeSlot.getDay() != currentTimeSlot.getDay()
                        || timeSlot.getLectureRoom().getId() != currentTimeSlot.getLectureRoom().getId()))
                .collect(Collectors.toList());

        // Check if the lecture has been allocated more than once
        if (timeSlotsWithSameLecture.isEmpty()) {
            score++;
        }

        List<Long> courseIds = new ArrayList<>();
        for (Course course : currentLecture.getCourses()) {
            courseIds.add(course.getId());
        }

        List<Long> studentIds = studentCourseRepository.findDistinctStudentIdsByCourseIdIn(courseIds);
        int noOfStudentsPerLecture = studentIds.size();
        boolean enoughSeats = currentTimeSlot.getLectureRoom().getCapacity() >= noOfStudentsPerLecture;

        // Check for lecture room capacity
        if (enoughSeats) {
            score++;
        }

        // Check for lab equipment requirements
        boolean labRequirementSatisfied = true;
        if (currentLecture.isRequireLab()) {
            labRequirementSatisfied = currentTimeSlot.getLectureRoom().isLab();
        }
```

```
        if (labRequirementSatisfied) {
            score++;
        }

        List<TimeSlot> overlappingSlots = this.slots.stream()
                .filter(timeSlot -> timeSlot.getDay().equals(currentTimeSlot.getDay())
                        && timeSlot.getStartTime().equals(currentTimeSlot.getStartTime())
                        && timeSlot.getLectureRoom().getId() != currentTimeSlot.getLectureRoom().getId()
                        && timeSlot.getLecture() != null)
                .collect(Collectors.toList());

        Set<Long> overlappingSlotsCourseIds = new HashSet();
        Set<Long> overlappingLecturesLecturerIds = new HashSet();
        overlappingSlots.stream().forEach(timeSlot -> {
            Set<Course> courses = timeSlot.getLecture().getCourses();
            List<Long> ids = courses.stream().map(Course::getId).collect(Collectors.toList());
            overlappingSlotsCourseIds.addAll(ids);

            overlappingLecturesLecturerIds.add(timeSlot.getLecture().getLecturer().getId());
        });

        // Check for student overlappings
        boolean studentsOverlap = false;
        List<Long> overlappingSlotsCourseIdsList = new ArrayList<>();
        overlappingSlotsCourseIdsList.addAll(overlappingSlotsCourseIds);
        if (noOfStudentsPerLecture > 0 && !overlappingSlotsCourseIdsList.isEmpty()) {
            List<Long> overlappingStudentIds = studentCourseRepository
                    .findDistinctStudentIdsByCourseIdInAndStudentIdsIn(overlappingSlotsCourseIdsList, studentIds);
            studentsOverlap = overlappingStudentIds != null && !overlappingStudentIds.isEmpty();
        }

        if (!studentsOverlap) {
            score++;
        }

        // Check for lecturer overlappings
        if (currentLecture == null
                || !overlappingLecturesLecturerIds.contains(currentLecture.getLecturer().getId())) {
            score++;
        }
    }

    int divisor = this.timetableDetails.getLectures().size() * hardConstraintsCount;

    // Calculate fitness value based on score
    this.fitness = score / divisor;
}
```

Figure 4.6: Fitness Function

4.4 Conclusion

In this chapter it has been discussed about implementation details of the solution for UCSC course timetable scheduling system in both algorithmic and technological perspectives.

Chapter 5: Evaluation and Results

5.1 Introduction

This chapter provides an evaluation on the solution provided for timetable scheduling in UCSC for postgraduate courses. Evaluation of the system is experiment based, and has done by studying the performance of the system, with the use of actual UCSC postgraduate course details. For this purpose postgraduate course details (subject selection by students, availability of lecturers and lecture rooms etc.) of semester2 and semester 4, 2015 (two semesters) has been used. The basis of semester selection is the nature of courses students have to follow. Semester 2 contains compulsory courses that all the students should attend and Semester 4 has optional courses that students can select.

5.2 Functional evaluation

When executing the system, at initial stages there were some situations where fitness get random values without increasing as per the increase of generation count. The root cause for this behavior is found after several runs and the reason was identified as removing random chromosomes and inserting any offspring into population, generated from crossover and mutation operations. System executed for about 30,000 – 50,000 generations without a proper result. As a solution fitness value was checked before removing a chromosome from population and inserting a new one.

Then the program was executed for more than 5 hours with no fitness change even though the generation count increased up to 20,000. Research papers suggested that increasing the population size [24], dynamic application of crossover and mutation operators and partial re-initialization of population can avoid such occurrences of premature convergence [25]. Therefore a new mutation mechanism was introduced with the intension of changing the model of available solutions drastically. And that method was performed forcefully without checking a mutation probability because the probability of performing a mutation is only 3%. With the introduction of that new and forceful mutation mechanism system started to provide results with a 20 – 30 minutes time span, but there were still some executions where fitness was at a constant value.

As a solution to this problem another operation which is similar to mutation was applied to check for some constraint violations manually such as lab requirement satisfaction and lecture room capacity satisfaction, and fixed any violation if available. This methodology improved the system so that it can now always provide a solution within 10 – 20 minutes.

5.3 Performance evaluation

Program execution time and the number of generations the solutions have evolved are the basic criteria to evaluate system performance. After implementing the final solution performance of the system was evaluated by changing population size. Figures 14 depicts the evolution of fitness values when initial population is 500 and the replace count is 300. Table 5.1 contains fitness data used to plot the chart in Figure 5.1.

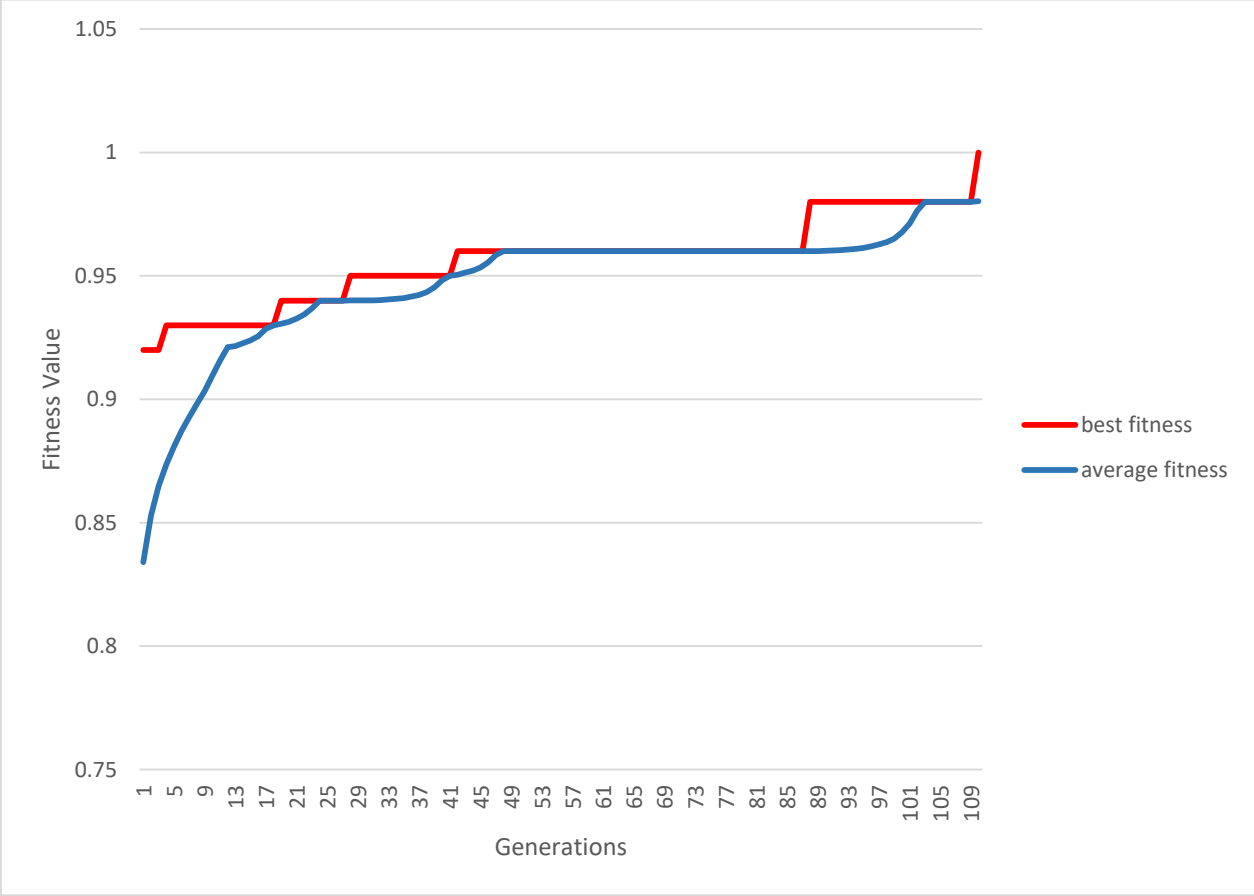


Figure 5.1: System performance with 500 population count

Generation	Best fitness	Average fitness
0	0.92	0.834
1	0.92	0.85286
2	0.92	0.86490001
3	0.93	0.873580004
4	0.93	0.880859999
5	0.93	0.887279994
6	0.93	0.892879991
7	0.93	0.898159991
8	0.93	0.903359994
9	0.93	0.909579999
10	0.93	0.915700009
11	0.93	0.921100016
12	0.93	0.921540015
13	0.93	0.922660014
14	0.93	0.923920013
15	0.93	0.925600011
16	0.93	0.928480009
17	0.93	0.930000007
18	0.94	0.930600007
19	0.94	0.931400006
20	0.94	0.932640005
21	0.94	0.934360003
22	0.94	0.936880001
23	0.94	0.939979998
24	0.94	0.939999998
25	0.94	0.939999998
26	0.94	0.939999998
27	0.95	0.940019998
28	0.95	0.940059998
29	0.95	0.940059998
30	0.95	0.940099998
31	0.95	0.940219997
32	0.95	0.940459997
33	0.95	0.940739997
34	0.95	0.940999997
35	0.95	0.941579996
36	0.95	0.942279995
37	0.95	0.943459994
38	0.95	0.945339993
39	0.95	0.94821999
40	0.95	0.949999988
41	0.96	0.950499988
42	0.96	0.951339987
43	0.96	0.952099986
44	0.96	0.953419985

Generation	Best fitness	Average fitness
45	0.96	0.955479983
46	0.96	0.95849998
47	0.96	0.959999979
48	0.96	0.959999979
49	0.96	0.959999979
50	0.96	0.959999979
51	0.96	0.959999979
52	0.96	0.959999979
53	0.96	0.959999979
54	0.96	0.959999979
55	0.96	0.959999979
56	0.96	0.959999979
57	0.96	0.959999979
58	0.96	0.959999979
59	0.96	0.959999979
60	0.96	0.959999979
61	0.96	0.959999979
62	0.96	0.959999979
63	0.96	0.959999979
64	0.96	0.959999979
65	0.96	0.959999979
66	0.96	0.959999979
67	0.96	0.959999979
68	0.96	0.959999979
69	0.96	0.959999979
70	0.96	0.959999979
71	0.96	0.959999979
72	0.96	0.959999979
73	0.96	0.959999979
74	0.96	0.959999979
75	0.96	0.959999979
76	0.96	0.959999979
77	0.96	0.959999979
78	0.96	0.959999979
79	0.96	0.959999979
80	0.96	0.959999979
81	0.96	0.959999979
82	0.96	0.959999979
83	0.96	0.959999979
84	0.96	0.959999979
85	0.96	0.959999979
86	0.96	0.959999979
87	0.98	0.960039979
88	0.98	0.960039979
89	0.98	0.960159979

Generation	Best fitness	Average fitness
90	0.98	0.960279979
91	0.98	0.960399979
92	0.98	0.96063998
93	0.98	0.96091998
94	0.98	0.961319981
95	0.98	0.961999983
96	0.98	0.962719984
97	0.98	0.963719986
98	0.98	0.965119989
99	0.98	0.967719994
100	0.98	0.971240001
101	0.98	0.976480012
102	0.98	0.980000019
103	0.98	0.980000019
104	0.98	0.980000019
105	0.98	0.980000019
106	0.98	0.980000019
107	0.98	0.980000019
108	0.98	0.980000019
109	1	0.980200019

Table 5.1: Fitness details with 1000 population count

Fitness evolution for execution with 1000 population size and 700 replace count is displayed in Figure 5.2 and fitness data that were used to plot the chart are displayed in Table 5.2.

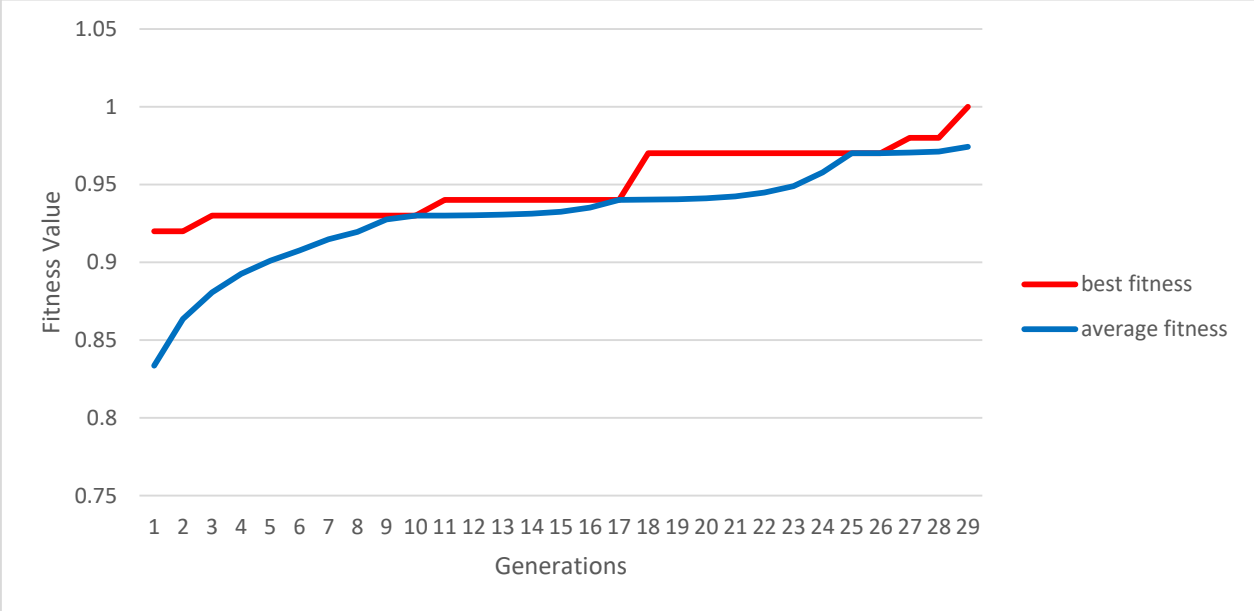


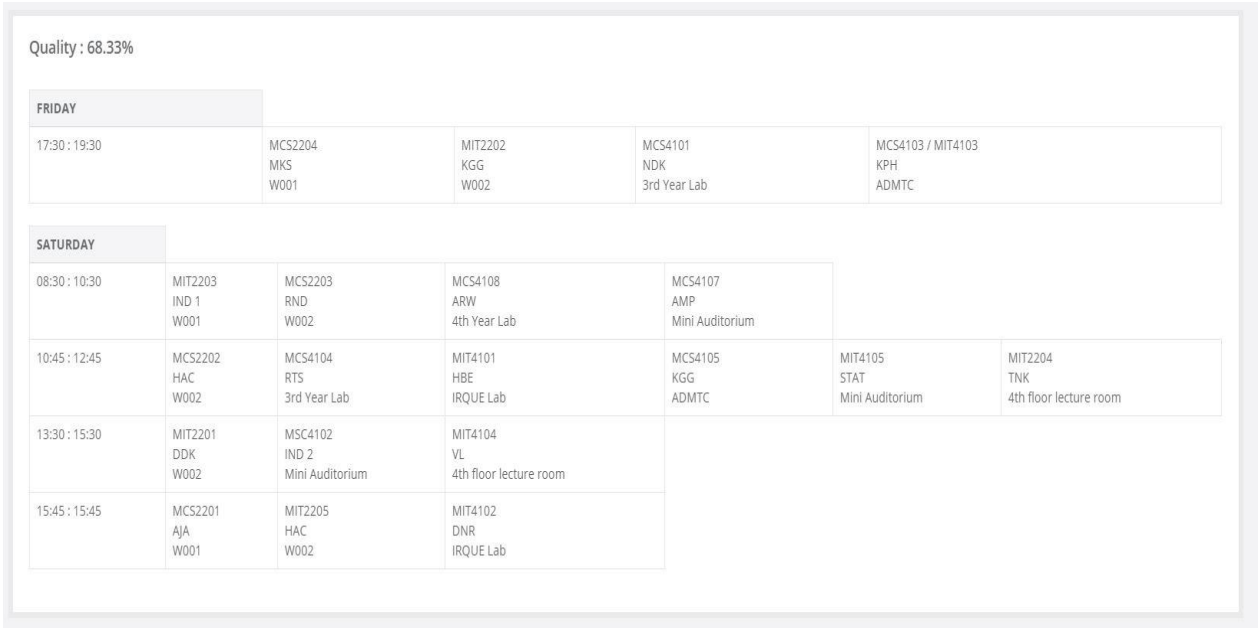
Figure 5.2: System performance with 1000 population count

Generation	Best fitness	Average fitness
0	0.92	0.833399998
1	0.92	0.863510002
2	0.93	0.880579998
3	0.93	0.892599991
4	0.93	0.900999991
5	0.93	0.907560001
6	0.93	0.914780022
7	0.93	0.919550017
8	0.93	0.927550009
9	0.93	0.930000007
10	0.94	0.930060007
11	0.94	0.930250007
12	0.94	0.930540007
13	0.94	0.931160006
14	0.94	0.932450005
15	0.94	0.935090002
16	0.94	0.939999998
17	0.97	0.940329998
18	0.97	0.940569998
19	0.97	0.941079999
20	0.97	0.94229
21	0.97	0.944780002
22	0.97	0.948870007
23	0.97	0.957690015
24	0.97	0.970000029
25	0.97	0.970000029
26	0.98	0.970590028
27	0.98	0.971160028
28	1	0.974290025

Table 5.2: Fitness details with 1000 population count

5.4 Test results

Figure 5.3 below depicts the best solution provided by the system after 10 runs. All the results received during 10 runs are included in Appendix A.



Quality : 68.33%

FRIDAY					
17:30 : 19:30	MCS2204 MKS W001	MIT2202 KGG W002	MCS4101 NDK 3rd Year Lab	MCS4103 / MIT4103 KPH ADMTC	

SATURDAY						
08:30 : 10:30	MIT2203 IND 1 W001	MCS2203 RND W002	MCS4108 ARW 4th Year Lab	MCS4107 AMP Mini Auditorium		
10:45 : 12:45	MCS2202 HAC W002	MCS4104 RTS 3rd Year Lab	MIT4101 HBE IRQUE Lab	MCS4105 KGG ADMTC	MIT4105 STAT Mini Auditorium	MIT2204 TNK 4th floor lecture room
13:30 : 15:30	MIT2201 DDK W002	MSC4102 IND 2 Mini Auditorium	MIT4104 VL 4th floor lecture room			
15:45 : 15:45	MCS2201 AJA W001	MIT2205 HAC W002	MIT4102 DNR IRQUE Lab			

Figure 5.3: Best solution generated by system

Chapter 6: Conclusion and Further Work

6.1 Introduction

This chapter discuss summary of the work has been done during this project, its limitations and future work.

6.2 Conclusion

This project is aimed at studying the impact of applying soft constraints on generation of timetables with the use of data of UCSC postgraduate courses. In order to achieve this aim a system was generated using genetic algorithm, java and a postgres database. Genetic algorithm was selected after a performing a widespread research study. During the research, algorithms such as graph coloring, integer liner programming, tabu search, particle swarm optimization, simulated annealing and genetic algorithm were studied, and genetic algorithm was selected to implement a timetable generation system due its evolving and global optimization nature.

When developing the system it was hard to incorporate soft constraints into fitness function due to score changes. Therefore soft constraints were applied after generating a feasible solution where no hard constraints are violated. Due to this methodology of soft constraints application to the system there is no impact of satisfying soft constraints on performance of the system, and the executions solely depends on hard constraints.

During the testing and evaluation phase, it was found that the size of initial population can make a great impact on the performance of genetic algorithm. Higher the initial population size is generations to be evolved to find a solution becomes fewer. And also higher amount of initial population can prevent premature convergence into some extend.

6.3 Further work

The system can be further improved to generate timetables for undergraduate courses as well. Moreover introducing more soft and hard constraints will also improve the system.

Another area that may be considered on improving is the actual algorithm. The current algorithm employed in the system is basically a Simple Genetic Algorithm (SGA) that consists of basic operations. This algorithm however can be more uniformed by transforming it into a Hybrid Genetic Algorithm (HGA). In order to implement such HGAs, mechanics such as 'Branch and Bound' can be used on the SGA. This improvement was suggested because given a difficult dataset; the system may consume a lot of time to generate a feasible solution

References

[1] T. Cooper, J. Kingston, “The complexity of timetable construction problems” in *Practice and Theory of Automated Timetabling* , E. Burke and P. Ross, Eds. Springer Berlin Heidelberg, 1996, pp. 281–295.

[2] S.A. MirHassani, F. Habibi, 2013. “Solution approaches to the course timetabling problem” *Artificial Intelligence Review*, vol. 39, No. 2, pp.133-49

[3] FET - Free Timetabling Software. [Online] Available: http://www.lalescu.ro/liviu/fet/

[4] K. Murray, T. Muller, “Automated System for University Timetabling” In *6th international conference on the Practice and Theory of Automated Timetabling*, 2006.

[5] H. Rudová,, T. Müller, K. Murray, 2011. “Complex university course timetabling” in *Journal of Scheduling* , Vol 14, No 02, pp.187-207.

[6] D. J. A. WELSH, M. B. POWELL, 1967. “An Upper Bound for the Chromatic Number of a Graph and Its Application” in *Timetabling Problems Comp. Jrnl*, Vol 10, pp.85-86.

[7] S. BRODER, 1964 “Final Examination Scheduling” in *Comm. A.C.M.* Vol 7, pp.494-498.

[8] E.K.Burke, D.G.Elliman, R.Weare, “A University Timetabling System based on Graph Coloring and Constraint Manipulation”

[9] S. Daskalaki, T. Birbas, E. Housos, “An integer programming formulation for a case study in university timetabling,” in *European Journal of Operational Research*, vol. 153, No. 1, pp. 117–135, 2004

[10] K. Schimmelpfeng, S. Helber, “Application of a real-world university-course timetabling model solved by integer programming,” in *OR Spectrum*, vol. 29, No. 4, pp. 783–803, 2007.

[11] F. Glover, 1989, “Tabu Search” in *ORSA Journal on Computing,*Vol. 1, No 3, pp.190-206.

[12] A. Hertz, 1991, “Tabu search for large scale timetabling problems”, in *European Journal of Operational Research*, Vol. 54, pp.39-47.

[13] A. Schaerf, 1996, “Tabu Search Techniques for Large High-School Timetabling Problems”, In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Portland, 1996.

[14] J. Kennedy, R. Eberhart, “Particle Swarm Optimization”, in *Proceedings of the Fourth IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995.

[15] D. Shiau, 2011, “A Hybrid Particle Swarm Optimization for a University Course Scheduling Problem with Flexible Preferences”, in *Expert Systems with Applications*, Vol. 38, pp.235–248

[16] “Particle Swarm Optimization Based Najran University Course Timetable Scheduling” in *NNGT*

[17] D. Abramson, 1991. “Constructing school timetables using Simulated Annealing: sequential and parallel algorithms”, in *Management Science*, Vol. 37, No. 1, pp.98-113.

[18] P. Kostuch, 2005. “The University Course Timetabling problem with a Three Phase Approach”, E. Burke & M. Trick, eds, in *Practice and Theory of Automated Timetabling V*, Springer Berlin Heidelberg, pp.109-25

[19] R. Bellio, et al., 2013, “A simulated annealing approach to the curriculum-based course timetabling problem”, In *6th Multidisciplinary International Conference on Scheduling: Theory and Applications*, 2013.

[20] A. Colorni, M. Dorigo, V. Maniezzo, 1990, “Genetic algorithms - A new approach to the timetable problem”, M. Akgül, H.W. Hamacher & S. Tüfekçi, eds, in *Combinatorial Optimization*, Springer Berlin Heidelberg, pp.235-39.

[21] S. J. Naseem, S. Yang, 2009. “A Guided Search Genetic Algorithm for the University Course Timetabling Problem”, In *Multidisciplinary International Conference on Scheduling : Theory and Applications*. Dublin, 2009.

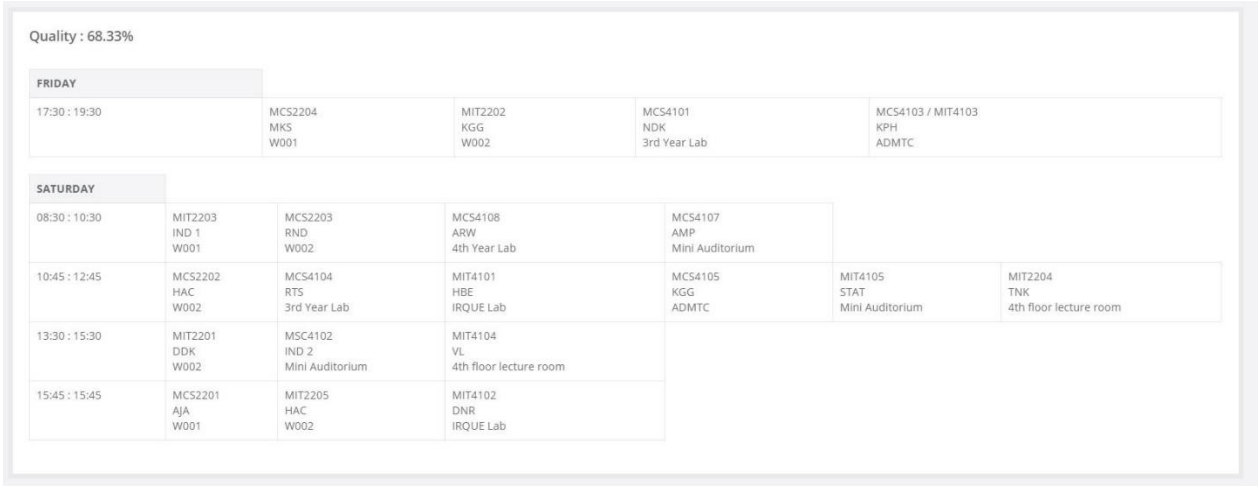
[22] E. Yu, K. Sung, 2002, “A Genetic Algorithm for a University Weekly Courses Timetabling Problem”, in *International Transactions in Operational Research*, Vol. 9, pp.703-17.

[23] Premature Convergence. [Online] Available: https://en.wikipedia.org/wiki/Premature_convergence

[24] L. Yee, et al.,1997. “Degree of Population Diversity||A Perspective on Premature Convergence in Genetic Algorithms and its Markov Chain Analysis” in *IEEE Transactions on Neural Networks*, Vol 8, No 5, pp.1165-1176

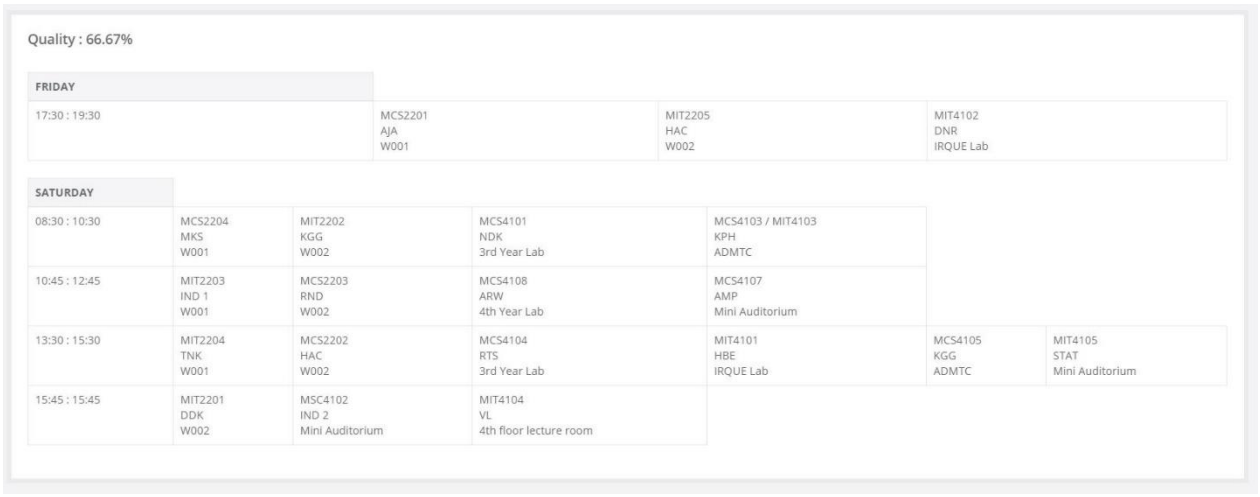
[25] E. S Nicoara, 2009, “Mechani sms to Avoid the Premature Convergence of Genetic Algorithms.”

Appendix A: Test Results



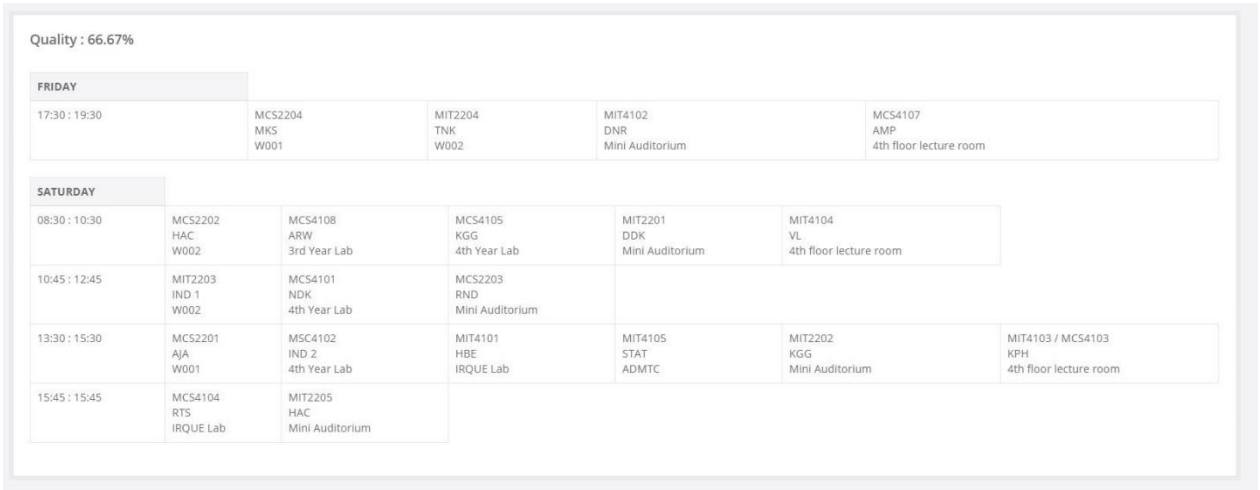
Quality : 68.33%

FRIDAY						
17:30 : 19:30	MCS2204 MKS W001	MIT2202 KGG W002	MCS4101 NDK 3rd Year Lab	MCS4103 / MIT4103 KPH ADMTC		
SATURDAY						
08:30 : 10:30	MIT2203 IND 1 W001	MCS2203 RND W002	MCS4108 ARW 4th Year Lab	MCS4107 AMP Mini Auditorium		
10:45 : 12:45	MCS2202 HAC W002	MCS4104 RTS 3rd Year Lab	MIT4101 HBE IRQUE Lab	MCS4105 KGG ADMTC	MIT4105 STAT Mini Auditorium	MIT2204 TNK 4th floor lecture room
13:30 : 15:30	MIT2201 DDK W002	MSC4102 IND 2 Mini Auditorium	MIT4104 VL 4th floor lecture room			
15:45 : 15:45	MCS2201 AJA W001	MIT2205 HAC W002	MIT4102 DNR IRQUE Lab			



Quality : 66.67%

FRIDAY						
17:30 : 19:30	MCS2201 AJA W001	MIT2205 HAC W002	MIT4102 DNR IRQUE Lab			
SATURDAY						
08:30 : 10:30	MCS2204 MKS W001	MIT2202 KGG W002	MCS4101 NDK 3rd Year Lab	MCS4103 / MIT4103 KPH ADMTC		
10:45 : 12:45	MIT2203 IND 1 W001	MCS2203 RND W002	MCS4108 ARW 4th Year Lab	MCS4107 AMP Mini Auditorium		
13:30 : 15:30	MIT2204 TNK W001	MCS2202 HAC W002	MCS4104 RTS 3rd Year Lab	MIT4101 HBE IRQUE Lab	MCS4105 KGG ADMTC	MIT4105 STAT Mini Auditorium
15:45 : 15:45	MIT2201 DDK W002	MSC4102 IND 2 Mini Auditorium	MIT4104 VL 4th floor lecture room			



Quality : 66.67%

FRIDAY						
17:30 : 19:30	MCS2204 MKS W001	MIT2204 TNK W002	MIT4102 DNR Mini Auditorium	MCS4107 AMP 4th floor lecture room		
SATURDAY						
08:30 : 10:30	MCS2202 HAC W002	MCS4108 ARW 3rd Year Lab	MCS4105 KGG 4th Year Lab	MIT2201 DDK Mini Auditorium	MIT4104 VL 4th floor lecture room	
10:45 : 12:45	MIT2203 IND 1 W002	MCS4101 NDK 4th Year Lab	MCS2203 RND Mini Auditorium			
13:30 : 15:30	MCS2201 AJA W001	MSC4102 IND 2 4th Year Lab	MIT4101 HBE IRQUE Lab	MIT4105 STAT ADMTC	MIT2202 KGG Mini Auditorium	MIT4103 / MCS4103 KPH 4th floor lecture room
15:45 : 15:45	MCS4104 RTS IRQUE Lab	MIT2205 HAC Mini Auditorium				

Quality : 66.67%

FRIDAY					
17:30 : 19:30	MIT2201 DDK W002	MIT4103 / MCS4103 KPH 3rd Year Lab	MCS4101 NDK 4th Year Lab	MCS2203 RND Mini Auditorium	

SATURDAY					
08:30 : 10:30	MCS2201 AJA W001	MIT2203 IND 1 W002	MIT4101 HBE ADMTC	MSC4102 IND 2 Mini Auditorium	
10:45 : 12:45	MIT2202 KGG W001	MIT4102 DNR W002	MCS2202 HAC Mini Auditorium		
13:30 : 15:30	MCS4104 RTS W001	MIT2205 HAC W002	MCS4105 KGG 3rd Year Lab	MIT4104 VL 4th Year Lab	
15:45 : 15:45	MIT2204 TNK W001	MCS2204 MKS W002	MCS4107 AMP 3rd Year Lab	MCS4108 ARW IRQUE Lab	MIT4105 STAT 4th floor lecture room

Quality : 66.67%

FRIDAY			
17:30 : 19:30	MCS4105 KGG W001	MCS4104 RTS IRQUE Lab	MIT2205 HAC Mini Auditorium

SATURDAY						
08:30 : 10:30	MCS2204 MKS W001	MIT2204 TNK W002	MCS4108 ARW 3rd Year Lab	MSC4102 IND 2 4th Year Lab	MIT4102 DNR Mini Auditorium	MCS4107 AMP 4th floor lecture room
10:45 : 12:45	MCS2202 HAC W002	MIT2201 DDK Mini Auditorium	MIT4104 VL 4th floor lecture room			
13:30 : 15:30	MIT2203 IND 1 W002	MCS4101 NDK 4th Year Lab	MCS2203 RND Mini Auditorium			
15:45 : 15:45	MCS2201 AJA W001	MIT4101 HBE IRQUE Lab	MIT4105 STAT ADMTC	MIT2202 KGG Mini Auditorium	MIT4103 / MCS4103 KPH 4th floor lecture room	

Quality : 66.67%

FRIDAY				
17:30 : 19:30	MCS4108 ARW W001	MIT2201 DDK W002	MCS4105 KGG 4th Year Lab	MCS2202 HAC Mini Auditorium

SATURDAY				
08:30 : 10:30	MIT2204 TNK W001	MCS2203 RND W002	MCS4107 AMP 3rd Year Lab	MIT4102 DNR 4th Year Lab
10:45 : 12:45	MCS2204 MKS W001	MIT2203 IND 1 W002	MIT4104 VL 3rd Year Lab	MSC4102 IND 2 4th Year Lab
13:30 : 15:30	MIT4103 / MCS4103 KPH W001	MIT4105 STAT W002	MCS4104 RTS 4th Year Lab	MIT2202 KGG Mini Auditorium
15:45 : 15:45	MCS2201 AJA W001	MIT2205 HAC W002	MCS4101 NDK Mini Auditorium	MIT4101 HBE 4th floor lecture room

Quality : 66.67%

FRIDAY					
17:30 : 19:30	MIT4103 / MCS4103 KPH W001		MIT4105 STAT W002	MCS4104 RTS 4th Year Lab	MIT2202 KGG Mini Auditorium

SATURDAY				
08:30 : 10:30	MCS2201 AJA W001	MIT2205 HAC W002	MCS4101 NDK Mini Auditorium	MIT4101 HBE 4th floor lecture room
10:45 : 12:45	MCS4108 ARW W001	MIT2201 DDK W002	MCS4105 KGG 4th Year Lab	MCS2202 HAC Mini Auditorium
13:30 : 15:30	MIT2204 TNK W001	MCS2203 RND W002	MIT4102 DNR 4th Year Lab	MCS4107 AMP IRQUE Lab
15:45 : 15:45	MCS2204 MKS W001	MIT2203 IND 1 W002	MIT4104 VL 3rd Year Lab	MSC4102 IND 2 4th Year Lab

Quality : 66.67%

FRIDAY					
17:30 : 19:30	MCS4108 ARW W001	MIT2201 DDK W002	MCS4105 KGG 4th Year Lab	MCS2202 HAC Mini Auditorium	

SATURDAY				
08:30 : 10:30	MIT2204 TNK W001	MCS2203 RND W002	MIT4102 DNR 4th Year Lab	MCS4107 AMP IRQUE Lab
10:45 : 12:45	MCS2204 MKS W001	MIT2203 IND 1 W002	MIT4104 VL 3rd Year Lab	MSC4102 IND 2 4th Year Lab
13:30 : 15:30	MIT4103 / MCS4103 KPH W001	MIT4105 STAT W002	MIT2202 KGG Mini Auditorium	MCS4104 RTS IRQUE Lab
15:45 : 15:45	MCS2201 AJA W001	MIT2205 HAC W002	MCS4101 NDK Mini Auditorium	MIT4101 HBE 4th floor lecture room

Quality : 66.67%

FRIDAY						
17:30 : 19:30	MIT2201 DDK W001	MCS2202 HAC W002	MIT4104 VL 3rd Year Lab	MCS4101 NDK 4th Year Lab	MCS4107 AMP IRQUE Lab	

SATURDAY					
08:30 : 10:30	MIT2202 KGG W002	MSC4102 IND 2 3rd Year Lab	MIT4105 STAT ADMTC		
10:45 : 12:45	MCS2203 RND W001	MCS4108 ARW 3rd Year Lab	MIT4101 HBE 4th Year Lab	MIT2204 TNK Mini Auditorium	
13:30 : 15:30	MCS2201 AJA W002	MCS4104 RTS 3rd Year Lab	MIT2203 IND 1 Mini Auditorium		
15:45 : 15:45	MCS2204 MKS W001	MIT2205 HAC W002	MCS4105 KGG 3rd Year Lab	MIT4102 DNR 4th Year Lab	MIT4103 / MCS4103 KPH 4th floor lecture room

Quality : 66.67%

FRIDAY					
17:30 : 19:30			MIT2202 KGG W002		MIT4105 STAT ADMTC

SATURDAY					
08:30 : 10:30	MCS2203 RND W001	MCS4108 ARW 3rd Year Lab	MIT4101 HBE 4th Year Lab	MCS4107 AMP IRQUE Lab	MIT2204 TNK Mini Auditorium
10:45 : 12:45	MCS2201 AJA W002	MCS4104 RTS 3rd Year Lab	MIT2203 IND 1 Mini Auditorium	MCS4105 KGG 4th floor lecture room	
13:30 : 15:30	MCS2204 MKS W001	MIT2205 HAC W002	MIT4102 DNR 4th Year Lab	MSC4102 IND 2 Mini Auditorium	MIT4103 / MCS4103 KPH 4th floor lecture room
15:45 : 15:45	MIT2201 DDK W001	MCS2202 HAC W002	MCS4101 NDK 4th Year Lab	MIT4104 VL 4th floor lecture room	