The University Of Sheffield.

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF ENGINEERING
UNIVERSITY OF SHEFFIELD
COM6906 – DISSERTATION PROJECT

Constructive and Unconstructive Initialisation of a Genetic Algorithm For Solving a University Timetabling Problem

Peter Uchenna Eze
(Registration Number: 140127603)

Supervisor: **Dr Dawn C. Walker**

A report submitted in partial fulfilment of the requirements
For the degree of **Masters of Science in Engineering**
In **Advanced Software Engineering**

September 09, 2015

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been specifically acknowledged. I understand that failure to do these amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: **Peter Uchenna Eze.**

Signature: _____

Date: **9th September, 2015**

Acknowledgement

My greatest gratitude goes to almighty God for his infinite mercy and divine providence.

The timely and constructive feedback from my supervisor, Dr. Dawn C. Walker, cannot be overemphasised in the successful completion of this dissertation. I greatly appreciate such dedicated supervision. It was clear and directional.

I will not fail to appreciate the Federal Republic of Nigeria, in general, and Federal University of Technology Owerri Nigeria, in particular, for their dedication in training their future academics. I am indebted to Prof. C.C Asiabaka, the Vice Chancellor of FUT Owerri for his vision and confidence in future leaders and academics of the University.

My further gratitude goes to all the lecturers in department of Computer Science at the University of Sheffield for their awesome teaching methodology and support. Special kudos to Prof. Guy Brown and Dr. Kirill Bogdanov.

The gift of my parents and siblings is a great propelling force to my success in life. I appreciate their support in the past and present studies. Special thanks to Mr. & Mrs Eze Sebastian, Miss Ezeh Veronica, Mrs. Ezeh Grace and Mr. Eze Anthony.

I will not fail to thank my friends and well-wishers who kept me company both here in Sheffield and remotely in the course of my studies and during this dissertation project. Special thanks to Engr. Uche Diala, Miss Ugwuja Linda, Mr. Ndukwe Cherechi, Miss Emekekwe Morning-Glory, among others.

Finally, I will also thank myself for resilience and understanding of the challenges ahead.

Abstract

The University timetabling Problem (UTP) is one of the scheduling problems ridden with numerous constraints. Each of these constraints has complex effect on the ideal solution and thus the combined effect make the problem hard to solve. This is a case for the inadequacies with the current electronic timetabling system in the Department of Computer Science, University of Sheffield. Thus, it is required that an automated electronic system that could resolve the complex constraints while scheduling course modules be designed and developed.

In order to solve this problem and also develop insight into other timetabling problems, the method of genetic algorithm (GA) was developed and adapted in the forms of constructive and unconstructive initialisation methods. Eight hard constraints and four soft constraints existing in the Department of Computer Science were used in developing the GA. Software was designed and experiments run using 29 lecturers, 33 modules from autumn semester for all taught programmes, 6 laboratory rooms and 19 lecture rooms. The resources and constraints were divided into five levels of difficulty to test the developed algorithm.

The result of the research shows that the constructive method can satisfy all the hard constraints, achieve up to 75% optimisation of the soft constraints and converge within 500 iterations or average of 2.74 minutes. The unconstructive method showed good evolutionary tendencies but did not produce a feasible solution within reasonable time. Further results show that the length of time taken to run the GA depends on if the fitness function used is reward-based or penalty-based and on how good the genes in the chromosomes are. In conclusion, constructively initialised GA is efficient in solving specific UTP while the unconstructively initialised GA could not but could be used to get insight into different timetabling problems before a more constructive algorithm can be designed for the specific problem instance.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

The University timetabling problem (UTP) is one of the scheduling problems, which involves the combinatorial allocation of resources under some predefined constraints [1]. This allocation is done within time slots (usually working hours) with the aim of maximising the use of available space and time as well as avoiding the violation of some constraints. These constraints may include the number of available lecture rooms, types of lecture rooms, sizes of lecture rooms, the number of lecturers and their availability, lecturer specialty, number of courses to be delivered, the number of students in a course, distance between lecture rooms, among others. Some of the constraints are **hard constraints** that must be satisfied to make the solution valid while others are **soft constraints** which are only desirable to be satisfied. Optimisation involves trying to satisfy as many of the soft constraints as possible. The departmental timetables are the basic functional units of any UTP. Every module could be traced to lecturers in various departments.

In this dissertation, the Genetic Algorithm (GA) concept will be used to develop a solution to the timetabling problem generally described in Section 1.1 but specifically defined in Chapter 3. GA works by selecting only a portion of the large search space and evolving it into the desired solution by applying some genetic operators to the initial population. However, the major challenge will be finding a suitable problem representation for the specific problem case to be defined (Chapter 3), determining the various constraints applicable to the problem, defining a suitable fitness function, determining how to apply GA to the problem and developing a software that utilises the algorithm to solve practical timetabling problems.

1.1 Aims and Objectives

The aim of this research is to design a suitable genetic representation of typical timetabling problem and apply it to solve a specific timetabling problem in the Department of Computer Science. This aim will be achieved with the following objectives in mind:

1. To determine the best genetic representation for the timetabling problem.
2. To define appropriate Hard and soft constraints for the specific problem and formulate a suitable fitness function based on these constraints.
3. To code the formulation algorithmically to solve a test problem
4. To apply the designed representation to solve a subset of timetabling problem in Department of Computer Science
5. To design software Graphical User Interface (GUI) to implement the algorithm to solve the problem set.
6. To compare the performance and the prospects of constructive and unconstructive initialisation methods for a GA.

The specific problem set that will be used to evaluate these objectives will be explicitly defined in Chapter 3, section 3.2.

1.2 Organisation of Report

The rest of this report is organised thus: Chapter 2 will survey existing literature in the areas of GA and University Timetabling problems. Chapter 3 will discuss system requirements and analysis for this dissertation while chapter 4 will detail the designed solution to the problem. Chapter 5 contains the implementation and tests carried out on the software GUI. Chapter 6 will present and discuss the results from the experiments while chapter 7 will evaluate the product and the entire dissertation work. Chapter 8 concludes the report and also suggests areas for further research.

Chapter 2: Literature Review

This chapter will review the fundamental background and theories of natural evolution and how they are adapted in Genetic Algorithm (GA) to solve human problems. Thus, GA concepts and terminologies will be discussed. It will also introduce the concepts of constructive and unconstructive initialisation methods for a GA. This chapter will also discuss the problem at hand and how other researchers have implemented GA to represent and solve timetabling problems and then set stage for this project work.

2.1 Natural and Artificial evolution

The ability of biological systems to adapt to varying environmental conditions as well as evolve over time is a good inspiration for designing both hardware and software systems that can also evolve over time[3]. A general understanding of natural or biological evolution will help one to have a better understanding of artificial evolutionary algorithms such as the GA.

Charles Darwin in [8] explained the theories of natural evolution. His work tallied with an independent work by A.R Wallace as described in [9]. Hence according to these works, species that have the fitness to survive in their environment are selected to exist in subsequent generations. Those with unfavourable attributes go into extinction. Hence, as generations of an organism are created, the offspring tend to change by random processes of mutation and exchange of genetic materials from parents. These random processes tend to make some individuals to adapt better to their environment and others to less adapt. The phrase "*survival of the fittest*" has been used by Hubert Spencer [5] to describe natural evolution. However, we are not interested in in-depth knowledge of natural evolution but on how its principles can be adapted and used in artificial evolutionary algorithms.

Artificial evolution is intended to solve a specific problem unlike natural evolution which does not have a pre-defined goal and solves open-ended, non-directional and unpredictable problems. Artificial evolution is an optimisation process that starts with poor solutions and applies the elements and theories (as will be explained in section 2.2) of natural evolution to solve a predefined problem. As stated in [3], artificial evolutionary techniques are most appropriate in solving problems where conventional optimisation techniques have failed to yield a good solution. They specifically mentioned situations where the functions to be optimised are discontinuous, non-differentiable or has many nonlinearly related parameters.

Today, artificial evolution is known as Evolutionary Computing (EC) and dates back to 1940s when Alan Turing, the father of modern computing first conceptualised the idea of optimising solutions to problems by natural evolution concepts [10]. The collection of algorithms used in Evolutionary computing are called Evolutionary Algorithms (EA). Both [5] and [10] agree that there are four major areas of EC: Genetic Algorithms (GA), Genetic Programming, Evolutionary Programming and Evolutionary Strategies. In this work, we will be applying GA to solving University timetabling and thus it will be discussed further. For further readings on the three other aspects of Evolutionary computing see [3], [5] and [10].

2.2 Terminologies and Concepts

This section explains basic terminologies and concepts that originated from natural evolution but are used in evolutionary computing and thus are applicable to GAs.

2.2.1 Genes - Genes are used to code the fundamental information that determines the characteristics of an organism. It is the smallest information-bearing element in an organism. It is made up of a sequence of nucleotides (coding elements) at a specific position in a chromosome. According to Mendel in 1865, offspring are reproduced by the transmission of these genetic material called genes by their

parents [3]. The structure of a gene is shown in figure 2.1. It contains a coded region (exons), non-coded region (introns) and a regulatory region, which determines where, when and how much protein is made [13].

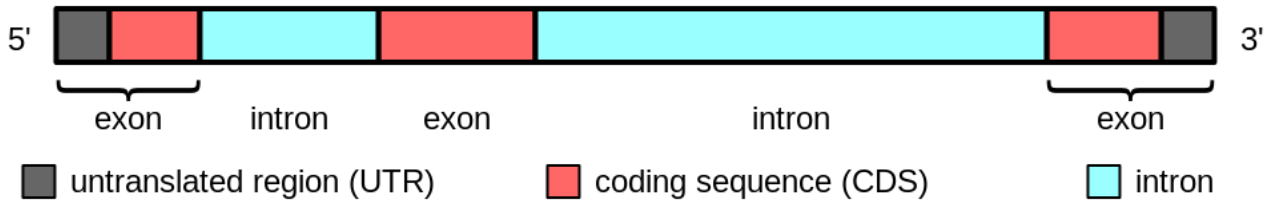


Fig 2.1 - Structure of a gene [17]

2.2.2 **Genotype** - This is the set of genes carried by an individual. It determines the sets of characteristics exhibited by this individual. According to [12], genotype is the genetic material which is inheritable and relates to a given phenotype in an individual. This assertion was supported by [3], which postulates that genotype is considered the blueprint of an organism as it contains the set of coded instructions that are used to create a unique organism. The alteration of the genotype or the genetic content of an individual (by means of genetic operators as described in section 2.3.2) will alter the physical and behavioural appearance of an organism.

2.2.3 **Phenotype** - The meaning of phenotype and how it relates to genotype is better illustrated by the diagram in Figure 2.2. Phenotype is the observable manifestation of a genotype in an individual. Phenotype is derived from genotype [4, 12, 14]. The manifestation in genetic changes in form of phenotype can be related to variation in eye colour, ability of an individual to survive in a given environment, the transmission of a genetic disease or not, among others [14]. In figure 2.2, the genotype 1111 is a representation for an oval shape of BLUE, while 1100 represents PURPLE and so on. Hence, it is obvious that a manipulation of the binary bits (genes) would lead to variation in the colour of the oval figure. This gives one a basic understanding of and relationship between genotype and phenotype.

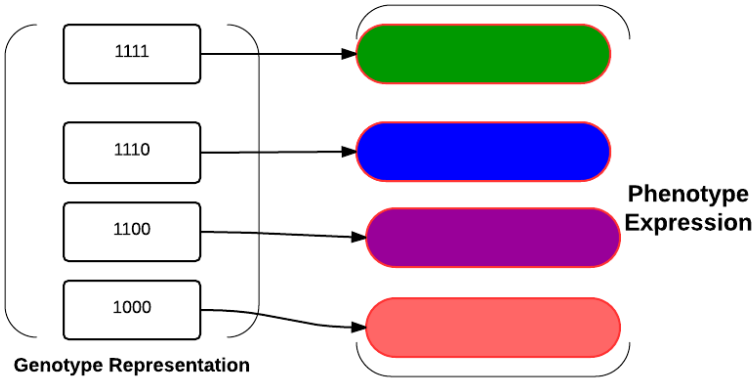


Fig 2.2 – Genotype-to-Phenotype Mapping

Thus in GA, problems are represented in suitable genotypic terms and operated on at that level. However, there must be a suitable mapping from the genotypic domain to the phenotypic domain.

2.2.4 Chromosomes - A chromosome is an ordered collection of genes that form part of the entire genetic content of an organism. Humans have between 20 to 25 thousand genes distributed across the 23 chromosomes pairs [13]. The length of a chromosome is related to the number of genes it contains.

2.2.5 Population - Population is a collection of two or more individuals. The actual number of individuals that will take part in reproduction to create generations is known as the *population size (N)*. The choice of N for a GA problem is critical. Increasing population size, N, increases the accuracy of GA. However, this increases the number of generations for the GA to converge. Hence, one has to strike a balance between these constraints. The work done by the authors in [20] is an attempt to determine the optimum size of N for GAs as a function of the search space of the problem instance. However, no hard and fast rule could be deduced from their work. Both [3] and [5] are of the opinion that individuals in a population must have diverse characteristics which can be transmitted from one generation to another through reproduction. This is very important in implementation of GAs because not all possible individuals (*candidate solutions*) will be part of the initial population. If the initial populations are similar and they are not solutions to the problem, then it is less likely that a solution will be produced by applying genetic operators. This is because the offspring will have almost the same features as the parents. Thus diversity and size are major characteristics of the initial population in GAs [5, 12].

2.2.6 Generation - Generation can be seen as the set of individuals living about at the same time. However, in artificial evolution such as GAs, the set of offspring produced before a *feasible solution* is found is the number of generations for the *run*. The concepts of Reproduction and Generations are illustrated in Figure 2.3. A new generation is represented by "New population" after reproduction among the parents in the "Current population".

2.2.7 Reproduction - In biology, reproduction is the process of producing offspring. This is done by crossing over the genes from two parents in order to produce a new individual of some characteristics that are both similar and dissimilar to those of their parents. These offspring grow up to mate and reproduce as well. For natural evolution, it continues till extinction.

In natural evolution it can be either sexual or asexual. In sexual reproduction two parents are involved while in asexual reproduction only one parent is involved. The important thing is that reproduction leads to the evolution of a new offspring with some peculiar characteristics than the parent(s).

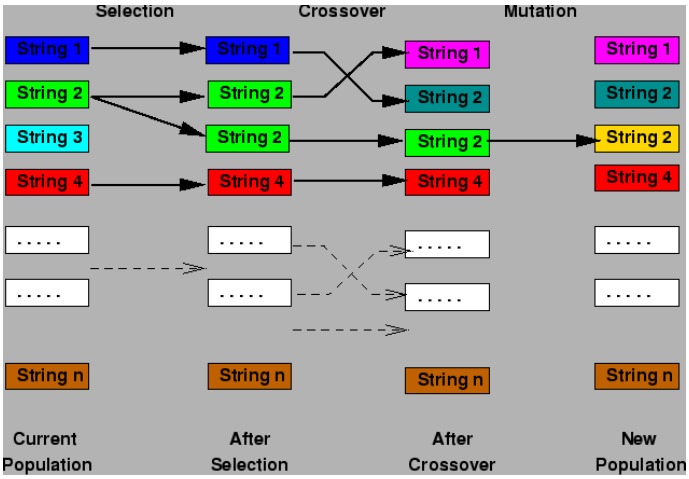


Figure 2.3 – Genetic operators for Reproduction [18]

In biological evolution, the reproduction process involves the exchange of the parts of the father's chromosome with part of the mother's chromosome. This is known as *crossover*. After this, random *mutation* may occur in which any chosen gene locus may have its genetic code transformed to any alternative. Both crossover and mutation are random. Their occurrence may increase or decrease the chance of survival of the offspring in the current environment.

The concepts of selection, crossover and mutation in the context of artificial evolution are explained in section 2.3.2 as genetic operators.

2.3 Theories of Genetic Algorithm

Genetic Algorithm (GA) is a particular class of evolutionary algorithms. Like many other artificial evolution-based algorithms, GA takes its inspiration from the concepts of natural evolution. These concepts include maintenance of population, creation of diversity, a selection mechanism and a process for transfer of genetic characteristics [3]. Evolutionary algorithms do not replicate biological evolution but only borrow its stochastic nature to solve hard problems. This suggests that any chosen evolutionary algorithm must be adapted in some ways in order to solve the problem concerned. This research is not an exception.

No special definition differentiates GA from other evolutionary computation algorithms [4]. This statement was corroborated by [3] who provided generic steps for constructing custom-made evolutionary algorithm of any type. Hence, we can define GA by listing the components of most methods that have been described as GA. According to both [3] and [4] a GA consists of at least the following: A genetic representation of the problem, a population of chromosomes, appropriate fitness function and genetic operators for evolution. The diagrammatic representation of a typical GA is shown in fig 2.4.

For termination criteria (in fig 2.4 above), two conditions are often used: either a solution has been found or a number of generations have been reached [6]. Oftentimes GAs can run for long without finding the best solution possible. General feasible solutions could be acceptable depending on the problem in question.

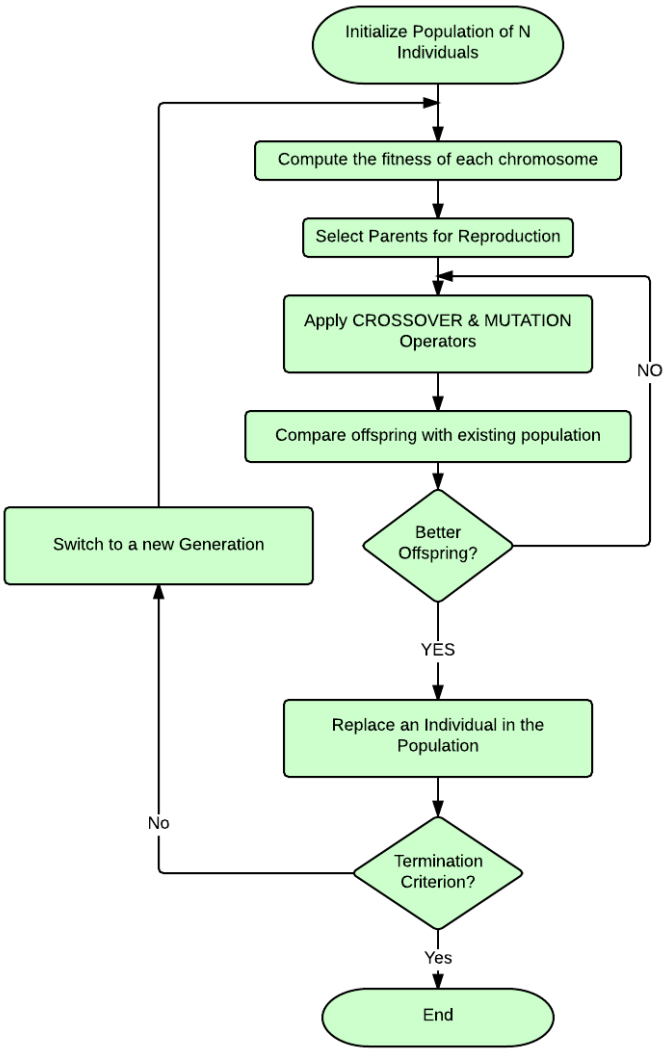


Fig 2.4 – The Genetic Algorithm

The rest of this subsection will be dedicated to explaining more about those components of GA between its start and stop, and on problem representation and solution evolution.

Constructive and Non-Constructive Initialisation

A general purpose GA as described in figure 2.4 above may never efficiently solve a real world problem. David Coley of the University of Exeter in his book in [40] used a graph to describe the efficiency and robustness of various class of algorithms used to solve problems with large search space. The graph is shown in Figure 2.5 below. The parts labelled ***'adapted GA'*** and ***'general purpose GA'*** are of interest to this research though.

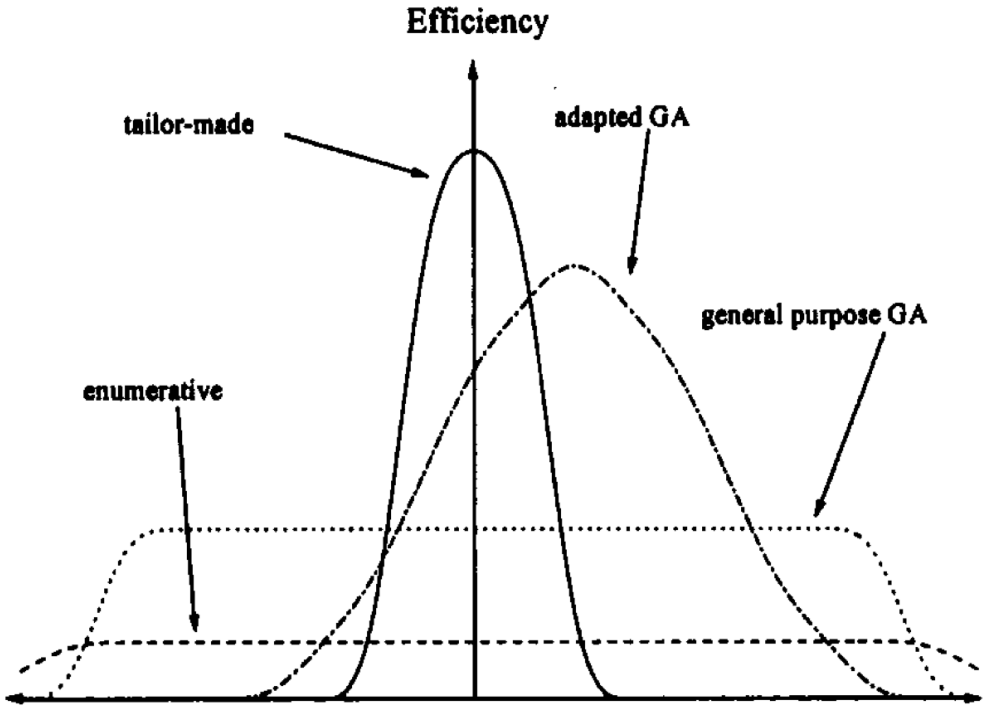


Figure 2.5 - Comparison of the Efficiency of Adapted and General Purpose GA [40]

There is a wide range of gap in efficiency between adapted GA and general purpose GA. He also quickly added that one of the ways to improve the traditional GA is to locate approximate solutions to the GA and then start running the traditional GA from such points. This is the major idea behind constructive initialisation. On the other hand, the case in which the initial population is totally chosen at random and the execution of the GA uses this entirely random population is called unconstructive initialisation GA.

The diagram above suggests that adapted GA is efficient in solving more specific and well defined problems while unconstructive methods might be less efficient but solves a wider range of problems. This research will investigate these in the context of University timetabling problem by trying to solve the defined problem in chapter 3 using the constructive method and identifying the wider range of problems that could be solved using the unconstructive method.

2.3.1 Genetic representation of Problems

The nature of problem to be solved determines how they are actually represented. Complexity and nature of manipulation needed to evolve the solution also needs to be put into consideration when deciding on the genetic representation of the problem. However, one general principle is that the chosen genetic representation should adequately map onto the observable features of the solution, which is the phenotype. Also, according to [3], the chosen genotypic representation normally depends on the problem domain considered.

This subsection discusses various ways of representing problems that are amenable to the application of GA. More general approach will be considered here while section 2.5 will focus more on how researchers have represented the timetabling problem for application of GA. The various ways of representing GA then follows.

- **Discrete Representations** – This method uses discrete-valued coding system made up of *m* discrete values made up alphabets of cardinality, k. This implies that for each of the *m, k* number of values can be used [3]. The simplest of the discrete representation methods is the binary representation method. In Figure 2.2 above, the binary representation was used. Binary has a cardinality of 2. In Fig 2.2, m=4 and k=2. Hence, a total of 2^4 (k^m) individual colours can be represented by this genetic representation method. As the search space increases, the use of binary representation also becomes inefficient as mapping to phenotype become more difficult and computational time increases. The advantage of binary representation is that it easily maps to several different phenotypes [3]. Other discrete representation method use string combinations with cardinality greater than 2.

- **Integer Programming** – This is related to discrete representation. However, the cardinality is larger. A set of integer combinations are used to represent combination of resources that are needed in order solve a problem. This can be used for more complex GAs, where direct binary mapping is inadequate. For instance, a lecturer -student meeting event might be represented as T1-L2-C5-R7 in the genotypic domain. This could map to a lecture holding at period 1 (T1) where Lecturer 2 (L2) named John Smith is to teach course 5 (C5) coded COM6516 at Lecture room 7.

- **Floating Point representation** – floating point representation of genetic chromosomes was used in [19] and compared to the use of binary presentation for maximizing a function. They concluded that floating point representation could be efficient and gives better precision than binary representation of problems. Floating Point representation can be compared to fuzzy logic as binary representation compares to Boolean logic.

- **Sequence Representation - This** is a variance of discrete representation method that represents a problem as a series of points having certain relationship. It is often used in the Traveling Salesman Problem [3].

- **Real-valued -** this is related to floating-point representation method. Here, the genotype is a set of real numbers that represent some parameters (phenotype). It is used high precision is required [3, 19] such as in safety-critical system like airplane wing profile optimisation [3].

- **Tree-based -** This used more often in Genetic Programming (GP) using languages such as lisp, which has a tree-based coding format. Such programs have branching points and terminal points that encode hierarchical structures. This enables evolution of programs where a branch can be cut off and joined by another branch during the crossover process [3].It should be noted that no method of representation is the best in all situations. Each is better suited to a particular problem domain. One should have the ability to discern what representation method and probable modifications are needed. However, any chosen method must be amenable to the application of the genetic operators, capable of evolving into a better solution for the problem considered and must be evaluated against a suitable fitness function.

2.3.2 Genetic Operators

In artificial evolution, two parents are often involved. The methods of selection, crossover and mutation can be used to reproduce offspring that will be part of the next generation. These three elements are known as genetic operators and are described in this sub-section.

2.3.2.1 Selection

In GA execution, individuals with good qualities need to be made part of the initial population and that of the subsequent generations. For initial population, the initial phenotypes are selected at random and evaluated against a fitness function (See section 1.3.2) to determine if it qualifies [3, 4, 5, 12]. At this stage, care should be taken in the selection process to ensure that an initial population of high diversity is achieved. In some cases some heuristic algorithms might be employed to ensure this.

Selection in GAs also refers to the method of choosing the individuals that will act as parents in order to 'mate' to produce the offspring for the next generation. This second perspective requires further explanation. Several selection methods exist depending on the type of problem. They are briefly explained below:

- **Tournament Selection** - This method involves picking two individuals from a population and staging a contest to determine which one will be selected as a parent. According to [16] and supported by [12], this contest involves having a pre-determined selection probability and generating a random number between 0 and 1 and comparing it with this pre-determined value. The choice is then made thus:

  if random number <= pre-determined value
  select the fitter candidate
  else
  select weaker candidate
  end

  To favour fitter candidate, the pre-determined value is set greater than 0.5.

- **Fitness-Proportionate selection** - this is a selection method that gives every individual the opportunity to be selected but as a function of its fitness for survival. In GA, the fitness is always related to the candidate's progress towards being a feasible solution. The most common variations of the Fitness-Proportionate selection are Roulette wheel selection and Stochastic Universal Sampling [16].

- **Rank-based Selection** - The fitness of the individuals are assigned relative to one another instead of being calculated absolutely from a fitness function. This strategy is good when it is very necessary to avoid premature convergence of the GA.

- **Truncated Rank-based Selection** - Here a given number of the top fittest individuals in the population are selected and replicated to form the next generation [3, 12, 16]. For instance in a population of 100, the top 25 individuals can be selected and each replicated 4 times to make up the next generation of parents. This strategy has been described as simple but least useful except in situations where it can be proved as the best option [16].

- **Elitism** - In most cases, the application of genetic operators such as crossover and mutation operators may lead to the loss of good candidates. Elitism is the practice in which some percentage of candidates with good fitness are copied as-is to the next generation without applying a genetic operator to them [16]. This concept is analogous to making a parent to become an offspring in the next generation. This is a concept that is really peculiar to artificial evolution due to the fact that there is a goal. Thus individuals that are making enormous progress towards the goal might need to be retained.

2.3.2.2 Crossover

This operator is used to produce new offspring by dividing and recombining the genes from two or more parents. Usually, two parents are used. Often two offspring are created but one is chosen depending on its fitness and added to the population. Fig 2.3 illustrates how two new colours were produced by crossing two parent colours. As stated in [5] the aim of crossover is to produce an offspring that resembles its parents. The intention is often to retain the good features of the parent while improving the offspring for better survival or progress towards the solution sought. Fig 2.6 below provides a simpler illustration of how crossover works.

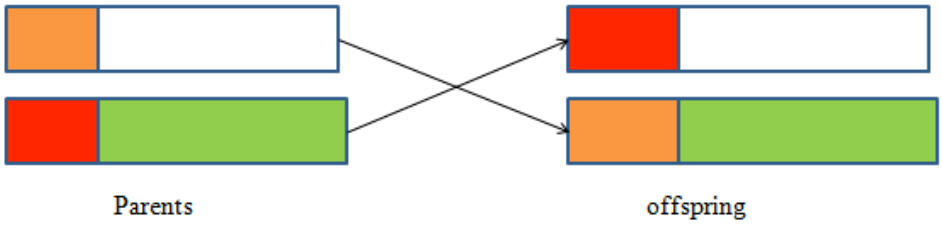


Fig 2.6 - Single Point Crossover [12]

Fig 2.5 uses a phenotypic example to enhance understanding. In reality, crossover is done at the genotypic level but the result manifests in observable changes in the phenotypic content of the offspring as related to the parent.

Crossover could be single-point, multi-point or uniform [12]. Multi-point crossover selects n points on the chromosome length where the swapping of genetic material between the parents can take place. This is illustrated in Fig 2.7.

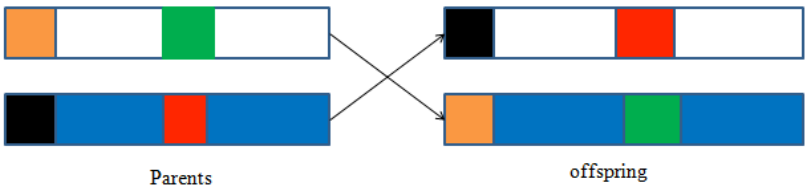


Fig 2.7 - Multi-point crossover [12]

The uniform crossover is a variant of multi-point crossover. As shown in Fig 2.8, the crossover occurs in more than one point. However, it is not at random points as is the case in both single-point and multi-point crossover. With uniform crossover, a fixed mixing ratio is used and crossover occurs at equal distance along the length of the chromosome.

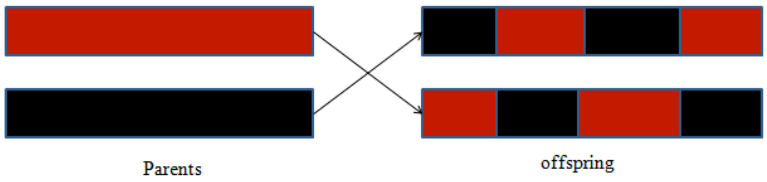


Fig 2.8 - Uniform Crossover [12]

It is evident that a single point crossover retains more of the features of the parent in the offspring. At which point the crossover occurs determines the parent whose feature is retained more in which offspring. Hence, it would help to carefully determine this point if pseudo-random crossover and elitism techniques will be used in running the GA.

However, the work done by [21] shows that it is always challenging to determine which of the crossover operators to use at a given point while running the GA in order to achieve optimum solution. This made them to propose an adaptive crossover operator known as "Selective crossover" operator.

The researchers in [36] compared three most popularly used crossover operators in solving UCTP: partially matched crossover (PMX), order crossover (OX), and cycle crossover (CX). They ran their experiments between 300 and 700 generations using each of these operators. They found out that PMX performed best among others around the 500 generation. With this in mind and using the fact that 500 is an average of 300 and 700, which will give a fair ground to accommodate both diversity and speed, PMX will be used in this research. Besides, the Sector-based PMX previously used by researchers in [28] is perfectly suited to grouping rooms into laboratory and lecture rooms.

2.3.2.3 Mutation - This operator randomly changes the genetic content of an individual at a given genetic locus. It involves only one parent or applied to an offspring after crossover. A simple diagram to illustrate mutation is shown in Fig 2.9.

Normally, gene mutation must not be a swap but a genetic locus assuming any of codes that is with the cardinality of the genetic space. This is illustrated in Fig 2.10.

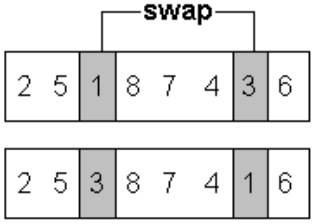


Fig 2.9 - Mutation by swapping of genes [22]

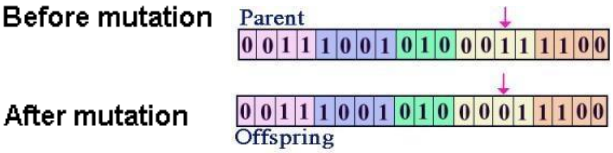


Fig 2.10 - Random Genetic Mutation [23]

In Figure 2.10, a genetic locus randomly changed from a '1' to a '0'. This genetic change will as well bring about a phenotypic change in the offspring in relation to the parent. Mutation is a random process and its probability of occurrence is always low compared to crossover.

2.3.3 Fitness Functions

Fitness function is an objective function that enables one determine which individual will be allowed to move unto the next generation and also how an individual has progressed towards the sought solution [12, 23] . As

specified in [23] it helps us to compute a unified figure of merit to summarise how close to the desired goal a solution is. This is because the fitness function for non-trivial problems is often computed after some inputs from other objective functions. Sometimes, it is an estimate of the extent to which some hard and soft constraints relating to a problem have been satisfied.

The choice for fitness function is problem-dependent and is often not an easy task. The complexity of some fitness functions can be illustrated in figure 2.11 with the chains of other functions which lead to the definition of single fitness function.

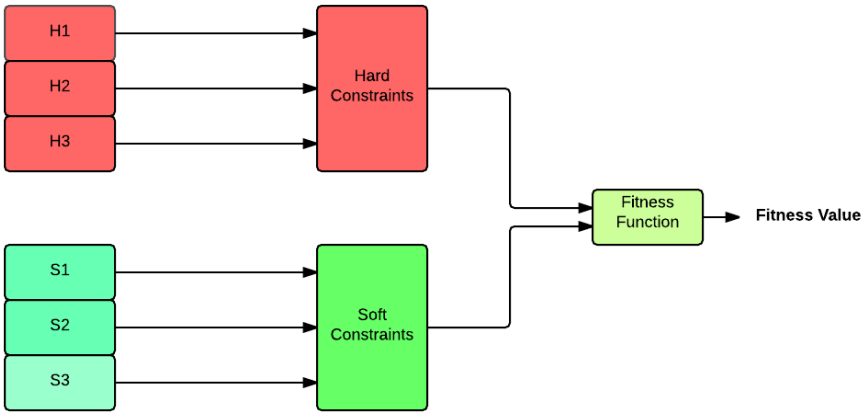


Fig 2.11 - Chains leading to fitness value

Each of the blocks could be a simple or complex objective function which needs to be minimised or maximised. However, in GA we are interested in the final fitness value. The challenge lies in getting proper function that sensibly produces a fitness value that reflects all other objective functions in the fitness chain.

2.3.4 Applications and Uses of GA

GA can be applied to solve various problems that face the human race.

The general classes of problems that could be solved by GA (and other evolutionary algorithms) have been mentioned in section 2.1. However, in this section we will briefly describe more specific problems that could be solved by the principle of GA. Some of these areas of use and application are as follows:

1. ***Train Scheduling System*** - Train timetabling problem involves setting the arrival and departure times for trains belonging to different operators, origination from different stations but utilising some set of common facilities such as the rail track and common junctions. This is an NP-hard problem involving multi-variable objective functions and lots of constraints. A real world Train timetabling problem from Spanish Manager of Railway infrastructure (ADIF) was solved by researchers in [25] using Genetic Algorithm. The problem involved determining a non-periodic Train Timetabling System to enable the allocation of paths requested by train operators without violating the rights of already existing path users.

2. ***Traveling Salesman Problem (TSP)*** - The basic idea behind TSP is to find the shortest path that can enable one travel among defined locations once and return to the starting point [12]. The path cost

could be just distance or a combination of cost factors including distance, time spent, among others. However, it should be noted that the classical GA operators will need to be modified in so way in order to yield optimal solution for such practical journey planning purpose. Example of this modification is the partially-mapped crossover (PMX) operator as used by [12] and [27].

3. ***Automated Test Case generation in Software Engineering*** - Automated unit testing are done with automatically generated test data that tends to cover all test cases that are needed to expose the errors in a computer program. Because comprehensive testing is not possible as exponentially large test cases are required, a heuristic method is required to automatically generate the required amount of test to prove the validity and robustness of software systems. The use of GA has proven efficient in automated test generation. This fact has been proven by researchers in [26] in the Evosuite project where they wrote that Evosuite uses GA internally to generate candidate test suites that meets some coverage criteria, by using a given fitness function.

2.4 Previous work on Timetabling Systems using GA

In this section, the contributions of other researchers in solving University Timetabling Problem (UTP) will be looked into. For the purpose of critical analysis, the works of selected researchers will be discussed in terms of (a) the problem instance and size (b) Problem representation method (c) Adaptation of the GA to solve the problem and (d) how the research work relates to this dissertation.

Various methods have been used by different researchers to solve UTP. The work done by [6] suggests that the methods of neural networks, simulated annealing, Tabu Search constraint programming and greedy search could be used to solve timetabling problem. In the research, they compared these methods to the method of Genetic algorithms (GA) to solve the same problem. They concluded that even though comparable results can be obtained from any of the methods, the method of GA has better performance in terms of efficiency and solution optimisation - hence, the use of the GA method in this work.

The researchers in [6] applied GA to solve a High school timetabling problem. Here, few hard and soft constraints were chosen. Also the notion of some classes relating to Laboratory and others relating to tutorials and lectures were not taken into consideration. They only considered situations where no special facilities were needed in a room for some type of lectures. The problem was represented as a 2-dimensional matrix in which the venues are the column headers and the time-slots are the row headers. Each cell, which is an intersection of a venue and a time-slot, contains a Class-Teacher tuple. This representation is shown in table 2.1 below.

These researchers did not go into the details of fitness function as they were comparing the final results got from different methods. The emphasis was not on GA. They used ONEMAX () function that returns the total number of 1s for all constraints that are NOT violated by a chromosome.

Table 2.1: Tuple-based UTP Problem representation [6]

	R1	**R2**	**R3**	**...**	**R*n***
T1	(C2,L3)	(C1,L2)	(C7,L1)	...	(C10,L*i*)
T2				...	
T3		(C4,L9)		...	
T4				...	

T5	(C4,L12)		(C3,L5)	…	(C2,L3)
…	…	…	**...**	…	…
T*m*		(C*j*, L11)		…	(C8,L*i*)

In table 2.1 above, there ***n*** rooms (R), ***m*** timeslots (T), ***j*** Courses (C) and ***i*** lecturers (L). Hence, the first event is (C2, L3). This means that Lecturer 3 will be teaching Course 2 at Room 1 on the first time slot T1. The above table represents an individual chromosome, which is a candidate solution.

The genetic operators were implemented on the above by exchanging either rows or columns between parents in order to produce new offspring. Also, the Course-Lecturer tuple was assigned in such a way that a lecturer can only be mapped to a subject they can teach.

The method of representation used by [6] is a good attempt and starting point for a typical UTP. However, in a typical UTP, a laboratory class will need to hold in a special type of room. Also room size will also need to be considered. The above method of representation does not allow fine-grained and random application of genetic operators. Mutation will be very difficult to define in the above method. However, mutation by swapping as described in section 2.3.2.3 could be easily implemented.

The researchers in [27] took an approach that is typical of University systems. The problem size was quite large and that probably made them to apply the grouping representation technique. Here, each class event is related to a group of students that are taking the same class. There were two separate chromosomes to represent the entire problem:

1. A ***master timetable*** in which the lecturer and time of class is allocated to a group and a course (module).
2. A ***room timetable*** that allocates rooms to groups based on time slices in step 1 above.

The representation for the first case is shown in table 2.2 below.

Table 2.2: Group-based UTP representation technique

G1L1	G1L2	G2L2	…….	G***n***L***m***
M1T12	M3T3	M4T8	…….	M***i***T***j***

In the above table, Group1 takes Lecture 1, which has been assigned to master 1 at time slot 12. Note that **G** = Group, **L**=Lesson (Course/Module), **M**=Master (Lecturer/Tutor) and **T** = timeslot.

In the second step each of the columns is assigned to a room ensuring that no room has different events at the same time.

This representation method is suitable in handling cohorts and people in the same specialisation across different levels. Thus, a module is assigned to a group and any student taking the module belongs to that group. The Application of GA to this representation will also be simple as one can easily apply crossover operator and mutation-by-swap operator. However, just as the authors in [6] no mention was made of the type of room to know if it will be suitable for the type of lecture assigned to it. Besides, defining groups without

overlaps will be difficult. Certain modules belong to more than one group. In such case, it will be difficult to say in what group a lecture will belong. Elective modules may not be properly handled by this method. Hence, there is no guarantee that a student will not be expected to have two different lectures at the same time - impossibility. However, a hard constraint might be defined and used to check this.

The work done by researchers in [28] seemed most relevant to a typical UTP. The problem space and size was large enough to represent a University faculty. They also considered a lot of hard and soft constraints which are very typical of a University environment. Furthermore, they noted the difference between normal lecture room and laboratory room. The representation is similar to Table 2.1. However, the venues were classified into labs and lecture rooms. Also, they employed the Sector-based Partially-mapped crossover (SB-PMX) operator to ensure that the types of rooms are not mixed up in the process. The chromosome representation used by these researchers is reproduced in table 2.3 below. Classes (Lecturer - Students meeting in a venue) were represented as events, which are the same as the tuples used in table 2.1 above. In Table 2.3, each working day is divided into time slots as shown in column headers.

Table 2.3: Sector-based UTP representation Technique [adapted from 28]

Venues/Timeslots	**Mon1**	**Mon2**	**Mon3**	**…**	**Fri7**	**Fri8**
Room 1	Event1-1	Event1-2	Event1-3	...	Event1-39	Event1-40
Room 2			Event 2-3	...		
Room n	Event n-1	Event n-2		...	Event n-39	Event n-40
Lab 1	Event b1		Event b3	...		
Lab 2		Event b2		...		
Lab m		Event bm		...		Event b-40

From the above representation each gene in the chromosome is an event. Each event must be classified as either a lecture or lab event. Hence, there is a constraint that checks if an event is being allocated to a proper room where it can be held. The thick block in the chromosome is used to illustrate how SB-PMX is applied. In this technique, according to these researchers, a given block from one chromosome is exchanged with ***corresponding*** block in another chromosome. It should be corresponding to ensure that genes from corresponding sector are exchanged with each other. It does not matter if they are empty and carries no genetic information. The practical implication of SB-PMX is that it does not make sense to replace a gene from a lecture room sector with a gene from lab sector. It would violate a potential hard constraint. The fitness function used by these researchers is given in equation 2.1 below.

$$Eval.(f) = 1/ (1 + cost(f)) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots .. \ (2.1)$$

Where cost (f) is a weighted sum of the penalty due to a violated soft constraint. The objective is to minimize cost and make Eval(f) as close to unity as possible.

The representation and GA adaptation used by researchers in [29] are relevant in situations where some timeslots are to be compulsorily kept vacant and when some lecturers can only teach within specific time

slots. They used the grouping representation concept (though in a different sense) and what they called the odd-even number representation technique. Odd integers represented morning events while even integers represented evening events. Hence, the chromosome is a set of integers placed inside a room. A single integer was used to represent time, module and lecturer.

The researchers did not define the problem size and instance to which this concept could be applied to. Their representation method is suitable for special cases in timetabling problem and in maintaining diversity (through injection and sorting as suggested by them), but will not scale for other hard constraints like room type. This is because a lot of information was lumped into a single integer. However, their concept could be modified into a constraint check to enhance the optimization of any obtained feasible solution.

The work of Rupert et al in [32] at the University of Nottingham did not focus on representation but on the algorithm for evolution of solution without violating most fundamental constraints. They focused on the best form of heuristics that could be used to ensure the production of a highly optimised table. Also there work is rather on Exam timetabling and not course timetabling. In terms of problem size, they asserted that the hybrid heuristic algorithm can perform well for very large problem sizes. However, 'how large' was not defined and they agreed that the problem instance used was small for experimentation purposes. Also, they did not take into consideration that some rooms may need to be specifically reserved for certain type of examination. Hence, this work is relevant in determining what elitism might be relevant to this current work but not on the representation of the problem.

Another pointer to the fact that a multidimensional table representation method is most appropriate for representing UTP is evident from the work of [33] which was carried at Covenant University, Nigeria. The researcher used a 3-dimensional table to represent [day][time][hall]. An actual class is assigned to the point in space defined by this 3-D array. An integer programming method was used in the chromosome to represent the day, time, hall and class. A class is represented by integer which has a lecturer associated with it. Listing 2.1 shows the code snippet used to operate 3-D chromosomes.

```
begin
   allocations[j][k][l] := allocations[day][hall][time];
  allocations[j][k][l-1] := allocations[j][k][l];
  allocations[day][hall][time] := 0;
 allocations[day][hall][time+1] :=0;
 allocations[day][hall][10] := allocations[day][hall][10]-2;
 allocations[j][k][10] := allocations[j][k][10] + 2;
end
```

Listing 2.1 - Code showing crossover operation on 3-D timetable chromosomes [33]

2.5 Summary

Natural evolution uses the concepts of crossover and mutation to randomly transfer inheritable characteristics from parents to offspring. Depending on the environment and the content of gene transferred, only organisms with suitable genetic content will survive while others will be eliminated.

This random evolutionary concept has been adapted, but with progress towards a particular goal, to solve various human problems. One of the adaptations is the GA. In GA, problems are presented first with suitable genotypes that can accurately map onto the observable phenotypes, which represent the real world solution to the problem. Some initial population are generated from the large search space. A fitness function is used to determine the fitness of the individuals in the population. Based on the outcome, parents are selected for

reproduction. Reproduction is carried out through the use of genetic operators such as Crossover and mutation operators. This process runs for a given number of generations in order to produce an optimal solution or at least a feasible solution.

Some researchers have applied the GA to solve UTP. The obvious fact is that each group of researcher has applied it either to different types of problem instance and size or in adapted form. For instance, that which might be considered a hard constraint to one researcher might be a soft constraint to another. Thus, none of them had exact constraints as that of the Department of Computer Science, University of Sheffield - hence, the need for this dissertation project. These constraints requirements and other requirements of this dissertation are discussed in the next chapter.

Chapter 3: System Requirements and Analysis

In this chapter, the requirements of the proposed system will be enumerated. Further analysis of these requirements will be presented as well. The requirements are in form of algorithmic, functional and non-functional requirements. Legal and ethical issues relating to this work will also be considered. Finally, the criteria for testing and evaluation of the system will be stated.

3.1 Problem Description

The timetable officer in the Department of Computer Science is currently faced with some challenges in producing efficient, feasible and optimised lecture timetable that satisfies the requirements of the University, the lecturers, the students and the available resources. Thus, there is a need to use an incremental approach to design, from bottom-up, an electronic timetabling system that puts most of the important constraints into consideration before scheduling a class for students in the University of Sheffield. These specific requirements are presented in section 3.2 below.

3.2 Requirements

The major requirement of this dissertation is to develop a suitable GA to represent and solve the timetabling problem in the Department of Computer Science. A corollary to this is to produce a software front end that solves specific problem instance of the timetabling as listed in Section 3.2.2 below. The Algorithmic requirements are listed in section 3.2.1. The functional requirements relate to the software and will serve as a validation for the developed algorithm.

3.2.1 Algorithmic Requirements

The following requirements relate to algorithm development:

- Design a suitable chromosome representation for the University timetabling Problem (UTP)
- Design suitable Crossover and mutation operators to be used.
- Model each of the listed constraints in section 3.2.2
- Design a suitable fitness function for the GA

3.2.2 Functional Requirements

The functional requirements will be divided into soft and hard constraints requirements. The hard constraints must be satisfied to produce a feasible timetable while the soft requirements will be optionally satisfied to produce an optimised timetable. Hence, the requirements below:

Hard Constraint Requirements

- **H1**: Ensure that no two events are fixed at the same venue at the same time.
- **H2**: Ensure that multiple modules taught by a given lecturer are not fixed at the same time.
- **H3**: Create a timetable where the core modules do not clash with one another for a given class group in DCS and other departments with whom they offer courses together.
- **H4**: Accommodate the time preferences off all lecturers who are part-time workers.
- **H5**: Accommodate all courses taught for the semester for all cohorts.
- **H6**: Accommodate lectures and labs that must hold at a specific time and venue within the week.
- **H7**: Ensure that the room capacity is enough for the number of students allocated to it for a particular class or lab.
- **H8**: Ensure that each class session takes place in appropriate room type.

Soft Constraint Requirements

- **S9**: Ensure that lecturers do not have more than 4 hours of lecture at a stretch per day.
- **S10**: Ensure that students do not have more than 4 hours of lecture at a stretch per day.
- **S11**: Avoid fixing lectures or labs on Wednesday afternoon.
- **S12**: Avoid fixing lectures within lunchtime.

These requirements are further broken down into User Stories and Use cases in section 3.3.1.

3.2.3 Non-functional Requirements

The non-functional requirements relates to requirements that will enable efficiency and accuracy in achieving the functional requirements. These relate to efficiency of algorithm, speed, legal issues, ease of use [5], among others. The various aspects of non-functional requirements taken into consideration include:

- **Ease of use** - The final system should have an intuitive and friendly user interface that will enable the user select and configure the parameters that is required to run the algorithm for generating a timetable.
- **Speed** - The speed of the software must be a significant improvement from how long it takes the timetable officer currently to create a timetable. Memory usage will also be minimised.
- **Ethical and Legal issues** - the software should satisfy data protection act and other legal issues relating to use of private data, where applicable. Ethical and legal issues relating to this software will be discussed further in section 3.3.2 below.

3.2.4 Assumptions and Scope

The following assumptions and scope have been defined for the purpose of this research

- Only the departmental modules offered by computer science students are considered.
- All lecture rooms belong to Computer science department and no other department competes for it.
- All modules have regular periods per week throughout the semester.
- Apart from part time lecturers, all lecturers are available for teaching throughout the working hours in a week.

3.3 Requirements Analysis

In this section a further breakdown of both the functional and some aspects of the non-functional requirements mentioned above will be done. The analysis will involve Use cases, User Stories, Ethical and Legal issues.

3.3.1 Use Cases and User Stories

As jointly asserted by both [5] and [30], ***Use Case*** describes the behaviour of a system under various conditions, showing how it responds to the inputs of a user called the ***primary actor*** by giving some output. Hence, it describes how each stakeholder in the system interacts with it.

A ***user-story*** is a 3-part sentence which is used to describe what a user does as part of his job function. It is written on a story card. A ***story card*** is a simple card which contains a description of the smallest functional unit of the software product [31]. A user story relates to Use Case in that each has an ***actor***, ***action*** and an ***outcome***. However, user-story gives a better breakdown of activities and values and is often written in form of text. Use case is often presented as diagrams. Both deal with stakeholders in the system.

For the UTP system, the timetable officer is the major user. However, information from the University management, Lecturers and Students are also used as input to the system. Hence, the stakeholders are:

*timetable officer, the Lecturer, the students and the University managemen*t. Both Use cases and User stories are derived from the requirements. The Use case diagram is shown in ***Appendix A***, while the user stories are presented in table 3.1 below:

Table 3.1: User Story for the UTP System

S/N	Actor/User	Action	Output/Value
1.	As a Student	I can ensure that none of my modules clash	So that I won't miss any of my classes
2.	As a Student/Lecturer	I can ensure that I do not have more than 4-hours of lectures at a stretch	So that I will not get famished and be able to concentrate.
3.	As a Student	I can print a timetable for a module and for my class group	So that I will know when I have lectures
4.	As a Lecturer	I can have my lab only in the rooms that have suitable equipment	So I can efficiently deliver to students
5.	As a lecturer/Student	I can attend lecture in room with suitable size	So I can comfortably sit and receive lectures
6.	As a timetable officer	I want to ensure two lectures are not fixed at same venue and time	So that I will not get complaints from lecturers and students.
7.	As a timetable officer	I want to ensure two courses taught by a lecturer are not fixed at the same time	So that the lecturer will not complain
8.	As a timetable officer	I want to ensure that a lecturer is assigned only to the courses he/she can teach	So that the students are well taught.
9.	As a timetable officer	I want to ensure a timetable can be printed for a course, a lecturer or a class group	So they only get information relevant to them.
10.	As university management	I want to ensure lectures are not fixed on Wednesday afternoon	So staff/Students are allowed to do their volunteering jobs.

These user stories will help one evaluate the success of this project. This is because each of the stories will either be achieved or not. Hence, the percentage of achieved stories is a good measure of progress. Thus, each of the above will be classified as either hard or soft constraint in Chapter 4.

3.3.2 Ethical and Legal Issues

As part of British Computer Society (BCS) accredited departments' requirements, ethical and legal issues have to be considered for each research work being carried out in the department. This is also required by the University of Sheffield for projects that involve human beings and human data. Though this research will not be on humans and will not involve real private data of Lecturers and Students, the following ethical and legal issues will be considered for this work and its future extension:

- Plagiarism will be highly avoided. All ideas of others, including knowledge and codes will be properly cited and referenced. Permission will be sought first where necessary.
- As this research work is not on human beings or real personal data, no formal ethics review is required. However, for future work where integration with Student and Lecturers' personal data is needed, then formal ethics review will be carried out through the supervisor.
- Though the use of any third-party software is not anticipated, license for such will be properly obtained before integrating it into the software, if required.
- Any configuration parameter which is specific to this problem instance but not to the classical GA will be stated.
- Deliberate falsification of test data or exaggeration of test result will be highly avoided.
- The software will be designed to avoid damage to the hardware and software in the system on which it runs.
- Warning on the use of cookies will be provided where applicable.
- Though most of the information required to carry out this research are in the public domain, any private information provided by DCS or my Supervisor will be kept confidential.
- The limitations and capabilities of the final system will be stated to the best of my knowledge.

3.4 Evaluation and Testing

Testing is often carried out in order to evaluate the performance of Software systems. Also, tests are carried out to evaluate how well or not, the requirements of the system have been met. Hence, testing and evaluation are integral aspects of both Software Development Life Cycle (SDLC) and that of any research work. Detailed tests, results and evaluation will be done in chapters 5, 6 and 7 respectively. However, in this section the ball will be set rolling by highlighting testing and evaluation requirements of this project work. The following tests will be carried out for validation and evaluation purposes:

- **White box testing** - to determine accuracy of functional units of code. Each module and function will be tested for flow and branch coverage. The units will also be compared to the designed algorithm.
- **Black-box testing** - Black boxing is interest in functionality and not structure [5]. This will be used to verify that the parameters used as input into the program produces the correct result. Each constraint will have some set of inputs that are expected to produce a given output. The input parameters will be fed into the system to ascertain how it satisfies the given constraint. Five test cases with varying levels of difficulty will be used to establish the capacity and capability of the designed algorithm and developed system.
- **Integration testing** - to determine robustness at interfaces. The ability of the various methods and modules to work together to solve the entire problem will be tested through the test cases.
- **Acceptance testing** - to ascertain the achievement or not of each of the stories in table 3.1. This will be done via user testing and evaluation by the timetabling officer.

3.5 Summary

The chapter outlined the specific requirements for this dissertation project: algorithmic, functional and non-functional requirements. It also listed the ethical and legal issues relating to this project.

Various tests will be carried out using test cases and results will be measured in terms of progress rate, number of runs with feasible solutions (accuracy), time taken for the GA to converge (efficiency) and quality of the timetable obtained. Five test cases will be used to represent five levels of difficulty.

The evaluation process will involve user evaluation of the product and personal evaluation of the results on how it has or has not met the objectives and requirements of this dissertation.

Chapter 4: System Design

This chapter presents the genetic representation of a timetabling problem typical of a University and also the design of the algorithm required to solve such problem. The design includes that of genotype, phenotype, constraints, fitness function, and crossover operators. The design of the software GUI required to verify the efficacy of the algorithms, using some subsets of the problem, is also shown.

4.1 Design Methodology

The design approach chosen for this dissertation involves analytical, deductive and synthetic approaches to problem solving. The work of previous researchers and academics will be analysed and deductions then made. These deductions will be integrated into this work but with special consideration to how it fits into the problem that we are faced with. The analytical and synthetic approaches will be used for the most part of the algorithmic design. However, the final solution to the problem will be evolved synthetically using the iterative refinement problem solving approach as shown in figure 4.1 below.

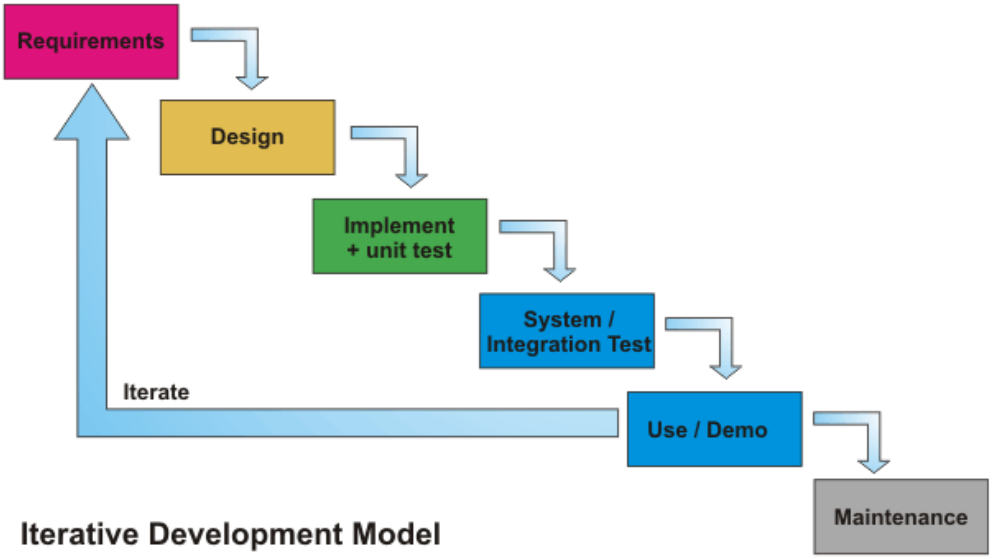


Fig 4.1 - Iterative Design Process [34]

In this iterative process, all steps towards the final solution might be revisited for further refinement. Both the algorithm and the GUI will be iterated upon. The solution will start from simple cases, where there are no class groups and all courses a core courses. This will lead into level 3 with some electives modules and then into Fourth year and M.Sc. where most modules are electives. Thus an incremental and iterative approach will be used in solving this problem.

For the software design, a loose-coupled modular Object-oriented class design approach will be followed. This means classes will be designed such that they are not tightly dependent on each other. Direct class inheritance will only be used where necessary. This design approach, together with modular design will ensure flexibility, maintainability and scalability. Codes can easily be moved around if necessary. Modularity also helps in debugging, quality assurance and reduction in design time.

4.2 Risk Assessment

The factors that may militate against successful completion of this work within the allotted time are identified here. This will help us define the scope and manage the project to a successful completion. These risks, level and actions to reduce the effect of each risk is detailed in **Appendix B**. Likelihood of a risk is a measure of the probability that it will occur. The impact is a measure of the effect of the risk on the success of the project. Here both the likelihood and impact are measured in a ***scale of 1 - 4***, with 4 being most likely and

threatens the entire project and 1 being very unlikely and having negligible effect, respectively. Action required to reduce the Exposure are also included. The ranking is done according to exposure using equation 4.1.

Exposure = Likelihood x Impact …… *(4.1)*

Risk assessment is a requirement for BCS accredited departments and this was duly taken into consideration for this dissertation.

4.3 Algorithmic Design and Mathematical Modelling

This section sets out all the algorithmic parameters and processes that will be employed to solve the timetabling problem provided in chapter 3. Chromosome representation, modelling of constraints, Fitness function, genetic operators and the overall GA to be implemented will be discussed in this section.

4.3.1 Notations

The following parametric notation will be used:

- **T** => **Time slots** of one hour duration starting from 9am to 5pm Monday to Friday.
- **R** => A **Room** for either laboratory session or lecture, which holds only one event at a time.
- **M** => **Module**, which is a course which may have both lab and tutorial components
- **L** => **Lecturer** who teaches a particular Module
- **C** => **Class group of students** in a particular discipline offering common Modules
- **E** => **Event** of either tutorial or Lab session holding at a given time **T**, at a room **R**, under the control of a Lecturer **L** and involving a class group **C** on a module **M**.

There will be ***a*** Time-slots ($\mathbf{T}_{a}$), ***b*** rooms ($\mathbf{R}_{b}$), **c** Modules ($\mathbf{M}_{c}$), **d** Lecturers ($\mathbf{L}_{d}$), **e** Class groups ($\mathbf{C}_{e}$) and **f** Events ($\mathbf{E}_{f}$);

Where **a, b, c, d, e** and **f** are positive integers.

The data structure and properties of each of **T, R, M, L, C** and **E** are further defined in section 4.3.2 below to enable the required genotypic - phenotypic representation and transformation to produce the final timetable. The representation of the timetable chromosome, choice of genetic operators and design of fitness function will be based on clear definition of the parameters stated above.

4.3.2 Chromosome Representation

As previously stated in section 2.2, a chromosome is made up of a collection of genes. Also these have to be arranged in such a way that they can map onto the phenotype of the individual being evolved. To design a genotype, phenotype and thus a chromosome for the timetabling problem, we need to define the data structure for the ***TRMLCE*** notations above. These are defined below:

- **T** = 1 to 40, where 1= Monday 9am …40 = Friday 4pm. Hence, there are 40 timeslots in a working week - 8 per day from 9am - 4pm.
- **R=** (***id***, name, capacity, room_type), where ***room_type*** = [‘lecture’, ‘lab’].
- **M =** (**id**,code,type,level,numstudents,lecturehours,labhours,category,cohort)
  -type = [‘lab’, ‘lecture’, ‘both’].
  -level = this is the academic level the module is first studied (1...6).
  -category = [‘core’, ‘elective’, ‘both’].
  -cohort = Class group in the level that has this module as core module.
- **L =** (***id***, lecturername, type, department), where ***type*** = [‘fulltime’, ‘partime’].

- **C =** (***id***,cohortname, numstudents,level_of_study
- **E =** (**id,** T,R,L,C,M) - id (subscript **f**)

To define the chromosome representation, we need to first state the genetic representation.

Genetic Representation

For simplicity, only the time slots, rooms and modules will be represented genetically.

The genetic representation will be the integer id which uniquely identifies any of the Modules at a locus defined by timeslot T and room R.

Phenotypic Representation

With the data structure defined in section 4.3.2, the phenotypic representation is evident. With the data properties of an individual gene, its presence or absence can be identified in a chromosome. For the UTP, the important phenotype is the event **E**. If only T, R and M are represented directly in a chromosome, then we need an intermediate translation to include L and C into the event. To achieve this, each module must have a lecturer (L) property and also a cohort (C) property and relationships described as follows:

- ***Module_Lecturers*** (id, module_id, lecturer_id) - This serves as module allocation and thus assigns a lecturer to a module. Fig 4.2(a) is its genetic representation.
- ***Cohort_Modules***(id, cohort_id, module_id) - This serves as academic regulation that assigns modules to a class group to ensure that they take the right module combinations that are relevant to the degree they will be conferred. The genetic representation for this is shown in Fig 4.2 (b) below.

id	lecturer_id	course_id
1	2	6
2	2	2

id	cohort_id	course_id	level
1	1	1	1
2	1	2	1

Fig 4.2 (a) - Course Allocation ***Fig 4.2 (b) - Academic Regulation for cohorts***

With these definitions, it will be possible to associate a module with more than one lecturer and also associate a module with one or more cohorts or class groups. A diagram to illustrate the required transformation from the genetic to the phenotypic domain is shown in Figure 4.3 below:

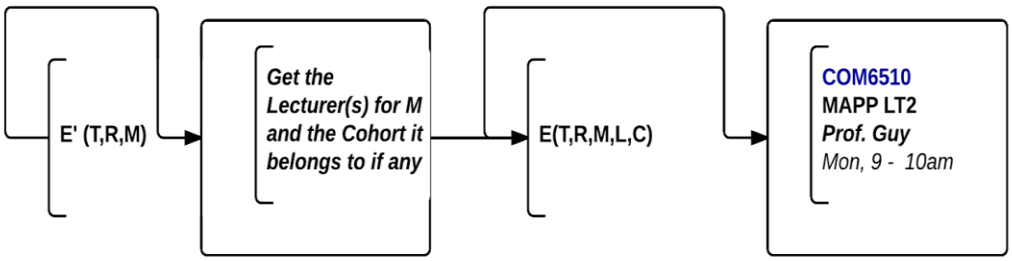


Fig 4.3 - Genotype - Phenotype Transformations for the UTP

It should be noted that T, R, M, L and C as used in Fig 4.3 represent integers that uniquely identify a time-slot, lecture or lab room, module, lecturer and cohort respectively. E' is the event as directly represented in the chromosome while E is a complete intermediate transformation of the genetic content before it is finally mapped onto its equivalent phenotype that represents a part of a timetable.

Having defined genetic content and their corresponding phenotypic transformation and having in mind that Laboratory rooms should be made different from normal lecture rooms, an adaptation of the representation used by researchers in [28] will be used for this dissertation work. Thus the chromosome structure for this UTP is shown in Table 4.1.

Table 4.1: Sector-based UTP representation Technique [adapted from 28]

Rooms/Timeslots	Mon1	Mon2	Mon3	…	Fri7	Fri8
Room 1	1	8	7	...	7	5
Room 2			3	...	4	
Room n	6	6		...	2	9
Lab 1	3	3	6	...		
Lab 2		4		...		
Lab m		1		...	10	10

It should be clearly stated the alleles for both rooms and timeslots will be represented by integers. Repeated courses in the timetable mean more than one timeslot. Also some courses have only lecture component, only lab component and both lab and lecture component. All these have been taken care of in the chromosome representation.

4.3.3 Constraints' Algorithm Design and Modelling

In this subsection, algorithmic design for each of the constraints described in section 3.2.2 will be done. In this design, each constraint will be given a counter which accumulates to indicate how well the particular constraint has **NOT** been violated. Each instance of ***non-violation*** of the constraint will ***increment*** the counter by one. Hence, ***a reward is given for each instance of non-violation of a constraint***. Thus a chromosome with ***all*** instances of a constraint violated will have a counter value (reward) of 0 for that constraint. Thus, a reward-based approach will be used.

The constraints were divided into hard (H1-H8) and soft (S9-S12) constraints. We revisit each here.

1. ***H1: Ensure that no two events are fixed at the same venue at the same time.***

This has been taken care of by the choice of the chromosome representation method. Looking at table 4.2 above, it is evident that the locus of an event gene (Lab or lecture session) is defined as the intersection of a unique venue and time. The above constraint has been satisfied.

2. ***H2: Ensure that multiple modules taught by a given lecturer are not fixed at the same time.***

The following algorithm will be used to detect the violation of this constraint:

```
For each lecturer in LecturerCount
        For each timeslot
                set countPerTimeslot=0
                For each lecture/lab room
                        if event belongs to current lecturer
                                increment countPerTimeslot
                        end
                repeat
                If countPerTimeslot  <=1
                        increment counter for this constraint
```

```
                    end
                repeat
            repeat
```

Algorithm H2 - Multiple lecturer fixtures detection algorithm

If the value of counter for this constraint equals ***40*LecturerCoun***t at the end of executing this algorithm, it means that this constraint was not violated for this individual chromosome. Algorithm H2 works by picking a lecturer and a timeslot and then moving through all the venues available to check if the lecturer has an event scheduled in any of the rooms during the time slot. If there is zero or one event, then a reward is given for NOT violating the constraint for that lecturer at that time slot. Thus maximum reward for Hard Constraint 2 ($H2_{max}$) is given by equation 4.1 below.

$$H2_{max} = 40 \text{ time slots} * Number\_of\_lecturers. \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.1)$$

3. H3:None clashing timetable schedule for the core modules in a class group

Before the GA is run, all modules must have been designated as core or elective module. It must also have been assigned to one of the class groups and for a particular level of study. Table 4.2 below states the class groups considered in this research.

Table 4.2 - Cohorts considered for the research

S/N	Cohort/Class group	Start Level	Duration (yrs)
1.	B.Sc. Computer Science/B.Eng. Software Engr.	1	3
2.	M.Eng Software Engineering	1	4
3.	M.Sc. Advanced Computer Science	6	1
4.	M.Sc. (Eng.) Advanced Software Engineering	6	1

Algorithm H3 below will be used to detect the violation of this constraint.

```
For each Cohort in each level
    For each timeslot
        set countPerTimeslot=0
        For each lecture/lab room
            if event belongs to current cohort and level
                increment countPerTimeslot
            end
        repeat
        If countPerTimeslot  <=1
            increment counter for this constraint
        end
    repeat
repeat
```

Algorithm H3 - Clashing module detection algorithm

Hence, if two modules belonging to the same cohort are allocated to any of the rooms in the same time slot, the reward from this constraint will decrease. The maximum reward ($\mathbf{H3_{max}}$) for this constraint is given by equation 4.2 below.

$$H3_{max} = Timeslots * \sum_{i=1}^{N} C_i * n \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.2)$$

Where ***Timeslots*** = Number of smallest time units (40 for this research work)

N = Total Number of Cohorts considered (5 for this research work)

C_i = Cohort number I, ***n*** = Duration of Study per cohort.

The maximum reward needs to be achieved per chromosome to ensure that it does not violate any instance of this constraint. From table 4.3, the maximum reward per chromosome is:

40[3+4+1+1] = 40 * 9 = 360.

4. H4: Accommodate the time preferences of all part-time lecturers.

Before the timetable is generated, the time of availability for each part-time lecturer will be input into the system. It will be in form of time blocks per day. Examples are 12-5pm Mondays, 9am - 5pm Tuesdays and 9am -12 noon Fridays. Hence, a reward will be given for each part-time lecturer whose class schedules all fell within his/her chosen available time. To detect the violation of this constraint, Algorithm H4 below will be used.

```
For all part-time lecturers
        Get the modules allocated to each lecturer
        Get the available time for each lecturer
        Get the time slot allocated to each module
        if all scheduled time within lecturer available time
                increment counter for this constraint
        end
Repeat
```

Algorithm H4 - Part-time lecturer availability constraint algorithm

The maximum reward for the above constraint is simply:

$\mathbf{H4_{max} = L_p}$ …… (4.3)

Where, $\mathbf{H4_{max}}$ = Maximum reward for NOT violating part-time availability constraint

$\mathbf{L_p}$ = Total number of part-time lecturers considered for the semester.

5. H5: Accommodate all courses taught for the semester for all cohorts.

The modules taught for a semester need to be represented in the timetable. In order to put this into account, the following algorithm will be used to detect if there is a missing module in an individual chromosome:

```
num_times_scheduled = 0, setModule = module[i]
For each allocatedModule in Chromosome
        For each timeslot in a week
                For each lecture/lab room available
                        if setModule ==allocatedModule
                                increment num_times_scheduled
                        end
                repeat
        repeat
        if num_times_scheduled !=0
                increment counter for this constraint
        end
        num_times_scheduled =0
repeat
```

Algorithm H5 - Algorithm to ensure all modules are scheduled in the timetable

In the above algorithm, a module is picked and searched all through the chromosome to count the number of times it has been scheduled. If after searching and the number of times it was scheduled still remains zero, then it means that it has not been scheduled at all and no reward will be given based on that module. The maximum reward (**H5*max***) for non-violation of an instance of this constraint is given by equation 4.4 below.

$$\mathbf{H5}_{max} = \text{Total Number of Modules to be Scheduled} \quad \text{..... (4.4)}$$

In this research, a module with both lecture and lab components are considered two separate modules.

6. H6: Accommodate lectures and labs that must hold at a specific time and venue within the week.

Some lectures and lab events must hold in certain rooms. For instance, COM6510 lab can only hold effectively in the MAC Lab. Hence before the algorithm is run, certain modules will be selected and fixed to hold in a certain room. The violation of any of these constraints will be detected with Algorithm 5 below:

```
For each room_constrained module
        Find the room it's allocated in the chromosome
        if allocated_room ==  set_room
                increment counter for this constraint
        end
repeat
```

Algorithm H6 - Special room allocation constraint detection algorithm

The maximum value or reward from the above algorithm is the sum of all individual lectures and labs that must hold in specific rooms and specific times, $\mathbf{N}_s$. If the maximum reward for not violating any instance of this special requirement is $\mathbf{H6}_{max}$, then equation 4.5 below gives the maximum reward for this constraint.

$$\mathbf{H6}_{max} = \mathbf{N}_s \quad \text{..... (4.5)}$$

7. H7: Ensure that the room capacity is enough for the number of students allocated to it for a particular class or lab.

To ensure that a lecture or lab is allocated to a room size that can accommodate the number of students who registered for the module, Algorithm H7 below will be used:

```
Set tolerance=10
For each module scheduled on a timeslot
        get the room it is scheduled in
        get room size
        get number of students in module
        if num_students < room_size OR num_students < room_size + tolerance
                increment the counter for this constraint
        end
repeat
```

Algorithm H7 - Matching room size and student number constraint algorithm

Algorithm H7 above is not based on number of modules, but based on each timeslot that has a module scheduled in it. Thus the maximum reward here is the total number of non-empty time slots on the timetable. Thus maximum reward for this Constraint, $\mathbf{H7}_{max}$ is given as:

$$H7_{max} = \sum_{i=0}^{a} \sum_{j=0}^{b} E_{ij} \ for\ all\ E_{ij} \neq 0 \text{ ..... } \textbf{(4.6)}$$

Where a = maximum number of timeslots (40)
b = maximum number of rooms, E = Event, which is a lecture or lab event.

8. H8: Ensure that each class session takes place in appropriate room type.

To ensure that a lab session is not allocated to an ordinary lecture room and vice versa, Algorithm H8 below will be implemented to detect the violation of this constraint.

```
For each module scheduled on a timeslot
        get the module type
        get the room type it is scheduled in
        if moduleType == room_type
                increment the counter for this constraint
        end
  repeat
```

Algorithm H8 - Matching room type constraint Algorithm

The maximum reward for this Constraint, $\mathbf{H8_{max}}$, is the same as equation 4.6 above.

9. S9:Ensure a lecturer does not have more than 4 hours of lecture at a stretch per day

Each lecturer has a module allocated in form of lecture or lab. As lectures and labs count separately, Algorithm S9 below will be used to check if a lecturer has more than 4 hours of lectures or lab placed at a stretch in the timetable.

```
For each lecturer
        Set eventCount = 0
        For all timeslots within a single day of the week
                if event(t) and event(t+i) belongs to this  lecturer
                        increment eventCount
                else
                        eventCount=0
                end
        repeat
                if eventCount = 4
                        increment count for this constraint
                end
        set eventCount=0
repeat
```

Algorithm S9 - Checking lecturer 4 hours stretch lecture algorithm

The constraint is only violated when the next four or more consecutive events to the current event belongs to the same lecturer under consideration. A reward is only given when no such violation exists for that lecturer throughout the week. Thus it is obvious that the maximum value for this constraint, $\boldsymbol{S9_{max}}$, is the total number of lecturers, ***d***, for the semester. This is given by equation 4.7 below.

$$\boldsymbol{S9_{max}} = \boldsymbol{d} \text{ ..... } \textit{(4.7)}$$

10. S10:Ensure a student does not have more than 4 hours of lecture at a stretch per day

This constraint is similar to constraint 9 above. However, an event will be checked against a class group and not an individual student or lecturer. Hence, Algorithm 8 above will be adapted to check the student version as shown in Algorithm S10 below.

```
For each Cohort
        For each of the levels of study in a cohort
                Set eventCount = 0
                For all timeslots t  within a single day of the week
                        if event(t) and event(t+i) belongs to this  cohort
                                increment eventCount
                        else
                                eventCount=0
                        end
                repeat
                if eventCount = 4
                        increment count for this constraint
                end
        repeat
repeat
```

Algorithm S10 - Checking Student 4 hours stretch lecture algorithm

A reward is given per cohort, per level of study for all the days of the week. The maximum reward (**$S10_{max}$**) for this constraint is given by equation 4.8 below.

$$S10_{max} = \sum_{i=1}^{N} C_i * n \quad \text{..... (4.8)}$$

Where ***N*** = Total Number of Cohorts considered (5 for this research work)

C_i = Cohort number i

n = Number of different year groups in a cohort (Duration of Study)

From Table 4.3, **$S10_{max}$** =[3+4+1+1]= **9.**

11. S11:Avoid fixing lectures or labs on Wednesday afternoon

Based on the genetic representation used in this dissertation, Wednesday afternoon (12noon to 4pm) are the timeslots 21 - 24. Hence, Algorithm S11 below will be used to check this constraint.

```
Set currentChromosome
For timeslot = 21 to 24
        for room = 1 to roomCount
                if chromosome[currentChromosome][room][timeslot] ==0
                        Increment count for this constraint
                end
        repeat
repeat
```

Algorithm S11 - Checking Wednesday afternoon schedules algorithm

The maximum reward for this constraint **$S11_{max}$** = 5, which is the number of hours in a Wednesday afternoon (12noon, 1, 2, 3,4pm).

12. S12:Avoid fixing lectures within lunch-time (1 - 2 pm)

Using the genetic representation for timeslots as explained in section 4.3.2 above, the launch time for Mondays to Fridays are: 5, 13,21,29,37. Thus for each day and each room in which no lecture or lab is scheduled within these time slots, we increment counter for this constraint to show how good our individual chromosome is. This results to algorithm S12 below.

```
Set currentChromosome
For timeslot = 5; increment timeslot by 8 till you reach 37
        For room = 1 to roomCount
                if
        chromosome[currentChromosome][room][timeslot] !=0
                        Increment count
                end
        repeat
        if count ==0
                increment counter for this constraint
        end
        count=0
repeat
```

Algorithm S12 - Checking Lunch time schedules algorithm

There are five lunch periods each week. That is one per day. Hence, maximum reward from non-violation of any instance of this constraint is given $\mathbf{S12}_{max} = \mathbf{5}$.

4.3.4 Fitness Function

Having defined the rewards for non-violation (or penalties for violation) of constraints, it is time to define the fitness function that will be used to determine the fitness value for each chromosome. The fitness function is divided into **Hard Fitness** and **Soft fitness** functions. The Hard fitness function is simply the sum of all the percentage rewards from each constraint considered as hard (H) constraint. A reward-based fitness function will be used instead of a penalty-based fitness function. Hence, the objective here is to maximise the fitness value. Thus, the algorithm visits all points of potential violation of a particular constraint and not just summarising if a constraint is violated or not. This approach ensures that there is no leakage in fitness evaluation. However, one might reason that it would prolong the run time for the algorithm, if there are many good genes in a chromosome.

This is given as *f (**H**)* in equation 4.9 below.

f (**H**) = % of **H2** + % of **H3**+% of **H4**+% of **H5**+ % of **H6**+ % of **H7**+ % of **H8** ……. *(4.9)*

For a chromosome to be a feasible solution to the timetabling problem, equation 4.10 must be satisfied.

f (**H**) = 100% of $f(\mathbf{H})_{max}$ ……(4.10)

Where $f(\mathbf{H})_{max} = \mathbf{H2}_{max} + \mathbf{H3}_{max} + \mathbf{H4}_{max} + \mathbf{H5}_{max} + \mathbf{H6}_{max} + \mathbf{H7}_{max} + \mathbf{H8}_{max}$ ……..(4.10b)

On the other hand, the Soft fitness function, f(S), is the sum of the rewards from each constraint considered as soft(S) constraint. This is given by equation, 4.11:

F(**S**) = **S9** + **S10** + **S11** + **S12** …… (4.11)

The maximum value that f(**S**) can have is given as $f(\mathbf{S})_{max}$ and is given by equation 4.12.

$F(\mathbf{S})_{max} = \mathbf{S9}_{max} + \mathbf{S10}_{max} + \mathbf{S11}_{max} + \mathbf{S12}_{max}$ ……..(4.12)

Thus, the overall fitness value,**$FV_{overall}$**, for a chromosome is therefore a combination of equations 4.9 and 4.11. Thus:

$$\mathbf{FV}_{overall} = f(\mathbf{H}) + f(\mathbf{S}) \quad \ldots\ldots (4.13)$$

However, given that equation 4.10 is true for a chromosome, it becomes evident that equation 4.11 is the **objective function** that the genetic algorithm will tend to maximize.

Equations 4.9 to 4.13 are simple linear equations. It is very easy to add more soft or hard constraints as well as move a soft constraint to become a hard constraint and vice versa. The significance of this design is to enable future researchers to re-use the algorithm based on the nature of their problem.

Thus, this design is believed to have adequately satisfied the scalability and flexibility requirements of this dissertation.

4.3.5 Choice and Design of Genetic operators

The genetic operators to be considered and designed here include: **selection**, **crossover** and **mutation** operators.

Choice of Selection operator

At this point, it might be helpful for a reader to refer back to **2.3.2.1** in **Chapter 2.** The nature of the timetabling problem being solved here is such that any good candidate solution needs to be retained into the next generation. However, in order to escape from premature convergence and local optima, high diversity is also needed. In order to strike a good balance between these two, this GA shall combine elitism and tournament selection operators.The algorithm for initializing the population in the case of constructive initialisation method will ensure that most of the hardest constraints (e.g H6 and H8) are not violated. The algorithm for the tournament operator is listed in Algorithm A13 below.

```
Randomly select any two chromosomes of different fitness from the entire population
Set predetermined_probablity
Generate a random number between 0 and 1
if random number  <= pre-determined_probability
        select the fitter candidate
else
        select weaker candidate
end
```

Algorithm A13 - The tournament selection algorithm

A pre-determined tournament selection probability of 0.7, will be used in this design. This is to favour the selection of better chromosome for reproduction.

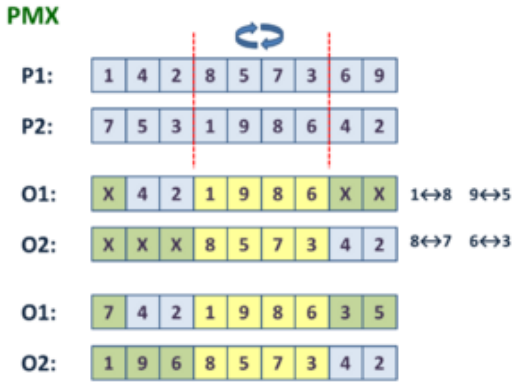
Choice of Crossover operator

The PMX algorithm has been adapted in this research for a 2-Dimensional chromosome unlike the 1-dimensional chromosome often used. It is listed in Algorithm A14 below:

Randomly select a segment (a,b) of genes from parent 1 and copy them directly to OffSpring2(O2).
For each element in same segment,i=a to b, in parent 2
Check if $P2_i(r,t)$ has been copied to O2
if not yet copied
while $P2_i(r,t)$is part of segment (a,b)
find the corresponding value,v, from parent1 i.e P1i(r,t)
get the location of v in Parent2 i.e P2v(r,t)
$P2_i(r,t) = P2v(r,t)$
repeat while
$O2_i(r,t) = P2_i(r,t)$
end if
repeat for
Copy any remaining positions from Parent2 to Offspring2
Reverse roles of P1 and P2 to create Offspring 1 (O1)

Algorithm A14: 2-D PMX crossover Algorithm

For the above algorithm, ***r*** = room index, while ***t***= time index. A gene in our representation is located by both indices. One-dimensional example of the above algorithm was given in [37] and partly reproduced in figure 4.3 below.



In this example, a=4, b=7. So segments 4-7 from P1 was copied directly to O2 while segments 4-7 from P2 was also copied directly to O1. Remaining parts P2 and P1 were used to complete O2 and O1, respectively following the rules from the algorithm above. The major idea is to maintain the origin position of most of the genes from parent to child.
In this research, another auxiliary algorithm will be used to ensure that the number of periods for each module is maintained after this crossover. This concern has been raised by the article in [37].

Figure 4.3 - PMX Crossover operator in 1-dimension [37]

Based on research output from [28] and [36] and based on experimental results in this research, the crossover probability for choosing points a and b in this research will be 0.75.

Choice of Mutation Operator

Because the genes used here do not have a cardinality of 2 (binary), we will use a swap mutation operator. The mutation operator will find a nonempty gene allocated to a randomly chosen time slot and randomly select another empty or nonempty gene to swap itself with. The algorithm is listed in A15 below.

Set a mutation_probability
for mutation_probability x no_of_timeslots
pick a room at random
locate a module-occupied gene C1(r,t) in this room
randomly pick another location C2(r,t)
copy C1(r,t) to a temporary location temp

copy C2(r,t) into C1(r,t) *copy temp into C2(r,t)* *repeat*
Algorithm A15 - Algorithm for the Swap Mutation Operator

In the above algorithm C1(r,t) must not be nonempty while C2(r,t) may or may not be empty. This is because it would not be sensible to swap two empty genes. It will not have any effect on the chromosome. A probability of 0.01 (or 1%) will be used to avoid possible disruption of very good chromosomes especially in the case of constructive initialisation GA.

4.3.6 Determination of termination criteria

The termination criteria are the conditions under which the GA would stop for a particular run. In a broad sense and as described in [35], three conditions should lead to the termination of a GA:

i) Optimum Solution has been found,
ii) The GA is taking too long to converge or find at least a feasible solution,
iii) The chance for improvement of fitness is low or reduction in fitness has set in.

However, in this research, the GA will be terminated when either (i) or (ii) below is satisfied:

i) **Optimum solution is obtained**. This is when the fitness value for a chromosome equals the maximum fitness that an individual chromosome can have.

ii) **The GA has been run for 500 generations**. This solves (ii) and (iii) above. "Too long" is dependent on the speed of computer in use. For low-end CPU-based systems and for web applications, this will just be enough time to avoid request timeout. Also, with constructive initialisation, appropriate genetic operators and good amount of resources, feasible solution could be reached in some cases for this number of generations.

Thus the above termination criteria were chosen to avoid request time-out and avoiding the possibility of infinite loops.

4.3.7 The overall GA and chosen Parameters

For the purpose of this research, the GA parameters listed in table 4.3 will be used:

Table 4.3: Parameter set for the proposed GA

S/N	Parameter	Value
1	Population size (N)	100
2	Number of Generations (G)	500
3	Probability of crossover (Pc)	75%(0.75)
4	Probability of Mutation (Pm)	1%(0.01)
5	Selection operator	Tournament Selection
6	Crossover operator	PMX
7	Mutation operator	Swap Mutation

The diagram in figure 4.4 below is the modified GA that will be used to solve the UTP in form of constructive initialisation GA. This algorithm is not very different from the classical GA shown in Fig 2.4 in chapter two. However, emphasis in figure 4.4 is on constructive initialisation and elitism. Constructive

initialisation involves checking if the allocation of a module in a timeslot violates a constraint or not. Not all constraints are actually checked. Constraints H4 and H6 are special cases which may never be satisfied by a complete random process as is the case with non-constructive initialisation. So one still has to allow some level of randomisation in constructive initialisation otherwise it would no longer be amenable to GA processes.

It should be noted that new algorithm is not required for the constructive initialisation. It follows from the algorithms already defined for each of the constraints.

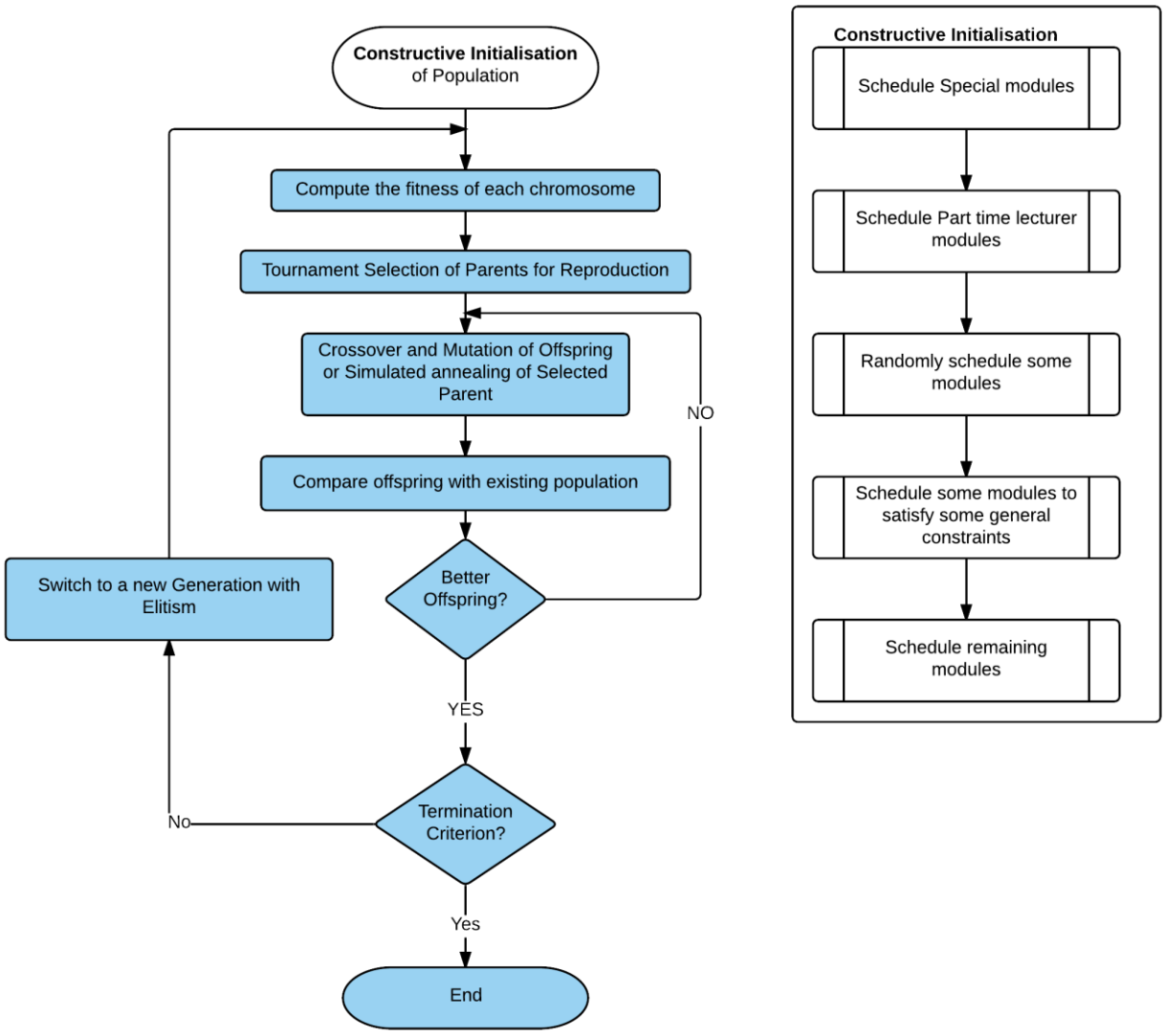


Figure 4.4 – Adapted GA through Constructive Initialisation Process

4.4 Software Design

In this section the software that will be used to implement aspects of the designed algorithms will be taken into consideration. An Object-Oriented design approach will be used to allow for flexibility and reusability. Software architecture, class design, software tools and database design will be discussed.

4.4.1 Design of software architecture

A web-based platform is adopted. This is because it is envisaged that various departments and faculties will use the software. It is also believed to possibly integrate with existing Corporate Information and Computing Services (CiCS) of the University of Sheffield.

In terms of Architecture, a Model-View-Controller (MVC) design approach will be used. This design approach helps to separate concerns among data sources, business logic and presentation to users. It ensures maintainability of all parts of code and makes it easier to update changes [38]. Furthermore, a loosely-coupled class design approach will be used as stated earlier. These architectural features are intended to allow for easy review of the entire algorithm and code in the future.

4.4.2 Choice of Programming Language

Java is a cross-platform programming language. It also has strong computational ability and memory management capabilities. GA requires a lot of computation and Java can comfortable handle the scope and number of iterations considered for this study.

Besides, the researcher is more competent in Java. This is to avoid further waste of time in learning similar languages like Scala, Ruby-on-rails, C#, MATLAB among others.

4.4.3 Design of software modules

An elaborate Unified Modelling Language (UML) class diagram is presented in the **Appendix C**. A brief description of each of the modules and class in the diagram now follow.

- **UploadInputs** - Information relating to available rooms and their types; lecturers and their availabilities; modules and their type and size, and cohorts and their study levels are uploaded to the database via this module. It extends the ***ReadInputs*** class so that it could utilise the database attributes and methods already defined in it in order to save data to database. An alternative for large scale inputs will be to use a file upload formatted as comma-separated variables (CSV) or in a spreadsheet. However, all data have to be properly structured to ensure data integrity is preserved.
- **ReadInputs** - this module reads all the input saved to the database and converts it into 2-dimensional array which can be held in memory while running the GA. It has direct connection to the underlying database. It is the major data source to all other parts of the application. Figure 4.5 clearly shows how all other modules points (associatively) to this module as it is their major data source. It ensures that no other module has direct access to the database.
- **DBConnection** - This module gives the application actual access to the underlying MySQL database. It uses MySQL Connector/J driver to read from and write into the database. It should be noted that timetables are not saved in the database but the data required to create a timetable all comes from the database. The module gets connection, executes create, read,update and delete(CRUD) queries and closes connection to the database. Only ReadInputs has direct access to this class.
  The classes **UploadInputs**, **ReadInputs** and **DBConnection** serve as model classes in this M-V-C architecture.

- **RunGA** - This class is actually a servlet. It serves as the entry point for generating a timetable. It simply instantiates the Chromosome and Crossover classes and runs the GA iteration for the set

maximumGeneration (500 by default). It then handles the printing of the generated timetable using the best chromosome after the iteration. The best chromosome is passed to **ReadInputs** class so that it can get data from the database to map the genotype into the required phenotype that represents an actual timetable.

- **Chromosomes** - This module handles the initialisation of the entire population using either constructive or non-constructive initialisation algorithm. It also passes the entire population to Crossover module to utilise all the GA operators (selection, crossover and mutation). This class gets all its data through the ReadInputs class.
- **Fitness** - This module computes the hard, soft and overall fitness of a chromosome and thus that of all the chromosomes in the entire population. It returns its results as an integer array. This enables elitism to be applied while doing selection of parents and while switching from one generation to another. Thus the chromosome with highest fitness is always preserved.
- **Crossover** - The crossover class actually handles tournament selection, crossover between parents using the PMX algorithm and then a mutation of the resulting offspring. It selects parents 1 and 2 using the tournament selection algorithm. The information needed to do this is passed to it by the RunGA servlet module.

It should be noted that the JSP pages and other minor servlets are not represented here. They serve as views. The JSPs are scripts and thus are not represented as classes. The controllers for this application are ***RunGA***, ***Chromosomes***, ***Fitness*** and ***Crossover***. They are the classes that will be used to implement all the algorithms mentioned in this chapter and that of figure 2.4 in chapter two.

4.4.4 Database Design

The database structure for this application has almost been described in section 4.3.2 above. However, a formal entity-relationship diagram (ERD) is presented in figure D.1 in **appendix D**. The following good design features were integrated into the database design:

- There were no many-to-many relationships in the database. This slows down database retrieval. Good relationship tables were used to avoid this. The *course_allocations* and *modules_In_Cohort* tables were good examples in the ERD.
- Over -normalisation and under-normalisation of the entities were avoided. The tables used in this project are in either third or fourth normal form. This helps to keep a balance between retrieval speed and data redundancy.
- Finally, the predominantly one-to-many relationships are clearly indicated in the diagram. Good uses of primary and foreign keys were used to ensure referential integrity.

It is believed that figure 4.6 above will help one understand how it was possible to transform the genetic representation of the timetable by the GA to the phenotypic representation which is the actual timetable generated when the GA terminates.

4.5 Summary

In this chapter, an iterative and loosely-coupled algorithmic and software design approach has been used. Eight hard constraints and four soft constraints have been considered. The constructive initialisation will consider some constraint satisfaction while initialising the chromosomes. However, it will still introduce some randomisation. The genetic operators have been designed too to ensure solution evolution. Elitism will also be used to keep good solutions throughout the generation. In this chapter, MVC architecture has been used to design software modules that will be used to implement the GA. A total of seven major classes will be used. In the next chapter, the algorithms designed here will be implemented and tested.

Chapter 5: Implementation and Testing

This chapter details how java programs were used to implement the algorithms and what tests were carried out to ensure that the programs are working as intended. The implementation will discuss the overall program structure and how each part interacts with one another to implement the entire algorithm. Some code fragments will also be included for illustration purposes. The testing part will include structural and non-functional testing, and their relevance to achieving the objective of this research.

5.1 Implementation

Implementation is concerned with writing the actual code that runs the GA in order to generate a feasible timetable. It also involves logical transition from data input, data processing and useful output.

The programming language and tool chosen for this program is Java Enterprise Edition (EE) in Eclipse Framework. Java Server Pages (JSP) were used for the presentation layer while Servlets and core java classes were used to implement the business logic. The bulk of the computation for the GA was also performed in core Java classes where the methods are built as Application programming Interface (API). Those classes can easily be moved around and integrated into other programs. These will be described in the succeeding sub-sections.

5.1.1 Overall Structure

The overall external user structure is presented in **appendix A** as Use Case diagram while the internal structure has also been shown in terms of UML class diagrams in **appendix C**.

The core java classes that handle the GA are grouped in a java package called '***utpsolver***'. This package contains all the modules mentioned in chapter 4.4.3 and more servlet classes that handles input/output operations. The Java Server Pages (JSPs), which handle data presentation and view rendering were all put in a separate folder called ***'inputs***'. Typical files put into this folder include: *addcohort.jsp, addlecturer.jsp, addmodule.jsp, addroom.jsp,courseallocation.jsp,addmoduleToCohort.jsp,* among others

Each of the JSPs typically have a form that collects data and calls a servlet through the action attribute of a HTML form. Example: <form action="/utpsolver/SpecialModuleConstraint" method="post">. This will actually call a servlet called ***SpecialModuleConstraint.class*** located in the ***utpsolver*** package. In order to use any of the java classes or servlets contained in the ***utpsolver*** package, one has to *'import*' either the entire package or the specific class in the package into a new class being defined. For example, to use the ReadInputs class in the Fitness class, the fitness class must have the structure listed in Listing 5.1.

```
package utpsolver;
import utpsolver.ReadInputs;
public class Fitness{
        private ReadInputs read = new ReadInputs();
}
```

Listing 5.1 - Creating a new Java class and importing a module

All java modules used in this program follows the above pattern. Each belong to the package, ***utpsolver***, and possibly imports one or more modules. The *'import'* statement in Java enables one to use the public methods and variables that exist in that module.

5.1.2 Initialisation of Population

All chromosomes or population initialisation were done in the ***Chromosome*** class constructor method. Two separate methods were written to handle either the constructive initialisation or the non-constructive initialisation. Listing 5.2 shows this.

```
package utpsolver;
import java.util.Random;
import utpsolver.ReadInputs;
import utpsolver.Fitness;
public class Chromosomes {
        public   int numChromosomes = 100, fitness=0;
            public   int[][][] chromosomes;
                public Chromosomes(){
                        //this.nonConstructivePopulationInitialisation();
                        this.constructivePopulationInitialisation();
                        fit = new Fitness(chromosomes,numChromosomes);
        }
}
```

Listing 5.2 - Initialisation of population

The commented line (preceded by //), shows that the `nonConstructivePopulationInitialisation()` method will not be called. Thus, it is obvious that both methods were not used simultaneously but alternatively. The entire population was implemented as a 3-D array: `public int[][][] chromosomes`. The initialisation of this array was done thus:

```
chromosomes = new int[numChromosomes][roomCount][timeslot];
```

Where numChromosomes=100, roomCount=23 and timeslot=40.

The `nonConstructivePopulationInitialisation()` method utilises the Rand() function in java.util.Random class to randomly allocate a module to a timeslot and room. No consideration is given to if an allocation violates a constraint or not. Thus, the two other major functions utilised by this method include `this.generateRandomInteger(timeslot)` and `this.generateRandomInteger(roomCount)`. These functions generate a random timeslot and room respectively, into which current module under consideration can be allocated.

The constructivePopulationInitialisation() is more complex and consists of five separate functions. These functions include: `scheduleSpecialModules()`, `schedulePartimeLecturerModules()`, `scheduleRandomly()`, `scheduleForCohortsWithoutClashing()` and `scheduleRemainingModules()`. The idea here is to handle uncommon and extremely stringent constraints (H4 and H6), introduce some random scheduling, handle more hard constraints and simply schedule remaining modules. Each of these functions are written to take care of feasibility, accuracy and diversity.

The data used for both initialisation processes came from inputs into the database in form of course codes, room names, lecturer names and cohorts.

5.1.3 Fitness functions

After initialisation of the entire population (irrespective of what method was used), the fitness of the entire population is determined to decide which one will be selected for reproduction. Fitness is also checked after application of genetic operators. Listing 5.3 below provides the major structure of the code used.

```
public int[] computeFitnessOfEntirePopulation(){
        for(int a=0; a<numChromosome;a++){
                fitnessValues[a] = computeOverallFitnessForAChromosome(a);
                }
        fitnessValues = this.sortFitnessDescending(fitnessValues);
        return fitnessValues;
        }
public int computeOverallFitnessForAChromosome(int chromosome){
        chromosomeFitness=0;
        chromosomeFitness =    this.getOverallRewardsOnHardConstraints(chromosome);
                chromosomeFitness += this.getOverallRewardsOnSoftConstraints(chromosome);
        return chromosomeFitness;
        }
```

Listing 5.3 - Calculation of Fitness for entire population

Listing 5.3 shows methods in the ***Fitness*** module. The instance of Fitness class created in Listing 5.2 shows that the ***Chromosomes*** class passes the entire population (`chromosomes`) and the number (`numChromosomes`) of chromosomes in the population to the ***Fitness*** class. This enables the `computeFitnessOfEntirePopulation()` method to iterate through each chromosome in the population to compute its fitness and store it in the corresponding index in a 1-D array of `fitnessValues[a]`. It accomplishes this by calling a method, `computeOverallFitnessForAChromosome(int chromosome)`, and passing the index of the chromosome whose fitness is to be calculated. This method, in turn, calls two other methods: `getOverallRewardsOnHardConstraints(chromosome)` and `getOverallRewardsOnSoftConstraints(chromosome)`. The sum of the values returned by these later methods forms the total fitness value of an individual chromosome.

5.1.4 Genetic operators

The genetic operators used include selection, crossover and mutation operators. The tournament selection operator was implemented as a method called `doTournamentSelection(int p1,int p2)` in the RunGA servlet class. The integers p1 and p2 are actually generated randomly between 0 and numChromosomes. Thus the method is called twice to select two parents that will be passed to the ***Crossover*** class in order to reproduce offspring.

The crossover operator is implemented as a separate class. The constructor to this class takes the selected parents and a reference to the entire population, that is, `int [][][] chromosomes`. The major methods in this class include: `createParents()`, which extracts the individuals p1 and p2 from the entire population; `createChildOne()`, which implements the first half of PMX crossover in Algorithm A14 and `createChildTwo()`, which implements the complementary part of Algorithm A14. A 3-D array defined by `children [2][roomCount][timeslots]`, holds the results returned from both createChildOne() and createChildTwo().

The mutation operator was implemented as a method which swaps the contents of two genes at different loci of the chromosome passed to it. The full function used for mutation is shown in listing 5.4 below.

```
private void mutateChromosomeByTime(int[][] child){
        int probability = (int)(rooms.length*0.1);
        int room=0,t1=-1,t2=0;
        for(int i=0;i<=probability;i++){
                room = this.generateRandomInteger(rooms.length-1);
                t1 = this.findModuleOccupiedTimeInRoom(room, child);
                int rm2=this.generateRandomInteger(rooms.length-1);
                t2 = this.generateRandomInteger(this.timeslots-1);
                if(t1!=-1 && t1 !=t2){
                        int temp = child[room][t1];
                        child[room][t1]=child[rm2][t2];
                        child[rm2][t2]=temp;
                }
        }
    }
```

Listing 5.4 - Implementing the mutation operator

The method `this.findModuleOccupiedTimeInRoom(room, child)` is used to ensure that a non-empty gene is found first in the randomly chosen room before an actual swap can occur with another location and time in another randomly chosen room, rm2. The offspring are then passed to the fitness function module to ascertain the better offspring chromosome produced.

5.1.5 Run, Terminate GA and display timetable

In order to understand how the entire GA runs, terminates and displays generated timetable, it will be helpful to look a very important code fragment taken from the RunGA servlet class. This is shown in listing 5.5 below.

```
//RUN GA IN THIS LOOP
for(int i=0; i<this.MaximumGeneration;i++){
        this.doCrossover();
        this.getFitnessOfChildren();
        if(overallFitnessOfBestChild > allFitness[this.worstChromosome]){
                cr.replaceChromosome(this.worstChromosome, this.betterChild-1, this.children);
                // Evaluate Entire chromosome only after replacement
                 allFitness = cr.evaluatePopulationFitness();
        }
    }
String timetable=cr.displayGeneratedTimetable(this.bestChromosome) ;// Master timetable
out.println("<br/>" + timetable);
out.println(cr.displayCohortTimetables(this.bestChromosome));//Cohort timetables
```

Listing 5.5 - Run, terminate GA and display Timetables

Listing 5.5 above implements the GAs of both figures 2.4 and 4.4 in the loop starting with for(int i=0; i<this.MaximumGeneration;i++). The default maximum number of generations is 500. The first termination criterion is attained when a chromosome with maximum fitness is obtained as the best chromosome in a generation. This is checked with the condition statement: if(allFitness[this.bestChromosome] =this.maxreward). If this condition becomes true, the loop will be terminated with the break statement.

If the above condition did not terminate the GA, the `this.doCrossover()` method is called. This method actually does tournament selection and instantiates the ***Crossover*** class to do PMX. It also does mutation on the offspring.

After maximum generation is reached, two groups of timetables are displayed. A general master timetable as well as cohort and level-specific timetables are generated. It should be noted that the bestChromosome is not necessarily a feasible one. However, a different method, `getCurrentFitnessMessage()`, in ***Chromosomes*** class is used to get detailed statistics about the level of feasibility of the best chromosome. A typical output is shown in figure 5.2 below.

5.1.6 Optimisation of Algorithm

Some modifications were made in order to optimise the original algorithms and designs in chapter four. Most of them were geared towards achieving non-functional requirements of the project as well as generating insights for future research.

- Code optimisation was done in the ***ReadInputs*** class to speed up the algorithm. This was achieved by creating a 2-D array that holds the database in memory. This abstracts the database away from the application and decreases greatly the amount of time spent in Input/Output operations. This approach reduced the evaluation time of 20 chromosomes in a population *from 15 minutes to 2 seconds*. This optimisation was done using the following methods in this class. All these methods are called only once in the constructor of ***ReadInputs*** class and was never called again until the GA finishes running.
- Also, in order to increase the number of constraints satisfied, the initialisation of the population was done such that modules belonging to same cohort are not placed too close to each other. Thus, input to the GA was such that modules from various levels and cohorts were mixed up and not serially entered into the database. However, the final timetable sorted all these out adequately and regrouped them back accordingly (See Figure 5.2).

5.1.7 Packaging and deployment

This is a web application and thus special packaging into *.jar* or *.exe* is not needed. However, appropriate files need to be kept in the appropriate place for proper functioning of the web application. The files for this project are safely kept on the GitHub repository ats. It can be hosted on Tomcat 6.0 server and MySQL 5.1 or higher. One needs to create a context folder called "utpsolver" on the Tomcat server. The placement of files in the server is done thus:

- **JSP files** - these files apart from those in the "***inputs***" folder should be kept inside the root folder. That is, the "utpsolver" folder. Also style.css should be kept here too.
- **Inputs folder** - all JSP files handling input data and file upload are kept in this folder in this project. Simply copy this folder and its contents to the "utpsolver" root folder.
- **Class files** - all servlets and java API are contained in the /build/classes/utpsolver folder in the project. This contains the class files and not the java files. The class files are needed to run the application. In the tomcat server, create a folder in the path /WEB-INF/classes/utpsolver in the utpsolver root folder. This is where all the *.class* files will be kept for the application
- **web.xml** - This is very important for the servlets to work. This file should be kept at /WEB-INF/web.xml in the Tomcat server.
- **MySQL driver** - The MySQL connector/J is required to connect to the MySQL database. This driver is kept at the WEB-INF/lib/ folder.

With all these are properly put in place, barring any server-specific glitches, one is now ready to test and use the application.

5.2 Testing

Test-driven development (TDD) is greatly advocated in Software Engineering. Though it was not followed in its conventional way in this project, it was integrated implicitly from the very start of the software development. Testing helps one to increase the quality of delivered software. More importantly, it helps one to ensure that both functional and non-functional requirements of the project are met.

5.2.1 Unit Testing

The major idea behind unit testing is to independently and individually scrutinize the smallest testable part of a program. These smallest testable units include statements, assignments, nodes and branches. They get grouped into methods and then into classes. Unit testing was performed using both manual and automated processes. The manual unit tests stems from the incremental development approach adopted. Each method was developed and tested separately. Known variables were passed to the method and the returned variables printed out to confirm its accuracy.

To examine the execution of statements, nodes and branches within a method, an automated tool was used: ***EclEmma Junit Code Coverage for Eclipse***. The detailed test output is shown in **appendix E**. However, code coverage of ***86.9 per cent*** was achieved.

5.2.2 Integration and system Testing

System Testing or black-box testing was conducted on the integrated modules to verify that the system meets the requirement specification. The integration and system testing was performed by calling the code in listing 5.5 via the Graphical User Interface (GUI) shown in figure F.1 in **appendix F**. However, before using the GUI in figure H.1, one is expected to have used the input upload GUI shown in figure 5.1 below. How well the system performed and on what data sets will be discussed in chapter six. However, an output of the system testing is a feasible timetable shown in figure 5.2 below.

It should be noted that some errors were detected and corrected during system testing. They include: (1) Database driver not found, (2) ArrayIndexOutBound exception, (3) a module not scheduled at all (4) a module scheduled but not up to the number of timeslots required for it and (5) request time-out for GAs running too long.

Thus, with system testing it was verified that the GA can generate feasible timetable for all levels and cohorts given adequate resources for the required number of events.

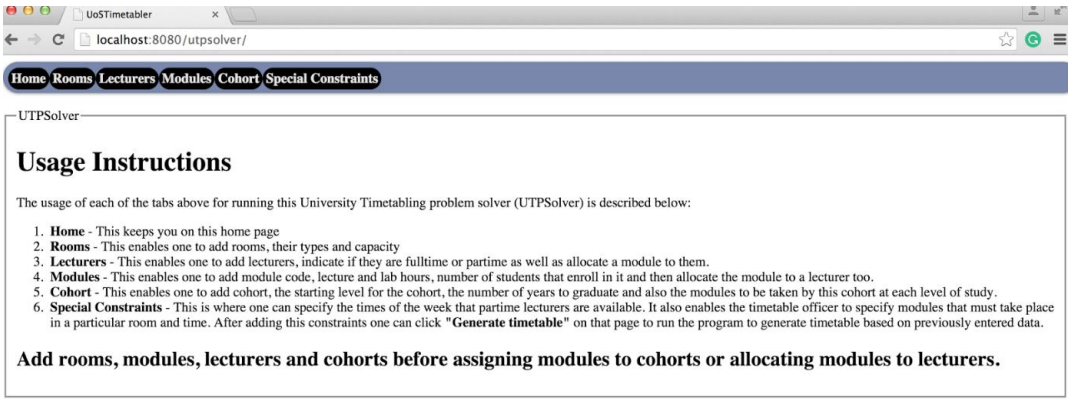
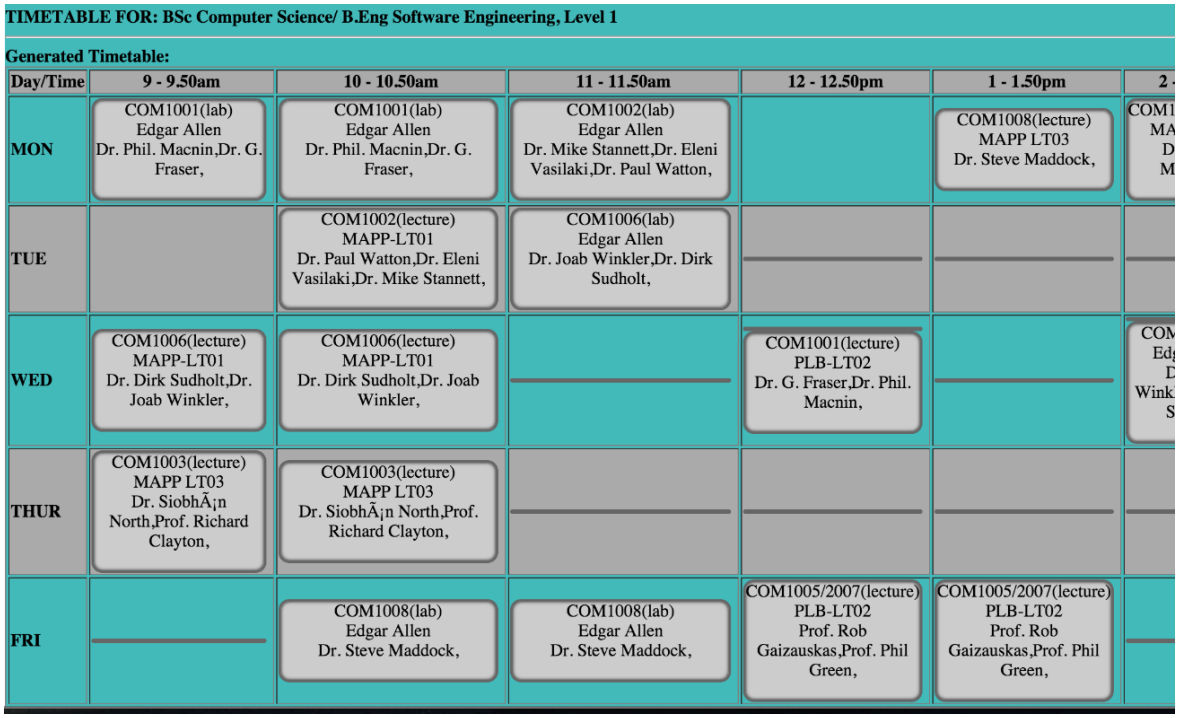


Figure 5.1 - GUI menu for uploading inputs to the system



TIMETABLE FOR: BSc Computer Science/ B.Eng Software Engineering, Level 1

Generated Timetable:

Day/Time	9 - 9.50am	10 - 10.50am	11 - 11.50am	12 - 12.50pm	1 - 1.50pm	2 -
MON	COM1001(lab) Edgar Allen Dr. Phil. Macnin,Dr. G. Fraser,	COM1001(lab) Edgar Allen Dr. Phil. Macnin,Dr. G. Fraser,	COM1002(lab) Edgar Allen Dr. Mike Stannett,Dr. Eleni Vasilaki,Dr. Paul Watton,		COM1008(lecture) MAPP LT03 Dr. Steve Maddock,	COM1 MA D M
TUE		COM1002(lecture) MAPP-LT01 Dr. Paul Watton,Dr. Eleni Vasilaki,Dr. Mike Stannett,	COM1006(lab) Edgar Allen Dr. Joab Winkler,Dr. Dirk Sudholt,			
WED	COM1006(lecture) MAPP-LT01 Dr. Dirk Sudholt,Dr. Joab Winkler,	COM1006(lecture) MAPP-LT01 Dr. Dirk Sudholt,Dr. Joab Winkler,		COM1001(lecture) PLB-LT02 Dr. G. Fraser,Dr. Phil. Macnin,		COM Ed D Wink S
THUR	COM1003(lecture) MAPP LT03 Dr. SiobhÃ¡n North,Prof. Richard Clayton,	COM1003(lecture) MAPP LT03 Dr. SiobhÃ¡n North,Prof. Richard Clayton,				
FRI		COM1008(lab) Edgar Allen Dr. Steve Maddock,	COM1008(lab) Edgar Allen Dr. Steve Maddock,	COM1005/2007(lecture) PLB-LT02 Prof. Rob Gaizauskas,Prof. Phil Green,	COM1005/2007(lecture) PLB-LT02 Prof. Rob Gaizauskas,Prof. Phil Green,	

Figure 5.2 Typical output of the UTPSolver

5.2.3 Non-functional testing

The non-functional requirements set out for this project in section 3.2.3 are speed and ease of use.

- **Speed** - The time taken to run any part of the GA and the entire GA was measured to the millisecond accuracy using the `System.currentTimeMillis()` java function. This helped to optimise various parts of the program and the entire algorithm. However, for the timetabling officer, the point of interest is the overall time it took to obtain a feasible solution. The longest running GA for the hardest case of the problem finished in 164.4 Seconds (2.74 minutes). At least one feasible solution was obtained 8 times in 10 runs for this hardest case (Case 3B in chapter 6). Thus, a worst case scenario presents a running time of 2.74 * 10 = 27.4 minutes. Currently, it takes the timetabling officer from days to weeks to manually construct and optimise timetables. Hence, this is a significant improvement in speed and efficiency.
- **Ease of Use** – This will be discussed under user evaluation in section 7.1.

5.3 Summary

The ***UTPSolver*** was successfully implemented using Java servlets and server pages together with several Java API classes. The major tool used is the eclipse integrated development environment (IDE). The loosely-coupled class design method was used to ensure flexibility. Object-oriented class design was used to encourage maintainability and reusability. Code optimisation was implemented to speed up the algorithm. This was done by abstracting the database away from the application. The final application was deployed in a Tomcat 6.0 local server with MySQL 5.1.

To ensure that implementation conforms to specification, both manual and automated tests were carried out. These include Unit testing, system testing and non-functional testing. More test results are discussed in the next chapter.

Chapter 6: Results and Discussion

This chapter details the empirical results obtained from experiments carried out on the developed algorithm and the resultant software. It will also discuss the observations from the result with reference to quantitative and qualitative results obtained. The Results will be measured in terms of progress rate from one generation to another, number of runs with feasible solutions (accuracy), time taken for the GA to converge (efficiency) and quality of the timetable obtained. Five test cases were used to represent five levels of difficulty.

6.1 Results of Experiments

The major aim of this research is to solve the University Timetabling problem as it occurs in the Department of Computer Science by designing and implementing a suitable GA. These problems were built into 8 hard constraints and 4 soft constraints (see section 3.2.2 in chapter 3). A feasible solution is obtained when 100 percent of the hard constraints are satisfied. Feasible solutions are improved if more soft constraints are satisfied. Hence, the experiments and tests are to ascertain the percentage of constraints satisfied, time taken and number of feasible solutions obtained at each level of difficulty (each test case) per initialisation method for the GA. The experiments were run on MAC OSX 10.9.5 with Intel Core i5 with processor speed of 2.6GHz and 8GB 1600MHz DDR3 RAM memory.

6.1.1 Classification of test data

Five levels of difficulty were tested. They are identified as Cases 1, 2A, 2B, 3A and 3B. The difficulty level increases from case 1 to case 3B. Table 6.1 below presents a breakdown of the test data used for each of the cases.

Table 6.1: Test cases and Test data

		TEST CASES				
		1	**2A**	**2B**	**3A**	**3B**
1	No. of Labs	6	6	6	6	6
2	No. of Lecture Rooms	19	19	19	19	19
3	No. of Lecturers	29	29	29	29	29
4	No. of Lecture-based modules	10	12	12	20	20
5	Number of Lab-based modules	5	7	7	13	13
6	No. Part time Lecturers	0	0	2	0	2
7	No. special room & Time constraints	0	0	3	0	10

- **CASE 1** - Only Level 1 and 2 modules were considered. H4 (Part time lecturers) and H6 (Special room and time constraints) were not put into consideration in calculating percent of hard constraints satisfied.
- **CASE 2A** - Levels 1 to 3 modules were considered but H4 and H6 constraints were not considered.
- **CASE 2B** - It is the same as 2A but there are two part time lecturers and three modules with special room and time requirements.
- **CASE 3A** - It includes all modules from levels 1-3, M.Eng and M.Sc. However, no part-time or special room and time requirements were considered.
- **CASE 3B** - It is the same as 3A but it has up to ten modules with special room and time requirements plus two part time lecturer requirements.

All other constraints, whether soft or hard, were always put into consideration for all calculations. For each of the test cases, the GA was run ***ten times*** on the developed system using each of constructive and non-constructive initialisation methods. The results are presented next.

6.1.2 Result Presentation and Discussions

The results will be presented under the following sub-headings: Progress rate, feasible solutions, level of soft constraint satisfaction and time taken to run the GA.

- ***Progressive Improvement in Solution and Convergence*** - Evolutionary algorithms are meant to improve a candidate solution until it is maximally optimised or it cannot be improved further (convergence). This evolutionary tendency is measured in this research as the rate of increase of fitness value of the best chromosome over the generations per run. These are shown in figures 6.1 for constructive and unconstructive initialisation methods respectively for case 3B.

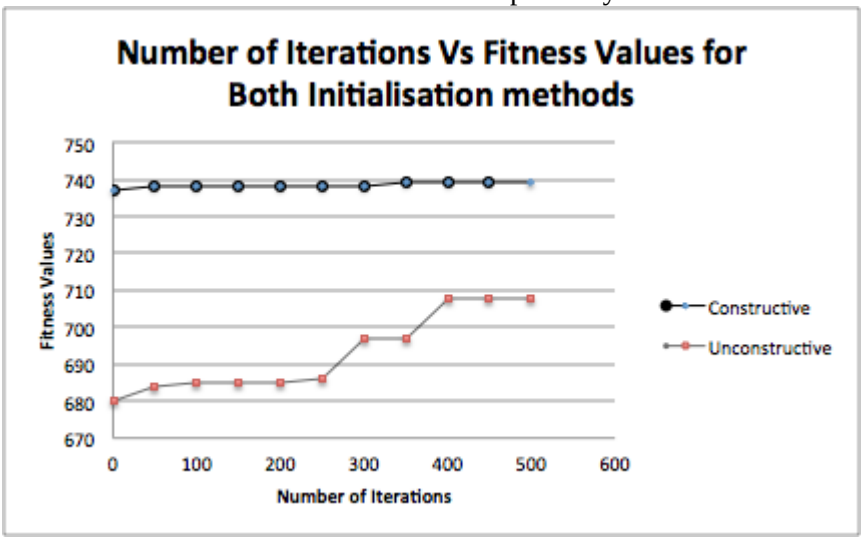


Figure 6.1 - Comparing progress rate over 500 generations

Figure 6.1 shows that there is a progressive increase in fitness of the chromosome from one generation to the next. Constructive initialisation converged faster at 350th generation but had less improvement per 50 generations. This is because it has been improved considerably by the initialisation process to a fitness of 737 (out of 748 maximum). The unconstructive initialisation GA had slower convergence rate at 400th generation but had more improvements per 50 generations and started at a fitness value of 680 (out of 748). Because of the fact that reward-based fitness function was used, unconstructive methods will not necessarily start from a fitness of zero. The availability of required resources and good random matching of events to resources also brought the fitness to a certain value before iterations started.

Figures 6.2 below shows error bars calculated as standard deviations from the mean taken over 10-runs for each of the initialisation methods. It shows convergence to a feasible and improved solution (738.8 fitness value) for the constructive method. This is given that 700 were maximum points for a feasible solution before optimisation. It also shows large error bars for the unconstructive method that are not convergent yet to a feasible solution after 500 iterations because it has average of 626.4 points of fitness.

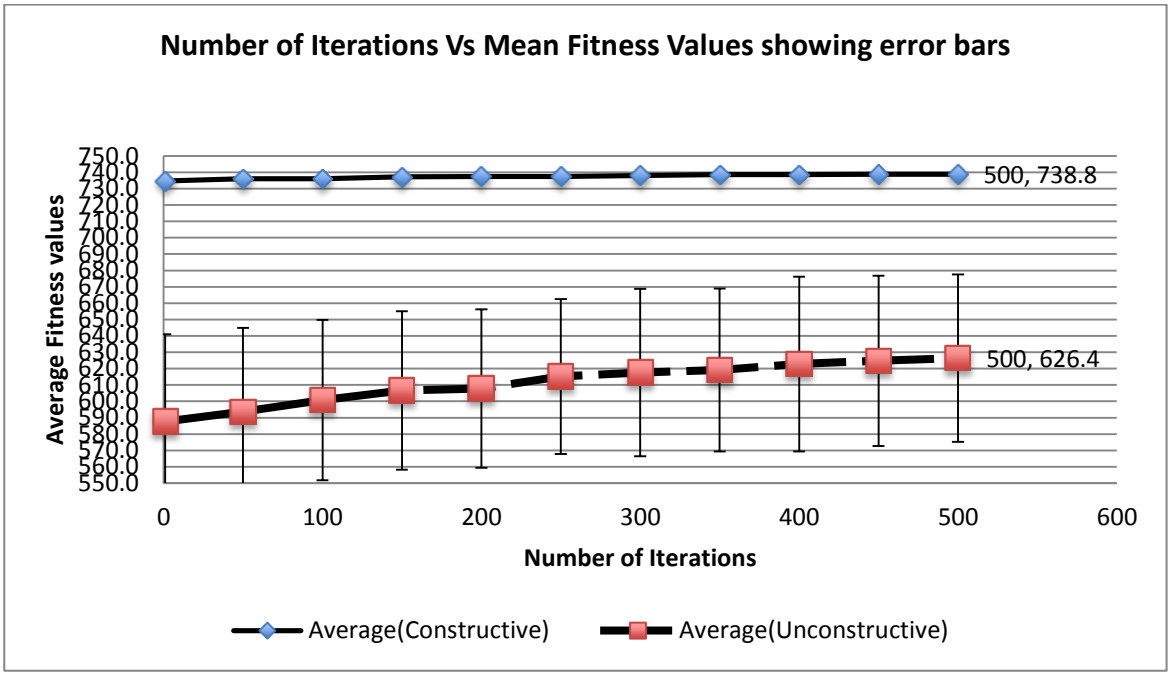


Fig 6.2 convergence rate to a solution

However, figures 6.1 and 6.2 clearly show that both methods have evolved the solution over time and have the capability to break out of local optima. Thus, it is possible that a run for longer generations may produce another convergence point for the unconstructive method. One is limited by processor speed in use to avoid server timeouts and infinite loops leading to crash. GPU-based computers are recommended for larger iterations and for offline applications.

- ***Number of feasible solutions obtained*** - For all the cases, ***no feasible solution was obtained using the non-constructive initialisation method***. Only the constructive initialisation method gave some feasible solutions. The distribution of the number of runs that produced at least one feasible solution per run is shown in figure 6.3 below.

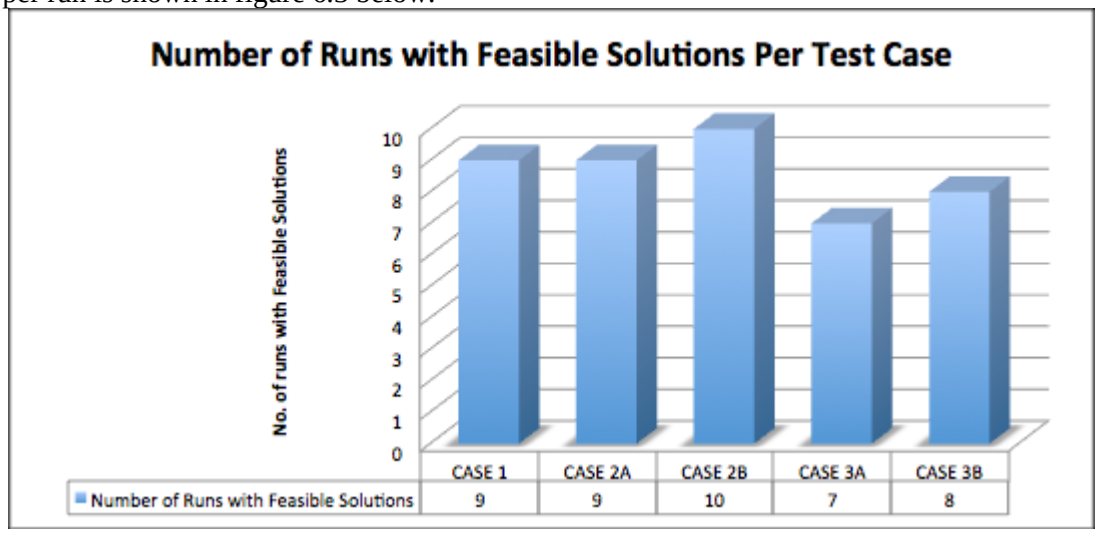


Figure 6.3 - Number of runs with at least one feasible solution

Hence, using constructive initialisation method, at least 7 out of every 10 runs for each of the test cases must produce at least one feasible solution.

Case 3B above is the case that is most typical of the Department of Computer Science here in the University of Sheffield. As shown in figure 6.3 above, 8 out of 10 runs of the algorithm with Case 3B produced feasible solutions. It should be noted that this case contains all the taught modules offered by the department of Computer science in the autumn semester for all levels of study. Thus, with constructive initialisation, feasible solutions can be obtained to solve the timetabling problem.

The rather unexpected pattern of feasible solutions needs some explanations. One would expect the number of feasible solutions to decrease progressively as difficulty level increases. The reason is that reasonable level of randomisation was still introduced in the constructive initialisation process to ensure diversity of the population. Hence, some random genes were introduced before constructively created genes were inserted. Feasibility was not guaranteed for any of the test cases during initialisation but could be possible. Hence, the above result was still stochastic in nature. Most of the infeasible solutions were very close to feasibility (each up to 98 percent feasible).

- **Level of Soft constraint satisfaction** - While feasibility guarantees that a useful solution exists, the level of soft constraint satisfaction determines optimality of a solution. Figure 6.4 compares the average (taken over 10 runs of the GA) optimisation achieved by both initialisation methods on the soft constraints.

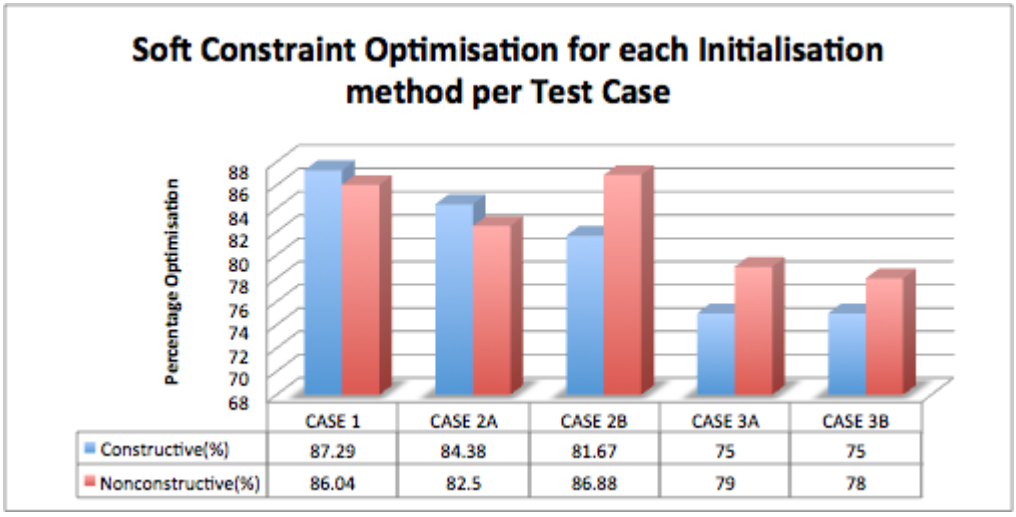


Figure 6.4 - Percentage optimisation obtained per Initialisation method

Generally, it will be seen that Case 1 has the optimum solution for constructive initialisation while case 2B has the most optimum value for unconstructive initialisation. But since, unconstructive initialisation was not feasible, there is no need to discuss it further here.

Another observable trend in figure 6.4 shows that optimisation reduces as problem size and constraint stringency increases. One can see the progressive decrease in optimisation from 87.29% in case 1 to 75% in case 3B. However, the optimisation level of 75% for case 3B, which is of greatest interest in this research, is good and promising level of optimisation.

In comparison of the constructive and unconstructive methods, one can identify some trends. For constructive initialisation, level of optimisation progressively decreases as problem size increases. However, that is not the case for unconstructive initialisation. There was a random trend of intermittent increase and decrease. Thus, constructive initialisation methods are pseudo random while

unconstructive methods are absolutely random. This explains why unconstructive methods are likely to explore more widely and will be helpful in discovering solution to other problems not considered by this research. Thus, unconstructive methods could be quite useful for study purposes.

- **Time taken to run the GA** - This will be divided into two parts: Average time taken to initialise the population and total time taken for the GA to converge or run up to 500 generations. The values used are averages taken over the 10 runs per test case per initialisation method. Figure 6.5 shows the initialisation times in milliseconds (mS) while figure 6.6 shows the average total time taken to run the GA for each test case and for each of the initialisation method.

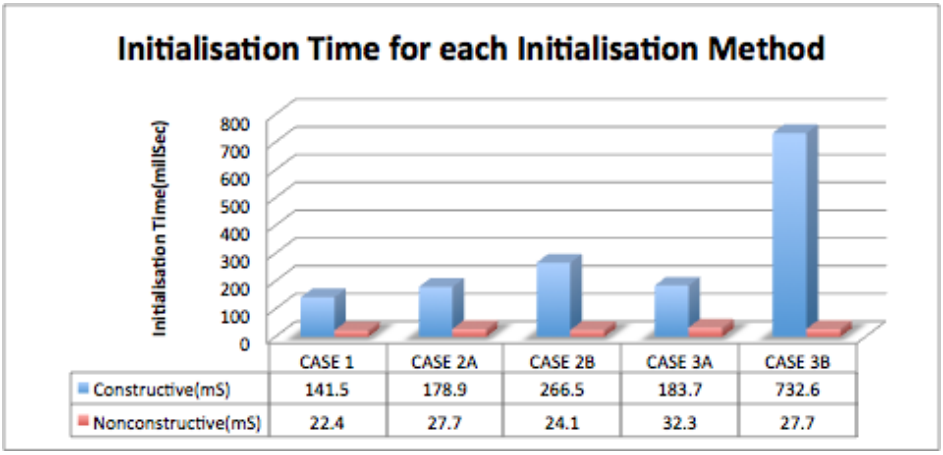


Figure 6.5 - Initialisation Times for Constructive and Unconstructive methods

Case 3B was exceptionally high for the constructive initialisation method. The initialisation time of 732.6 milliseconds is much higher than 266.5 milliseconds which is the second longest. It should be noted that cases 2B and 3B both have special constraint module requirements. However, while case 2B has only three, case 3B has up to ten. Their high initialisation times suggests that evaluation of feasibility due to the special constraint modules added significantly to the initialisation time for the constructive initialisation process. That notwithstanding, the initialisation times are all less than a second and thus are negligible.
The significant difference between the initialisation times for the constructive and unconstructive cases shows that constructive method "thinks" before initial allocation of resources to events. It tries to do at least a right guess and get a good match between events and resources. On the other hand, unconstructive method is purely random and matches are only by chance. Thus, less time is spent on initialisation process for unconstructive methods.

Figure 6.6 presents the overall time taken to run the GA for each of the methods. For all of the cases considered, it takes the constructive GA longer time to execute. One would actually wonder why this much difference in time between the methods owing to the fact that it is only the initialisation process that is different.

In this research, the GA was designed to ensure that much time is not spent on iterations that will not produce a better offspring. Also, according to the profiling that was performed, the GA took longer time to evaluate a chromosome that has better genes than a chromosome that has bad genes. Constructively initialised population has better genes than unconstructively initialised genes. Thus, using the reward-based fitness function a condition is evaluated to know if a gene is fit to satisfy an instance of a constraint. If the gene is fit, the body of the conditional statement is evaluated in order to give a reward to the chromosome containing the gene. Hence, if the gene is a bad one, the body of the conditional is not executed. This implies that the more the bad genes in the chromosomes in a

population, the less time it will take to evaluate the population for iteration. Given the size of this problem instance, the population size and the number of iterations, the collective longer time taken by the constructive method should be clearly understood. Also, the implementation of the GA and its optimisation process was such that if none of the offspring that results from the application of crossover and/or mutation is not better in fitness (considering both hard and soft constraint satisfaction) than the worst parent in current generation, then the entire population will not be evaluated for fitness. A worse offspring cannot replace a bad parent. If no parent is replaced, then there is no need evaluating the entire population in order to rank the fitness of the chromosomes in the population. The GA simply goes back to selection, crossover and mutation. This simply helps to avoid wasting time on temporary local optima until a progress is made. This second reason does not explain the reason for longer constructive GA time but the variation in time spent per run of the GA.

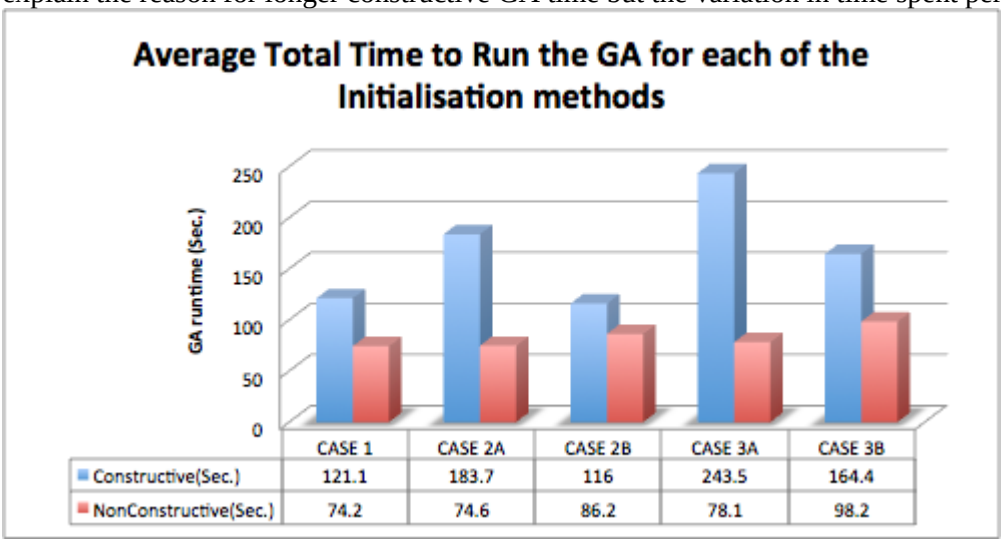


Figure 6.6 - Average time taken to run the GA for both Initialisation methods

The idea behind this is to ensure more time is spent only on runs that are progressing towards a better solution.

From figure 6.5 one can see that case 3A took the longest average time to run for the constructive GA while case 3B took longest time to run for unconstructive initialisation method. The trend in figure 6.6 is not really significant as it is a random and unpredictable process. What is of interest here is the average value taken to run case 3B, which is the problem most concerns this research. Thus, for case 3B and using constructive method, one can conclude that it took an average of 164.4 seconds (2.74 mins) to run the GA and (possibly) obtain a feasible solution.

6.1.3 Qualitative Analysis of "Bad" Result

The required output of this dissertation project is a timetable for four cohorts from Levels 1-3, M.Eng and M.Sc that satisfies the hard constraints in chapter three. However, there are "bad" outputs that are worthy of note. They do not satisfy the functional requirements of this project but are possible solution to a different timetabling problem. We shall discuss only two bad outputs, which were mainly products of the unconstructive initialisation method.

- **Split class groups** - In cases where a class is too large and there is no room to accommodate or personnel to attend to all the members of the group at once, then they might be split into two or more class groups. In this case, fixing lecture for same cohort in different rooms at the same time is no

longer a violation of hard constraint (H3). Thus, the bad output shown in figure 6.7 is no longer a bad one.

3 - 3.50pm	4 - 5pm
COM1006(lecture) MAPP-LT01 Dr. Dirk Sudholt,Dr. Joab Winkler,	COM1006(lecture) MAPP-LT01 Dr. Dirk Sudholt,Dr. Joab Winkler,
COM1003(lecture) MAPP LT07 Dr. SiobhÃ¡n North,Prof. Richard Clayton,	COM1003(lecture) MAPP LT07 Dr. SiobhÃ¡n North,Prof. Richard Clayton,

Figure 6.7 - "Bad results" useful for split group classes

COM1003 and COM1006 are both fixed at the same time but at different venues and by different lecturers. If the number of students in Level 1 offering these modules are large and none of either MAPP-LT01 or MAPP-LT07 can take the students at once, then they can be split into group one and group two. Thus, each group can attend separate lectures at the same time provided the alternative lectures will be repeated another week before the end of the semester. However, digital broadcasting method of delivering lecture will also solve this problem where the facility exists.

- **Alternate week group lecturing** - Most of the modules in Computer Science department are allocated to more than one lecturer. Looking at figure 6.7 above, one would like to know if both Dr. Sudholt and Dr. Winkler are expected to be in class during all weekly lectures throughout the semester or if they are to teach different topics at different weeks of the semester. If the latter is the case, then fixing another lecture for either Dr. Sudholt or Dr. Winkler from 3pm on the same day may not be absolute violation of multiple fixtures for a lecturer (H2). This is true provided both modules do not run for all the 12 lecture weeks in a semester. This is useful also where there are shortage of lecturers. However, this is not suitable for weekly course timetabling as assumed in this research. Figure 6.8 below simultaneously illustrates both split group and alternate week group lecturing. COM1001 lab and lecture were fixed at the same time and the same lecturers and cohorts are attending. Either Dr. Fraser or Dr. McMinn can attend to each class group for different events and at different venues for each half of the semester. However, COM1006 will need to be removed from the 11-11:50am time slot.

Fitness Unit Tests:

H2:Non-Multiple Scheduling for Lecturer: 1116
H3:Non-Multiple Scheduling for Cohort:356
H4:Part-time Lecturer availability observed:1
H5:All modules Scheduled:10
H6:Special Module allocated to room and time NOT violated:0
H7:Classes held in correct room size:9
H8: Classes held in correct room type: 6
S9: Not more than 4-hr consecutive Events for Lecturer:28
S10:Not more than 4-hr consecutive Events for a Cohort: 9
S11: No lecture/Lab fixed on Wednesday afternoon: 3
S12: No lecture/Lab during Launch time: 3 out of 5
Generated Timetable:

Day/Time	9 - 9.50am	10 - 10.50am	11 - 11.50am
MON			
TUE			
WED		COM1001(lab) G12a Dr. G. Fraser,Dr. Phil. Macnin, COM1001(lecture) MAPP-F110 Dr. G. Fraser,Dr. Phil. Macnin,	COM1001(lab) G12a Dr. G. Fraser,Dr. Phil. Macnin, COM1006(lecture) G12c Dr. Dirk Sudholt,Dr. Joab Winkler, COM1001(lecture) MAPP-F110 Dr. G. Fraser,Dr. Phil. Macnin,

Figure 6.8 - "Bad result" that violates H2, H3 and H8.

6.2 Further comments and discussions

For the purpose of replication or extensibility of this research work, readers might find the following comments and findings very useful:

- The algorithm in this research always produces a solution whether the solution is feasible or not. This is because the fitness function used in this research is based on the ***principle of distance-from-feasibility of the infeasible solution***. This has been described as the best principle for designing practical fitness functions [39] to ensure that approximate solutions can always be produced even for highly constrained problems. Thus, even for more complex problem than case 3B above, a solution will be displayed. However, a detailed statistics on the fitness any solution is always provided (See Appendix G).
- Also, a dynamic-based fitness function was used based on the number of hard constraints introduced. Thus the feasible values are 500 (for 6 hard constraints), 600 (for 7 hard constraints) and 700 (for the 8 hard constraints). However, the final system delivered for this research has all the eight constraints and this has a feasibility value of 700.
- Constructive initialisation is advocated as one of the methods of ensuring feasibility as well as utilising the diversity of infeasible solutions of a GA in practical solutions [39] without incurring much CPU time as the use of repair algorithms and discarding and regeneration of infeasible solutions from point of generation. Whereas the method strikes a balance between efficiency and robustness, this method might not be applicable to critical problems where infeasible solution is not tolerated at all.
- The nature of the modules considered in this research is such that most of them run across levels and cohorts. For instance, the module COM2002/3002/6222 is offered across three levels by 4 cohorts. This greatly limits the optimisation of the soft constraints. With these types of module, it was difficult to ensure students do not have more than four hours of lecture at a stretch. Though some of the modules are electives, it is not guaranteed that a student in a cohort will not choose adjacently placed modules on the timetable. So, having achieved no clashing of modules belonging to cohorts and lecturers as part of hard constraint (H2 and H3), it was not often easy to ensure a cohort (or sometimes a lecturer) will not have up to four hours lecture at a stretch. A post GA optimisation process was introduced into the software GUI to solve this problem.

The next chapter provides an evaluation of this project work, including user experience.

Chapter 7: Evaluation

The purpose of this chapter is to evaluate the output of this dissertation work in relation to the objectives of this research and the requirement specifications set out in Chapter three. This will be done through user evaluation of the product and personal objective evaluation of the results on how it has or has not met the objectives and requirements of this dissertation.

7.1 User Evaluation of Product

This system was developed for solving the University Timetabling problem in Department of Computer Science. Hence, the timetabling officer, Zoe C. Fletcher, was asked to use the developed system. She was asked to fill the questionnaire in table 7.1 below. The instruction was: ***"On a scale of 1-10, with 1 being the lowest and 10 being the highest rating, kindly respond to the following questions"***.

Table 7.1: System Usability Scale Survey

S/N	Question	Response
1.	I think I would like to use this system	**4**
2.	I thought the system was easy to use	**7**
3.	I found the functions well integrated into a good work flow	**7**
4.	I think most people would learn to use this system very quickly	**8**
5.	I felt very confident using the system	**8**

The response above and further feedback from the timetabling officer are further discussed below.
The usability of the system is high but the usefulness of the system at this stage is still low. Question 1 above is on usefulness while 2 – 5 is on usability. For the timetabling officer, she would like to see modules from other departments that are taken by computer science students. The design for this research is to handle those modules offered by computer science department. On the average, the usability is 75 percent while the usefulness is 40 percent.

The design of this research was to experiment with the modules from computer science department. Thus, the response from the timetabling officer on usefulness is quite expected. The scope management for the research led to such a decision. Hence, future work would expand on the usefulness of this system by adding more modules and further logistics as required by the timetabling officer.

The user suggested a more specific highlight on where a constraint has been violated on the timetable itself rather than in the GA output data. This would be taken care of in subsequent iterations on the software.

Hence, within the scope of this research, the user believed that the system was easy and intuitive enough to be used.

7.2 Evaluation of Entire dissertation

The purpose of this section is to evaluate to what extent the requirements and the overall objectives of this dissertation have been met. To determine the success of this dissertation work, it is necessary to look back at the requirements set out in chapter three. These requirements logically determine to what extent the overall objective of this research has been accomplished.

7.2.1 Algorithmic Requirements Evaluation

A suitable chromosome representation that has the inherent characteristic of not allowing two events to be fixed at the same venue and time was successfully designed. Also, the chromosome was designed to use more granular time slots unlike the one chosen by researchers in [36], which used single 3-hour time slot per module and does not differentiate a lecture from a laboratory class. The representation method used in this

research adapted the features in [28] but not with multi-valued representation of a gene. A detailed genotypic-phenotypic mapping description was duly presented in Chapter four, section 4.3.

Also, a suitable fitness function was designed using the ***principle of distance-from-feasibility of the infeasible solution,*** which has been advocated by [39]. A suitable algorithm was designed to compute a credible and verifiable fitness value from the hard and soft constraints as listed in chapter four. This fitness function determines the feasibility of a solution.

The fact that results with various feasibility levels were produced shows that the chosen crossover and mutation operators effectively evolved the candidate solutions into feasible and better solutions. The PMX crossover operator and swap mutation operators were chosen and they produced feasible solutions with the chosen crossover and mutation rates.

The most intolerable hard constraint, H1, is never violated in the algorithm. That is, two events can never be scheduled at the same venue at the same time. This was built into the chromosome representation method. No fitness value was actually assigned to it because it will never occur for all cases and for all initialisation methods. It is actually a major requirement of this project that has been successfully achieved.

Hence, one can conclude that the algorithmic requirement of this dissertation as set out in section 3.2.1 was successfully achieved.

7.2.2 Functional Requirements Evaluation

The functional requirements involve satisfying the hard and some soft constraints using the constructive initialisation method. Figure 6.3 (in Chapter 6) shows that all the test cases used produced feasible solutions. That is, 100% satisfaction of hard constraints. Case 3B is the one most related to the actual problem being solved. Figure 6.4 shows an optimisation level of 75% for the constructive initialisation method, which also gave eight feasible solutions out of ten runs. For the soft constraints, it was actually challenging to ensure that all cohorts do not have more than four hours of lectures at a stretch. Satisfaction of the hard constraints inversely affected the satisfaction of the soft constraints. A post-GA optimisation process was designed via Software Engineering of HTML 5 drag-and-drop feature to work around this limitation for practical purposes.

Thus, with eight hard constraints and four soft constraints, one can estimate the percentage achievement of the functional requirements thus:

$$\frac{4}{12} * 75 + \frac{8}{12} * 100 = 25 + 66.67 = 91.67\% \quad \text{..... (6.1)}$$

So, one can say that ***91.67 percent*** of the functional requirements were met by the GA process.
It should be known as well that the post-GA optimisation process was able to help the timetabling officer achieve up to 98% satisfaction of soft constraints. This was not automatically calculated by the software but manually estimated.

7.2.3 Non-functional requirements Evaluation

The BCS requirement for this project was properly met. The ethical and legal issues relating to this dissertation was documented in sect 3.3.2 while the risk assessment and risk register were recorded in section 4.2 and Appendix B respectively.

Other non-functional requirements include speed and ease of use. Simple menu items were used with no second-level sub-menus. This is the simplest and easiest GUI design one can think of. Running the GA used a simple form to collect information about H4 and H6 constraints. Descriptive labels and placeholders were used to clearly indicate what inputs are needed. Apart from data relating to modules, lecturers, rooms, cohorts

and module allocations (which were often collected and entered once per semester), little input is required to run the GA. The usability survey achieved 75% rating from the timetabling officer.

A lot of work was also done to optimise the algorithm in order to increase speed. The problem size of the UTP is large. With 40 time slots and 25 rooms, one has a chromosome of 1000 genes. With 100 individuals and iterating up to 500 generations, one should appreciate the problem size as compared to the test system described in section 6.1. Thus, the average runtime of **2.74 minutes per run of the GA** is a fast one. Comparing this with the time currently spent by the timetabling officer, it is obvious that there is a significant improvement in the time taken to generate a feasible timetable.

7.2.4 Overall Research Evaluation

Referring back to the aims and objectives of this research in section 1.2, one can agree that the mandatory requirements of this dissertation have been met - a justifiable genetic representation method, well defined hard and soft constraints, algorithms for solving the UTP, software GUI to solve a subset of the UTP problem from the Computer Science starting from level one to Masters of Science degree and comparison of constructive and unconstructive GA initialisation methods for solving UTP.

A major deficit of the system is inability to ensure that cohorts do not have more than four hours of consecutive lectures. However, this is a soft constraint in this design and could be made hard constraint if desired. It should also be noted that not all modules are core modules in reality. Thus, not all modules will be offered by all students in a cohort. In this research, all modules were treated as core modules to avoid module clashes of any kind. The consequence is a perceptible longer lecture periods for a cohort. However, the post-GA optimisation feature has greatly provided a leeway for practical purposes.

Hence, considering the scope of this dissertation project, the requirements and objectives of this research have been met. Also, some questions for further research were also raised. These issues that call for further investigation will be listed in the concluding chapter on the next page.

Chapter 8: Conclusion

In this research work, a University timetabling system of great significance and usefulness was produced by designing and implementing a suitable genetic algorithm. It was actually the first of its kind (In the University of Sheffield) in that it automatically puts lots of constraints into consideration before scheduling an event. This was confirmed by the timetabling officer.

However, the success of this research came from various literatures that were reviewed to get a good grasp of GA processes and that of the University timetabling system. This enabled one to get a good understanding of existing best practices and the knowledge that enabled one to conduct this research in a timely and result-oriented manner.

Apart from designing a suitable genetic representation for the University (Departmental-based) Timetabling Problem (UTP), the requirement of this dissertation includes using constructive and unconstructive initialisation GA methods to schedule all the internal modules offered by computer science students from Level 1 to 3, M.Eng and M.Sc. subject to eight hard constraints and four soft constraints. However, it was assumed that no other department competes with the available resources.

An iterative design approach was used for both the algorithmic and software design processes. This was inevitable as the problem in question is a complex one, having complex interaction among the operating variables. The system was also implemented incrementally with Object-Oriented Programming Language, Java. It is a web-based application that runs on Tomcat Server 6.0.

The results show that both constructive and unconstructive methods can evolve solutions but only constructive method was able to produce feasible solutions for all test cases. However, the constructively initialised GA took longer period to run. The unconstructive method did not converge to a feasible solution within 500 generations of which the GA was run but gave insight into possible solutions to other timetabling problems not considered in this work. The constructive method was able to achieve **100% satisfaction of hard constraints** and **75% optimisation of soft constraints**.

Thus, this dissertation was successfully carried out within the assumptions of which it was initiated. The algorithm is flexible enough to accommodate further tests using modules from other departments.

8.1 Suggestions for Future Work

The feedback from the departmental timetabling officer and observations by the researcher in the course of carrying out this project suggest that future work could be done in the following areas:

- Designing a timetabling system that considers modules taken from departments order than that of Computer Science.
- Designing a system that takes into cognisance the fact that other departments also compete for the venues used in this project.
- Designing a semester timetabling system where some modules may have irregular periods for some weeks of the semester.
- A comparative investigation into the speed of reward-based and penalty-based fitness functions for GAs.
- Design of appropriate genetic operators that could allow unconstructive initialisation GAs to converge to a solution or the impossibility of its convergence to a solution - GPU-based computers could be used.

References

[1] N.S. Sadaf and Y. Shengxiang (2009). "***A guided search Genetic Algorithm for the University Course timetabling problem***" In Multidisciplinary International Conference on Scheduling: Theory and applications 10-12 August 2009, Dublin Ireland. [Online] www.cs.le.ac.uk/people/syang/Papers/MISTA09.pdf *retrieved* 4th February 2015.

[2] S. Even, A. Itai and A. Shamir (1976). "***On the complexity of timetable and multi-commodity flow problems***" In SIAM Journal on Computing, 5(4) pp 691 - 703.

[3] D. Floreano and C. Mattiussi (2008). ***Bio-Inspired Artificial Intelligence: theories, methods and technologies***. MIT Press Cambridge, USA.

[4] M. Mitchel (1996). ***An Introduction to Genetic Algorithms.*** MIT Press, Cambridge.

[5] S. Jevon (2014). ***Speed and Diversity in Cellular Evolutionary Algorithm***. MSc Dissertation, Department of Computer Science, University of Sheffield.

[6] R. Raghavjee and N. Pilley (2008) "***An Application of Genetic Algorithms to the School Timetabling Problem***" In SAICSIT Conference Proceeding, 6-8 October, 2008. pp. 193 - 199. [Online]: www.titan.cs.unp.ac.za/~nelishiap/uploads/45.pdf *retrieved* 16th March, 2015.

[7] J. H. Kingston (2006). "***Hierarchical Timetable Construction***" In Edmund K. Burke & Hana Rudova (Eds.). Practice and Theory of Automated Timetablin***g***. Proceedings of the 6th International Conference on the practice and Theory of Automated Timetabling, 30th August - 1st September 2006. pp 196 -208.

[8] C. Darwin (1859). ***On the Origin of Species***. London: John Murray.

[9] T. Bäck (1996). ***Evolutionary Algorithms in Theory and Practice***. ISBN: 0195099710. Oxford University Press, Inc., 7-9, 59, 66, 107.

[10] A.E. Eiben (2002). ***Introduction to Evolutionary Computing II***. Free University, Amsterdam. [Online]: http://www.informatica.uniroma2.it/upload/2009/IA/Intro-to-EC-2.pdf

[11] http://en.wikipedia.org/wiki/Allele [Online] *retrieved* 3rd March, 2015.

[12] S.A. Bashir (2014). ***Developing a Java-based Genetic Algorithm to solve the Travelling Salesman's Problem***. MSc Dissertation, Department of Computer Science, University of Sheffield.

[13] Virtual Genetics Education Centre (2014). ***DNA, genes and Chromosomes***. The University of Leicester. [Online]: http://www2.le.ac.uk/departments/genetics/vgec/schoolscolleges/topics/dna-genes-chromosomes *retrieved* 3rd March 2015.

[14] D.J Weatheral (2001). "***Genotype - Phenotype Relationships****"* In Encyclopaedia of Life Sciences . Nature Publishing Group.

[15] http://en.wikipedia.org/wiki/Generation retrieved 3rd March, 2015.

[16] D.W. Dayer (2010). ***Evolutionary Computation in Java: A practical guide to the watchmaker Framework***. [Online]: http://watchmaker.uncommons.org/manual/index.html *retrieved* 3rd March 2015.

[17] http://commons.wikimedia.org/wiki/File:Gene_structure.svg [online] retrieved 5th March 2015.

[18] V.M. Tom. ***Genetic Algorithm***. Department of Civil Engineering, Indian Institute of Technology Bombay, Mumbai-400076. [Online]: http://www.civil.iitb.ac.in/tvm/2701_dga/2701-ga-notes/gadoc/gadoc.html *retrieved* 9th March 2015.

[19] M. Golub(1996) . "***An Implementation of Binary and Floating point Chromosome representation in Genetic Algorithm***". Faculty of Electrical Engineering and Computing, University of Zagreb Croatia. [Online] : zemeris.fer.hr/~golub/clanci/iti96.pdf *retrieved* 14th March, 2015.

[20] S. Gotshall and B. Rylander(2001) "***Optimal Population Size and the Genetic Algorithm***". University of Portland, USA. [Online]: citeseerx.ist.psu.edu/viewdoc/download? [online] retrieved 15th March, 2015.

[21] K. Vekaria and C. Clack (1998). "***Selective Crossover in Genetic Algorithms : An Empirical Study***" In A.E. Eiben et al (eds.) Proceedings of the International Conference on Parallel Problem Solving from Nature, Berlin.LNCS 1498, pp. 438-447, [online] http://www0.cs.ucl.ac.uk/staff/C.Clack/PPSN98.pdf retrieved 15th March, 2015.

[22] http://www.wardsystems.com/manuals/genehunter/mutation_of_enumerated_chromosomes.htm [online] *retrieved* 15th March 2015.

[23] S. Mohaghghegh (2000). "***Virtual Intelligence and its Applications in Petroleum Engineering: Evolutionary Computing***" In Journal of Petroleum Technology, October 2000. url: http://www.intelligentsolutionsinc.com/Technology/AITheory/AI2-GA.shtml *retrieved* 15th March, 2015.

[24] http://en.wikipedia.org/wiki/Fitness_function [Online] retrieved 15th March, 2015.

[25] P. Tormos et al. (2008). "***A Genetic Algorithm for Railway Scheduling Problems***" In Studies in Computational Intelligence (SCI) 128, pp. 255–276.

[26] G. Fraser and A. Acuri (2013) "***A Large Scale Evaluation of Automated Unit Test Generation Using EvoSuite***". [Online]: http://www.evosuite.org/wp-content/papercite-data/pdf/tosem_evaluation.pdf , *retrieved* on 19th March, 2015.

[27] Mehdi et al (2012). "***Solving University Course Timetabling Problem using Genetic Algorithm***" In 2nd World Conference on Information Technology. AWERProcedia Information Technology and Computer Science. Vol 1 (2012). pp 1033 - 1040.

[28] E. Yu and K. Sung (2002). ***"A genetic Algorithm for weekly courses timetabling problem"*** In International transactions in Operational Research, 9 (2001), pp 703 - 717.

[29] M.S Abubakar et al (2006). "***Maintaining diversity for Genetic Algorithm: A case study of timetabling problem***" In Jurnal Teknologi 44 (D) June, 2006, pp.123 - 130.

[30] A. Cockburn (2001). ***Writing effective Use Cases: Humans and Technology***. Addison Wesley.

[31] K. Beck et al, ***The agile Manifesto***. [Online] from www.agilemanifesto.org *on* 11th April, 2015.

[32] W. Rupert, B. Edmund and E. Dave (1995). ***A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems***. Computer Science Technical Report No. NOTTCS-TR-1995-8.

[33] S.A. Oyebanjo (2013). ***Development of a University Timetabling Automation System***. B.Sc Project, Department of Computer and Information Science, Covenant University, Nigeria.

[34] http://www.flexiblesoftwaresolutions.co.uk/articles/software/art_of_developing_process, [online] *retrieved* 25th May, 2015

[35] S. Martins, C. Jessica, P. Ignacio, and B. Nelida (2004) "***On Stopping Criteria for Genetic Algorithms***" In Ana L. C. Bazzan • Sofiane Labidi (Eds.) Advances in Artificial Intelligence. SBIA 2004, LNAI 3171 .pp 404 -413.

[36] W. Chinnasri, S. Krootjohn, and N. Sureerattanan (2012) "***Performance comparison of Genetic Algorithm's crossover operators on University Course Timetabling Problem***" In Proceedings of 8th International Conference on Computing and Information Management (ICCM), 24th -26h April, 2012 in South Korea.

[37] http://www. wiki.org/Genetic_algorithm [online] *retrieved* 6th July, 2015.

[38] S. Caballé , J. Ortega et al (2014) ***"A Presentation Framework to Simplify the Development of Java EE Application Thin Clients"*** In 2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems. Birmingham, UK, 2-4 July 2014. pp. 421 - 426.

[39] P. Vassilios, K. Spyros, and B. Anastasios. (1998) ***"Varying Fitness Functions in Genetic Algorithm Constrained Optimization: The Cutting Stock and Unit Commitment Problems"*** In IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics, Vol. 28, No. 5, October, 1998. pp. 629 - 640.

[40] D.A Coley (1999). ***An Introduction to Genetic Algorithms for Scientists and Engineers*.** Singapore, World Scientific Publishing Co., Pte, Ltd.

APPENDIX A: Use Case Diagram for the UTP Software

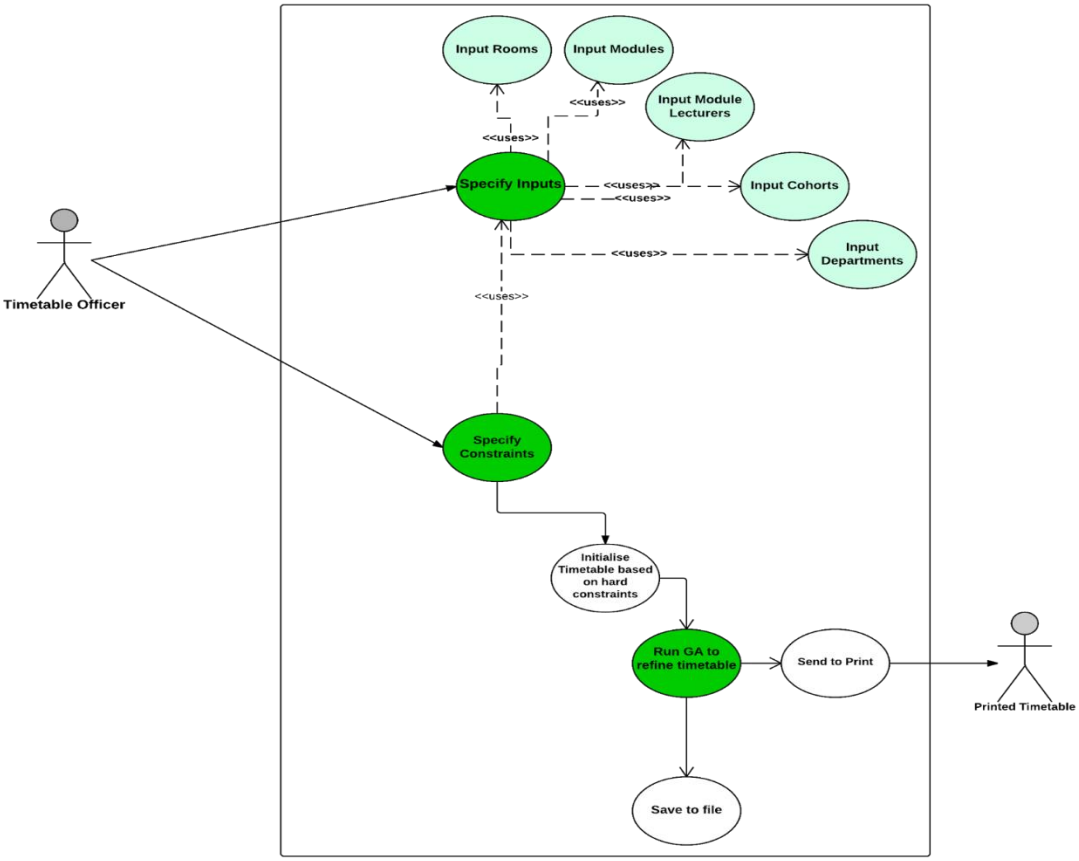


Figure A.1 – Use Case Diagram

The timetabling officer will have input the rooms, modules lecturers and programmes offered by the department. He/she will also have to enter lecturer, room and student constraints before the GA is executed to produce a printable timetable.

Appendix B: Risk Register for UTP System

Table B.1: Risk Register for UTP System

Rank	Risk	Likelihood	Impact	Exposure	Action
1	Unclear requirements may lead to re-working and late delivery	4	4	16	-regular refinement -prototyping before development. -Incremental delivery
2	Uncertainty about GUI and parameters required to run the GA	3	3	9	-Build a mockup -Incremental delivery

3	Supervisor unavailability may delay progress	2	4	8	- Early visit. -Use of google drive for documents -Sending emails
4	Misunderstanding Requirement specification	2	4	8	-Prototyping -Incremental delivery -frequent feedback
5	Computer crash may lead to document and code loss	2	4	8	-Version control system -Google drive for docs.
6	Identifying correct GA representation for problem instance may delay development	2	3	6	-wide literature review -Prototyping
7	Hardware capacity for large problem instance	3	2	6	-Reduce problem size -Apply for special computer
8	Appropriate choice of language to handle complex data structure	1	3	3	-Wide literature review - Quick prototyping in different languages

Appendix C: UML Class Diagram for UTPSolver Software

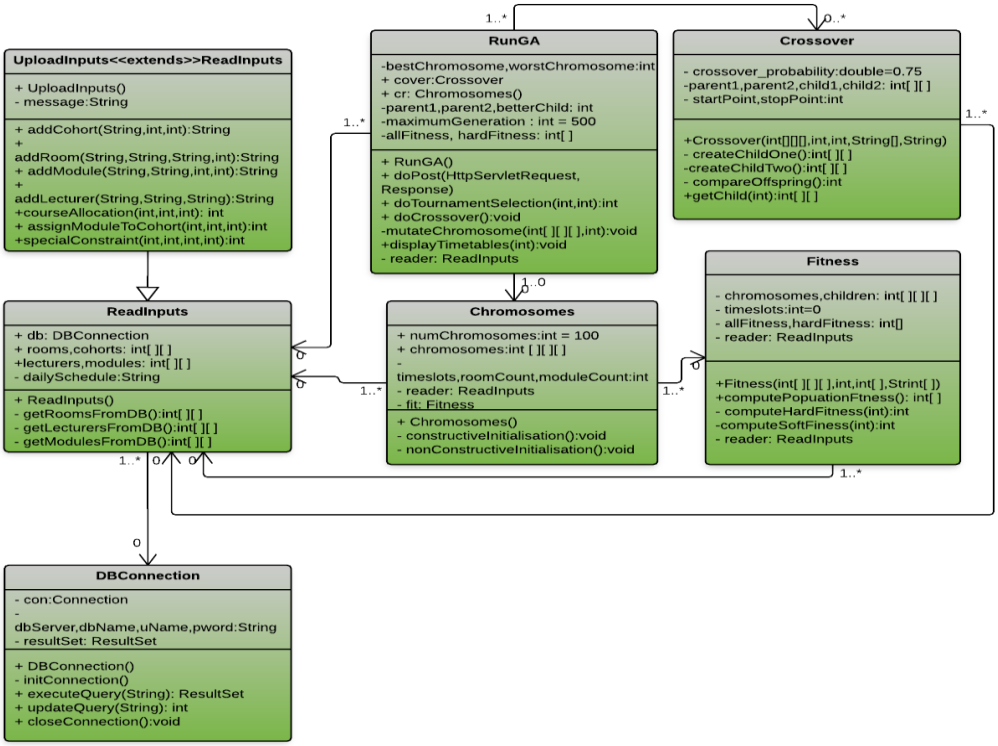


Fig C.1: Elaborate Class Diagram

Appendix D: Database Entity-Relationship diagram

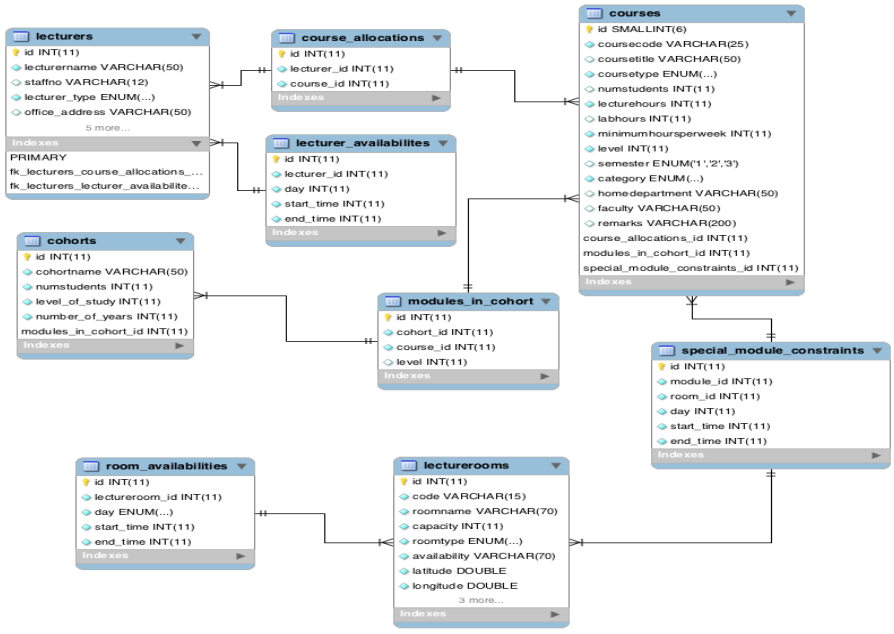


Figure D.1: Entity-Relationship diagram (ERD) for the UTPSolver database

Appendix E: Unit Testing with Eclemma JUnit

This was used to verify and visualise the execution of parts of the code and what variables they contain at a given moment. An example output is shown in Figure G.1 below.

```
    //Check if class held in appropriate room type for a gene event in a chromosome
    private boolean checkIfClassIsHeldInCorrectRoomType(int chromosome,int room, int module){
        boolean isCorrect = false;
        String roomType = read.getRoomType(room);
        String moduleType = read.getModuleType(module);

        if(roomType.equals(moduleType)){
            isCorrect = true;
        }
        else{
            isCorrect = false;
        }
        return isCorrect;

    }

    //Check if class held in appropriate room Size for a gene event in a chromosome
    private boolean checkIfClassHeldInCorrectRoomSize(int chromosome,int room,int module){
        int tolerance = -10; // A tolerance of 10 is when module size is more than room Capacity
        boolean isCorrect = false;
        int roomSize = read.getRoomCapacity(room);
        int moduleSize = read.getModuleSize(module);
        if(moduleSize <= roomSize || (roomSize - moduleSize)  >= tolerance)
            isCorrect=true;
```

Figure E.1 - Eclemma JUnit code coverage

The green bands show the executed codes, yellow band shows conditionals with one branch executed while the red band shows part of the code that has never been executed. Also, breakpoints, toggle points and watchpoints in Eclipse were also used to perform unit testing. For the core modules relating to the running of the GA (ReadInputs,Chromosomes,Fitness, Crossover and RunGA), a code coverage of ***86.9 per cent*** was achieved.

APPENDIX F: Specifying constraints for System Testing

Clicking the "Generate Timetable" button runs the entire GA by calling almost all the methods from the modules previously mentioned. This singular action calls the RunGA servlet whose core code component was shown in Listing 5.5. This successfully ran the GA within reasonable time and produced a good timetable most of the time. All modules previously designed will be called by the codes in listing 5.5.

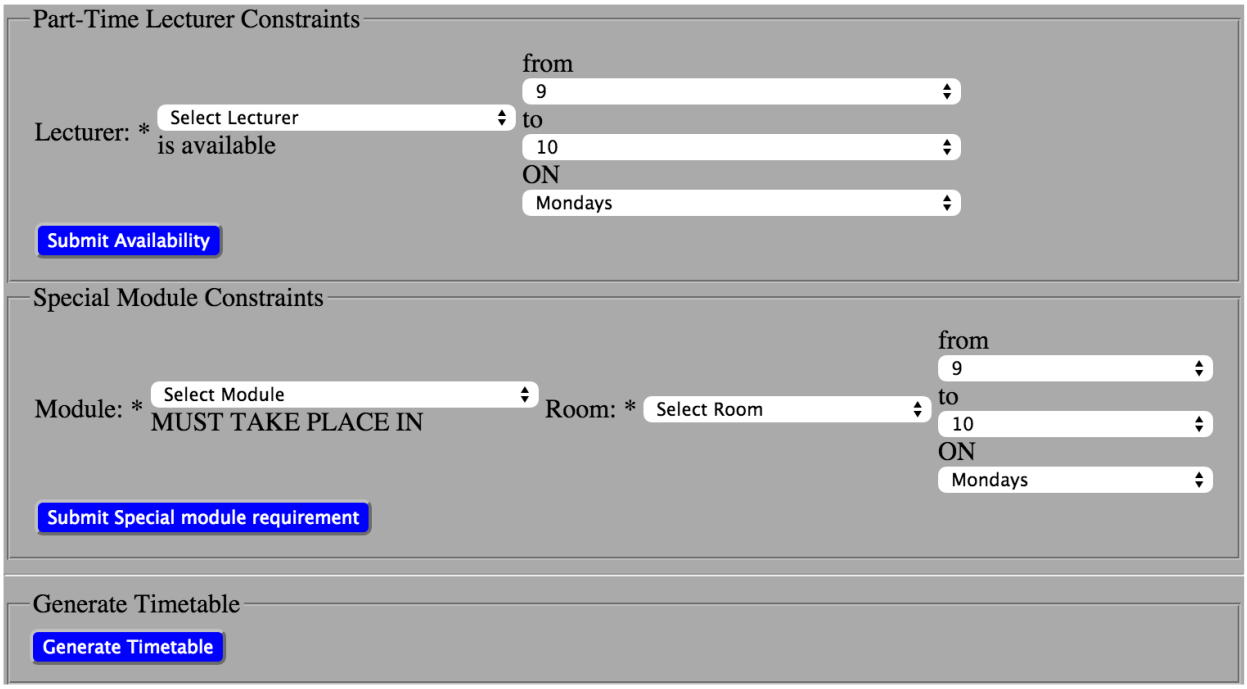


Figure F.1: Integration/System testing GUI

Appendix G: Statistics for best Chromosome

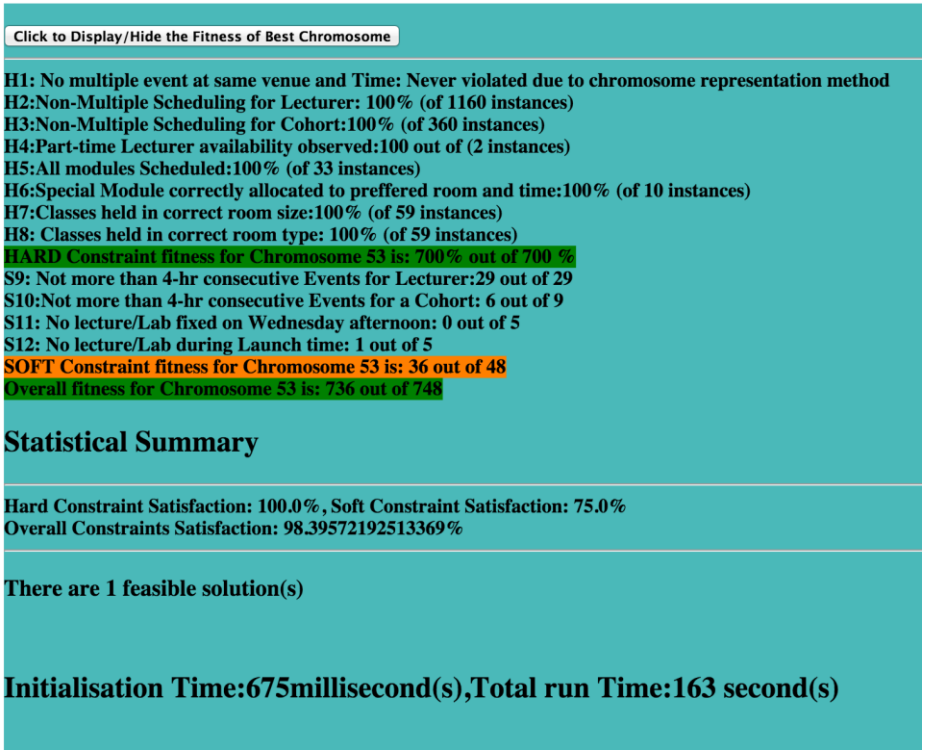


Figure G.1 – Data for the fitness of a typical best solution