2017

Genetic Algorithm For University Course Timetabling Problem

Achini Kumari Herath
University of Mississippi

GENETIC ALGORITHM FOR UNIVERSITY COURSE TIMETABLING PROBLEM

A Thesis
presented in partial fulfillment of requirements
for the degree of Masters of Science
in the Department of Computer and Information Science
The University of Mississippi

by
Achini Kumari Herath
May 2017

ABSTRACT

Creating timetables for institutes which deal with transport, sport, workforce, courses, examination schedules, and healthcare scheduling is a complex problem. It is difficult and time consuming to solve due to many constraints. Depending on whether the constraints are essential or desirable they are categorized as ‘hard’ and ‘soft’, respectively. Two types of timetables, namely, course and examination are designed for academic institutes. A feasible course timetable could be described as a plan for the movement of students and staff from one classroom to another, without conflicts. Being an NP-complete problem, many attempts have been made using varying computational methods to obtain optimal solutions to the timetabling problem. Genetic algorithms, based on Darwin's theory of evolution is one such method. The aim of this study is to optimize a general university course scheduling process based on genetic algorithms using some defined constraints.

DEDICATION

I dedicate this thesis to my family. Their patience and inspiration never ceases to amaze me.

ACKNOWLEDGEMENTS

Foremost, I would like to express my thanks and sincere gratitude to my advisor Dr. Conrad Cunningham for the unwavering support and encouragement during my Masters study. His patience, availability and vast knowledge in computing helped me during my graduate study and research.

I would also like to thank the rest of my thesis committee, Dr. Dawn Wilkins and Dr. Yixin Chen, whom I've also been blessed with the opportunity to learn and get advice from. Their vast knowledge in AI and machine learning have been very helpful in my studies.

I thank my department chair, Dr. Dawn Wilkins and the rest of our amazing faculty and staff. Thank you for your support and the wonderful job at the department.

Finally, to my family, thank you for all your advice, sacrifice, encouragement and support throughout my life. I couldn't ask for a better family.

TABLE OF CONTENTS

ABSTRACT ……..ii

DEDICATION ……..iii

ACKNOWLEDGEMENTS …… iv

LIST OF FIGURES ……v

INTRODUCTION……1

1.1 Timetabling……1

1.2 Aim of Thesis ……3

1.3 Methods for automated timetabling……3

GENETIC ALGORITHM :8

2.1 Fundamentals of Genetic Algorithm (GA) ……8

2.2 How Genetic Algorithms Work ……10

2.3 Initial population of chromosomes ……11

2.4 Suitability of chromosomes to mate ……11

2.5 Selection to the mating pool……12

2.6 Methods of change in reproduction……16

2.7 Advantages of genetic algorithm ……20

2.8 Disadvantages of genetic algorithm ……21

DESIGN AND IMPLEMENTATION STRATEGY : 22

3.1 Encoding ……22

3.2 Initialization 23

EXPERIMENTS AND RESULTS : 26

4.1 Architectural specifications 26

4.2 Performance Analysis 27

CONCLUSION : 31

BIBLIOGRAPHY : 32

VITA : 37

LIST OF FIGURES

2.2 Genetic Algorithm. 10
2.4 Suitability of chromosomes to mate. 12
2.5.1 Five member population fitness scores. 13
2.5.1 The normalized fitness scores and the percentage values. 13
2.5.2 Selection based on the wheel of fortune. 13
2.5.4 Tournament mechanism. 15
2.6.1 Cross-over 17
2.6.1 One point cross-over 17
2.6.1 Selecting a random point 17
2.6.1 Coin-flipping approach 17
2.6.2 Mutation 18
2.6.3 Program trees in Genetic Algorithms 19
3.1 Encoding the chromosome 23
4.2 Performance of Course Scheduler 27
4.2 Performance of population size increase 28
4.2 Performance of mutation rate vs number of clashes 29
4.2 Performance of mutation rate vs fitness value 30

CHAPTER 1

INTRODUCTION

1.1 Timetabling

Scheduling or timetabling is the process of allocating time for planned activities in an orderly manner to bring about a satisfactory result, free of constraints [1]. Some of the examples include transport, sport, workforce, course, and examination scheduling. In an academic institution for instance, two types of timetables can be recognized. They are the course and the examination schedules. Courses should be assigned to specific time slots for five working days of the week taking into consideration the specific classrooms suitable for the respective courses and the registered number of students [2]. A feasible timetable in an academic institute is thus a description of the movement of students and staff from one classroom to the next one, the location of the classrooms, and the time slots. In creating a timetable an institute has to face many constraints which could be defined as 'hard' and 'soft' depending on whether they are essential or desirable [3].

1.1.1 Timetabling: Hard constraints

Hard constraints concern issues that are physically impossible – such as a teacher or student being in two places at the same time. Similarly, two teachers are not permitted to

teach two separate courses to the same group of students in a given time slot. Further, allocations of two or more classrooms are not made for the same course for a given group of students. Thus, it is necessary that, neither the staff nor the students can be in more than one place at a given time. Further, all the necessary resources such as the staff, rooms etc. should be available for each time slot. All the available courses must be entered in the timetable with flexibility for multi-period sessions if a teacher so desire and to repeat sessions for small groups when all the registered students cannot participate simultaneously (such as lab work) [2]. Depending on the nature of meetings allocating lecture theaters of varying sizes as well as specific laboratory spaces are added problems [3].

1.1.2 Timetabling: Soft constraints

The soft constraints include the assignment of "hard courses" in time slots in the morning sessions when students are able to pay attention to such subjects. When higher number of students occupy large classrooms, changing rooms after every lecture is avoided unless small groups have to move to small spaces such as laboratories. Staff preferences such as teaching at times and classrooms of choice are additional constraints [3].

1.1.3 Timetabling as an NP problem

Although manual scheduling is time consuming and inaccurate, small universities adapt to generate their schedules. As the complexity of university increases it to become necessary to adopt computer methods to ease the task of timetabling [4][5]. It is to be noted that when the student population with diverse interests and requirements increases

and the teaching programs get complicated with the growth of university, the number of constraints grows resulting in an exponential rise in the computational time, making it an NP-complete operation [2],[6],[9],[10].

1.2 Aim of Thesis

The aim of this work is to demonstrate the usefulness of genetic algorithm usage to obtain optimal solutions in general timetable scheduling. Although much commercial scheduling software is available, its lack of generality rarely meets the demands of various institutions. Therefore, the requirement of specific coding as per respective universities is the biggest hurdle to overcome.

1.3 Methods for automated timetabling

In constructing a timetable, it is useful to have knowledge about the previous timetable, especially about the weaknesses associated with it. If drastic changes are introduced a new timetable has to be computed. Timetabling could be treated as a feasibility problem when the requirement is a single feasible solution. However, an optimal solution is sought according to an objective function when it is treated as an optimality problem. Many techniques have been developed to generate acceptable timetables [8], [11].

1.3.1 Some methods for automated timetabling

One of the most studied NP-hard problems is the “graph coloring problem”, and as a result has many applications that are useful in scheduling problems, specifically, timetabling.

1.3.1.1 Graph coloring algorithms.

Graph coloring [12] is the assignment of a minimum number of colors (timeslots) to the vertices (nodes) to represent events (courses) of a graph in such a manner that two vertices are connected with different colors to avoid time conflicts. The edges (constraints) of the graph represent the conflict between events. With the increase in size (increase in the number nodes) of a graph, getting an optimal solution becomes difficult as it needs exponential time, making it a NP-hard problem [3],[13],[14]. The method involves the coloring of the nodes of the graph properly and utilizing it to create a conflict free timetable[15].

1.3.1.2 Mathematical programming algorithms.

Many mathematical models have been described in the literature to solve timetabling problems in various universities [16]-[24]. All the models have utilized commercial solvers.

1.3.1.3 Use of database management systems.

Database management systems is the use of computer database support systems for the development of timetables [25].

1.3.1.4 Genetic algorithms.

A genetic algorithm is a heuristic search method used to find solutions for optimization problems. It is based on the Darwinian theory of evolution. This technique involves the ultimate selection of the fittest (best timetable) from a randomly created population (chromosomes) of solutions for the timetabling problem where each individual (chromosome) represents a timetable. The optimality (perfection) of a chromosome is evaluated by a fitness function based on hard and soft constraints [26],[27]. Genetic algorithms begin by creating a random population of timetables followed by their evaluation according to defined criteria to select parents (timetables) for the next generation which is expected to produce better timetables by way of crossovers and mutations. The process is repeated until a satisfactory solution is reached [27].

1.3.1.5 Logic programming approaches.

Logic programming is a method used to find solutions to combinatorial optimization problems [28].

Constraint logic programming is a declarative programming paradigm mainly suited for encoding combinatorial minimization problems. It is the natural union of the two declarative paradigms known as *Constraint Solving* and *Logic Programming*.

1.3.1.6 Timetabling: genetic algorithm approach

Scheduling a Timetable for College.

As mentioned earlier, scheduling classes for a college timetable is often encountered with constraints (hard and soft) due to diversity as compared to a school timetable where the requirements are highly limited [29]. The problems associated with hard constraints need to be resolved to produce a functional solution. To optimize the performance of the scheduling it

is important to address the issues linked to soft constraints. However, a careful approach should be formulated without compromising the solutions to hard constraints to minimize serious disruptions to the system. As such, a straightforward scheduling system as for a small school system cannot be successfully applied to a complex organization without turning to a different approach for quick and optimum results [30].

There may be conflicts between multiple soft constraints and as a result a tradeoff will need to be reached between them [31]. As an example, a class might have 12 students, and as such the soft constraint will assign a suitable classroom that can accommodate the students. However, there may be a classroom that the professor prefers which can hold up to 45 students. The class scheduler will hopefully find a preferred configuration as professor preference is considered in the soft constraints.

The Problem:

The class-scheduling problem will be based on the following data.

- Available Professors
- Available Rooms
- Timeslots
- Student groups

A college timetable is different from a grade school timetable in that there may be free time periods on a college student's schedule. This depends on the number of courses taken by the student.

Each class will be assigned a timeslot, a professor, a room, and a student group by the class scheduler. The total number of classes that need to be scheduled can be obtained

by summing the number of student groups multiplied by the number of modules each student group is enrolled in.

For each class scheduled by this application the following hard constraints will be considered.

- Classes can only be scheduled in free classrooms.
- A professor can only teach one class at any one time.
- Classrooms must be big enough to accommodate the student group.

When encoding the class schedule, certain class properties are needed. They are: the timeslot the class is scheduled for, the professor teaching the class, and the classroom required for the class.

Room: RoomID, Room Number, Room Capacity

Time slot: TimeslotID, Timeslot

Professor: ProfessorID, Professor Name.

Course/Module: Course/ModuleID, Course/Module Code (Sections), Course/Module, Professor ID.

Group: GroupID, Group Size, ModuleID

CClass: Represents the class taken by the student. ClassID, GroupID, ModuleID, ProfessorID, TimeslotID, RoomID.

The timetable will encapsulate all these objects. This class will be able to differentiate how different constraints interact with each other. Also, it will be responsible for parsing for example a chromosome in genetic algorithm to create a candidate timetable to be evaluated and scored.

CHAPTER 2

GENETIC ALGORITHMS

Genetic algorithms are metaheuristic methods used to solve computational problems which require large search areas for possible solutions. They very often depend on adaptive systems to perform well in changing environments [32]. In timetabling, for example, a self-adaptive method is desired to increase the level of generality. Complex behavior required by a robot to navigate its surroundings is another example [33],[34].

2.1. Fundamentals of Genetic Algorithm (GA)

A genetic algorithm (GA) is a powerful problem-solving programming technique. It is in a category of evolutionary algorithms which is a subset of evolutionary computation in artificial intelligence [35]. It was developed in 1960 by Professor John Holland of the University of Michigan. His book, *Adaptation in Natural and Artificial Systems* pioneered genetic algorithm (GA) research in the 1970s [36]. This technique was inspired by the Darwinian theory of natural evolution, which states that the organisms in the world multiply in geometric proportions leading to a struggle for existence due mainly to limited space and food. In this struggle the fittest will survive. The fittest are those with favorable variations, the accumulation of which lead to the evolution of species. The chances for the survival of organisms with injurious variations are rather slim. Thus, evolution is a process of natural selection [38].

Each cell of an organism of a species has a definite number of genetic structures called chromosomes consisting of linearly arranged units called genes (blocks of DNA) carrying genetic information for one or many characters.

A complete set of genetic material, namely all the chromosomes is called a genome. Man for example has 23 pairs of chromosomes. A group of genes in a genome is called the genotype. The expression of a genotype is referred as the phenotype. Each gene has a definite position in the chromosome called the locus. A given gene can be in several states called alleles which express characters such as the eye color of the organism.[34] When organisms reproduce, the resulting offspring will have half the genes from each parent. Due to cross-over and mutations of genes offspring may get new features, favorable or unfavorable for survival.

Genetic algorithms closely mimic the biological model of chromosomes and genes with each chromosome representing an individual organism and genes forming components of a solution that is to be used with a genetic algorithm. Thus, a chromosome represents a data structure. As in nature, new chromosomes are generated by mixing genetic material by means of cross over and mutation. Cross over is equivalent to mating in the biological process. New information is introduced to the population by way of mutation. Thus, it is the chromosome (individual) that changes (evolves) by altering the order and the makeup of its genes. It is noteworthy that a finite number of genes are used in the process.

2.2. How Genetic Algorithms Work

In a genetic algorithm, a population of chromosomes consisting of a given random collection of genes is initiated according to the following steps (Figure 1).

1. Generating an initial population of chromosomes.
2. Evaluating the suitability of each chromosome (individual) that forms the population.

3. Selecting the chromosomes for mating based on the above results.
4. Producing offspring by mating (cross over) the selected chromosomes.
5. Mutating genes randomly.
6. Repeating steps 3-5 until a new population is generated.
7. Ending the algorithm when the best solution obtained has not changed after a preset number of generations.

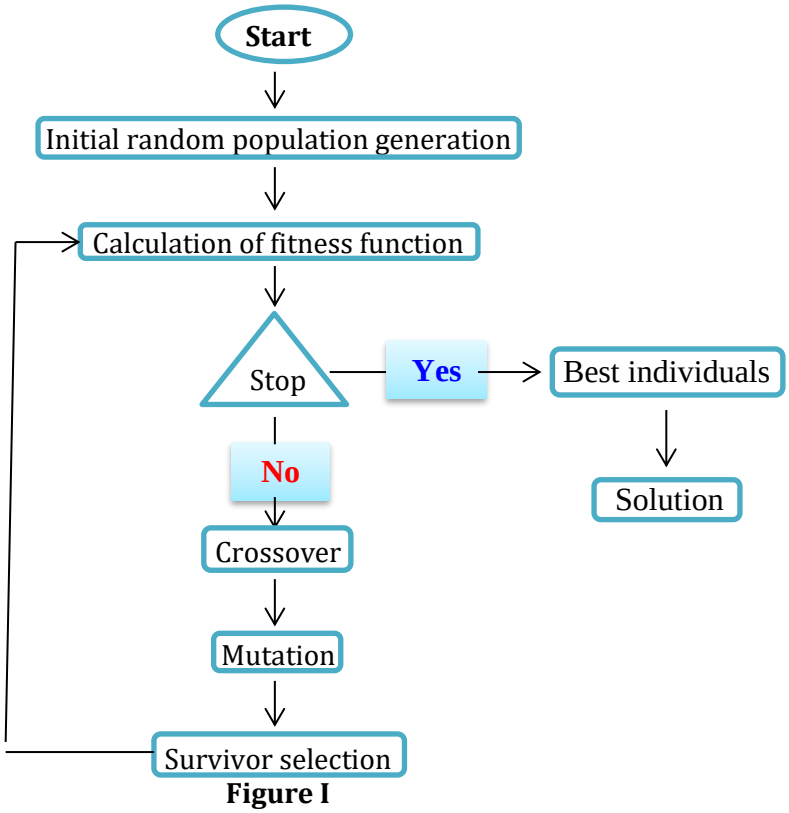


Figure I

Three basic operations are evident from the above procedure. They are the selection, cross over, and mutation. The technique provides more than one solution for complex problems such as the “NP-hard” problems which probably cannot be solved exactly in polynomial time. Such problems increase at a much greater rate, described by the factorial operator (n!). An example of an NP-hard problem is the traveling salesman problem.

Polynomial problems solved by present day computers involve several steps to arrive at a solution. Each problem is bound by a polynomial, a mathematical expression of exponents and expressions.

The above steps of genetic algorithm are elaborated as follows.

2.3. Initial population of chromosomes

A genetic algorithm creates an initial population of chromosomes with each chromosome (organism) representing a complete solution to a given problem. The genes which constitute the chromosomes are initialized to random values. Based on a function specific to a given problem each chromosome is evaluated for its “fitness” which defines the quality of the solution.

2.4. Suitability of chromosomes to mate

In a genetic algorithm, creation of an improved population is the aim of mating the most suitable chromosomes in a population. It results in the production of offspring (chromosomes) which join the existing population of chromosomes to become part of a

mating population in subsequent generations. For the genetic algorithm to function effectively a "fitness function" in the form of a numeric score has to be designed to evaluate the solutions (chromosomes). In nature there is no assignment of a score the -- organisms just die or survive.

As an example of designing a numeric score to evolve the word "run" as described in the literature, a population consisting of three members namely, "son", "rug" and "cat" can be considered [39]. The word "rug" has two correct features as against the other two and has the highest number of points. The word "son" has one feature against that of "mat" with none of the characters having a score amounting to zero. Thus, the "fitness" in this example is attributed to the number of correct features as depicted in **Table I**.

DNA	**Fitness**
son	1
rug	2
mat	0

Table I

2.5. Selection to the mating pool.

With the above example it is obvious that only two members are fit to be parents to be placed in a mating pool as shown by the fitness calculations made on the members of the population. One of the approaches called the *elitist* method can be used to select the members with highest scores are allowed to make offspring for the next generation. If there are only two members among thousands available as parents, the variety of the offspring will be very much limited, which could stunt the process of evolution. The obvious conclusion therefore is to make a large mating pool to obtain better results. However, this

will not render optimal solutions as the top scoring individuals have the same chance of being selected as the ones toward the middle.

2.5.1. Roulette Wheel Selection

A method popularly known as the "wheel of fortune" or the "roulette wheel" based on probability seems to provide a better solution in selecting members to the mating pool. A simple example to illustrate the method is to consider a five member population having the following fitness scores **Table II**.

DNA	Fitness
P	4
Q	3
R	0.7
S	1.3
T	1

Table II

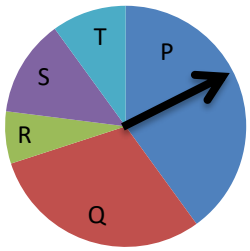
The total fitness is 4 + 3 + 0.7 + 1.3 + 1 = 10

The normalized fitness scores and the percentage values are given in **Table III**.

DNA	Fitness	Normalized Fitness	Expressed as a Percentage
P	4	0.4	40%
Q	3	0.3	30%
R	0.7	0.07	7%
S	1.3	0.13	13%
T	1	0.1	10%

TABLE III

2.5.2 Selection based on the wheel of fortune.



Parent	Probability
P	40%
Q	30%
R	7%
S	13%
T	10%

Figure 1I

The selection chance for members **P - T** are **P> Q> S> T>R** **(Figure II)**

This procedure is considered satisfactory because it ensures the reproduction of the highest scoring individuals without completely eliminating the ones with lowest scores providing an opportunity to pass the information to the next generation. This could be explained by trying to evolve the phrase, "to do or not to do" using the available individuals P-R.

P: to do or not to be
Q: to do or not to go
R: xxxxxxxxxxxxxxxxdo

The chromosomes P and Q have most of the features of the required phrase having the highest scores as against R with lower score. However, the individuals P and Q lack the correct features for the end of the phrase. R on the other hand contains the necessary genetic data for the end of the phrase. As it can be seen, elements P and Q are clearly the most fit and would have the highest score. But neither contains the correct characters for the end of the phrase. Element R, even though it would receive a very low score, happens to have the genetic data for the end of the phrase. Whilst P and Q are selected to produce majority of offspring of the next generation it is necessary to have a some opportunity for R to take part in reproduction to obtain a perfect solution for the present problem.

2.5.3. Tournament selection

This method also selects individuals for the mating pool depending on their fitness function. It is a frequently used method in GA due to its simplicity and efficiency. The

method involves a random selection of individuals from a bigger population. They will compete with each other for a place in the next generation population.

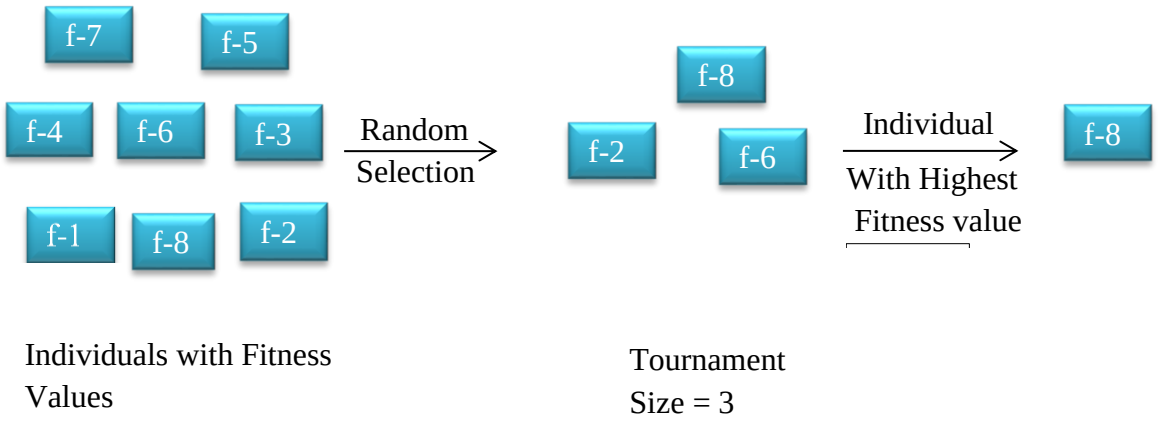


Figure IV: Tournament mechanism

2.5.4 Stochastic Tournament selection

In stochastic tournament selection, the individual selected by the Roulette Wheel is taken to the Tournament.

Scaling, Rank, Generational, Steady-state and Hierarchical are other selection methods used [40].

The convergence rate together with the number of generations required to yield the optimal solution defines the performance of the genetic algorithm [42].

The Roulette Wheel method is associated with high time complexity as well as faster convergence with the selection of certain parents. The tournament method, on the other hand, exhibits lesser time complexity and therefore is faster [41]. In this study, tournament

selection method is used for its simplicity with regards to coding and efficiency towards parallel and nonparallel constructions [42],[43]. In addition, this method leaves room for selection pressure adjustments which has an impact on its usage [43],[44]. Selection of individuals with higher fitness values for the next generation is controlled by selection pressure which is described as the extent to which the better individuals of the population are favored. The tournament size can be increased or decreased to obtain a desired selection pressure. A higher tournament size (higher selective pressure) will prevent some individual to take part in a tournament where as some others may not be selected at all. This brings about loss of genetic diversity resulting in premature convergence restricting the results to a local optima.[38] Genetic diversity in essence, is the maintenance of a diverse solution population which makes sure that the solution space is searched adequately. A too small selection pressure (lower tournament size) on the other hand, may take a longer time to converge to an optimum. Thus, a proper selection pressure which ensures genetic diversity is required for the genetic algorithm to converge to a global optimum [42].

2.6. Methods of change in reproduction.

2.6.1 Cross-over

The characteristics of the suitable individuals selected must be altered in order to improve their fitness to the next generation. This is achieved by means of cross-over and mutation. Crossover simulates the recombination of chromosomes in sexual reproduction which involves two parents. To explain the phenomenon the following two parent phrases (individuals) from the mating pool can be considered.

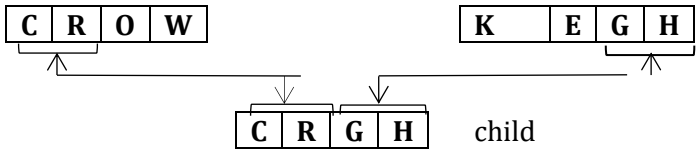


Figure V

A child phrase can be made in a 50/50 manner by taking two features from each parent as shown in **Figure V.** This is one-point crossover

Picking up a random point instead of the 50/50 selection is preferred as it increases variety for the next generation as indicated in **Figures VI and VII**.

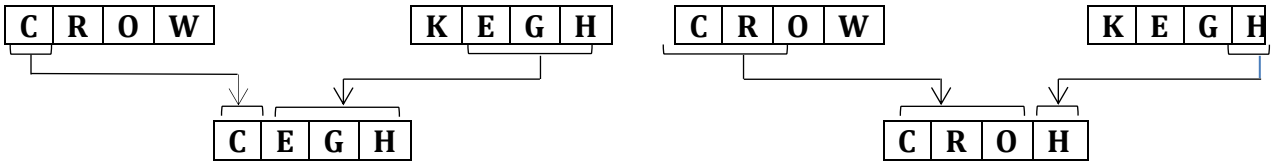
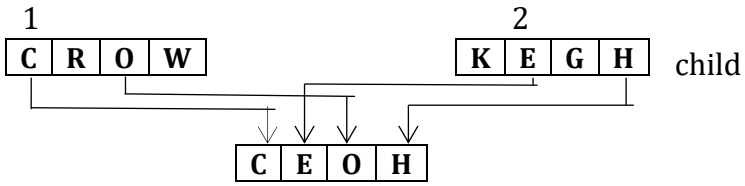


Figure VI: Selecting a random point.

Figure VII:



KEOH, CEOW, CROH, CEOW are some of the possible combinations.

2.6.2 Mutation.

Additional variety to the offspring DNA which was created during the crossover can be introduced by mutation as shown in **Figure VIII**. It is a process of picking up a new random character. A mutation rate of 1%, indicates that each character in the phrase

resulted from crossover, there is a 1% chance that it will mutate. A mutation may produce a desirable or undesirable character. Natural selection will decide upon the fate of the mutated chromosome.

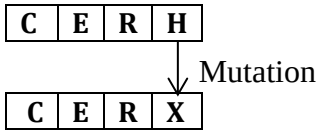


Figure VIII

The processes of selecting parents and reproduction are repeated until desirable results are obtained.

2.6.3. Methods of representation

Several methods are available to encode potential solutions to a given problem in a form that the computer is able to process the data [40]. In one method binary strings in sequences of 1's and 0's are used to encode solutions with each digit representing some aspect of the solution. A similar method is to encode solutions in integers and decimal numbers with each position representing some feature of the solution allowing precision and more complexity than using only the binary numbers [45]. In a third method a string of letters are used to represent an individual where each letter is designated to some aspect of the problem.

All three methods clearly define the changes which the selected individuals undergo during the process. The changes include the flipping of a 0 to 1, changing a letter to another or adding or subtracting a randomly selected amount to a number.

Another approach called genetic programming where programs are represented as branching data structures (**Figure IX**) [46] and random changes are brought about by changing the value at a specific node in the tree. Changes can also be made by replacing one subtree with another.

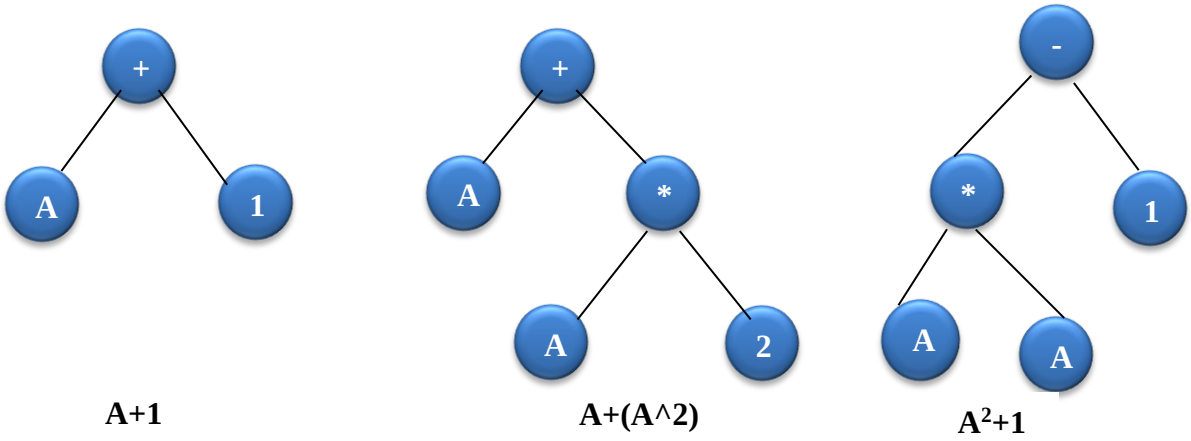


Figure IX

Three common program trees used in genetic programming. The mathematical expression for each program is given under each diagram.

2.7 Advantages of genetic algorithm

As compared to genetic algorithms, other optimization algorithms look for solutions in a serial manner (in one direction) in the search field at a given time. The disadvantage of a serial search is that if the solution obtained is not favorable, the work carried out thus far has to be abandoned and a new search must be started. Genetic algorithms, which have multiple offspring on the other hand, are able to look for solutions in many directions at a given time, abandoning the paths which lead to suboptimal solutions. Thus, the opportunity of finding a favorable solution in a genetic algorithm is high during each run. Further, as

per Holland's schema theorem, the fitness schemata which are above average grow while the ones below average decay. As such, parallelism in addition, indirectly evaluates many different schemata to identify individuals of highest fitness values. Thus, although genetic algorithms seem to evaluate a small group, in reality they evaluate a very large group of individuals in a reasonable amount of time. As an example, an 8-digit binary string designated as, $********$ could be considered. $*$ is either 0 or 1. All these 8 digit strings form a search space. 01101010 string, for example with a fitness of 24, becomes not only a member of this search space, it also becomes a member of the spaces, $0*******$, $01******$, $0******0$, $0*1*1*1*$,$10*01**0$ Genetic algorithms will sample to evaluate the fitness of the particular string in each of these spaces to which the string, 01101010 belongs.

Another advantage of a genetic algorithm (GA) is its capability to handle several parameters simultaneously. Usually many of the real-world problems can only be described in terms of multiple objectives. As such, they cannot be expressed in the form of a single value for minimization and maximization purposes. A parameter can only be improved at the expense of another. Due to parallelism however, a GA can generate many solutions with one individual optimizing one parameter and another optimizing another. This allows one to select a desired solution for use.

2.8 Disadvantages of genetic algorithm

There are disadvantages to genetic algorithms. The representation for the problem must be clearly defined. The language employed to describe solutions should be able to bear random changes so that no fatal errors occur. Binary, integer or real-valued list of numbers is usually used to define individuals.

Choosing a fitness function is difficult. It should be carefully written to obtain the desired solution. An ill-defined fitness function could end up in not getting any solution or it might result in solving the wrong problem [47]. In addition to getting a good fitness function, it is also necessary to pay attention when selecting such factors as the number of individuals (population), crossover and mutation rates. Less solution space created by a small population leads to inaccurate results.

Premature convergence is another problem associated with a GA, especially when small populations are involved. This results from an individual (more fit) who is relatively capable of reproducing abundantly diminishes the population diversity too soon leading to convergence to a local optimum. In such an event, finding the global optimum becomes impossible as there could not be a search in the search field due to the particular individual representation. However, there are methods that can be adopted to overcome this problem.

Analytical techniques are generally recommended to solve analytical problems instead of GA methods, as the former take less computational effort and time.

CHAPTER 3

DESIGN AND IMPLEMENTATION STRATEGY

3.1. Encoding

As stated earlier, scheduling classes for a college timetable is often met with constraints (hard and soft) due to diversity as compared to a school timetable where the requirements are highly limited.

The class-scheduling problem will be based on the available professors, classrooms, timeslots and student groups.

Each class will be assigned a timeslot, a professor, a room, and a student group by the class scheduler. The total number of classes that needs to be scheduled can be obtained by summing the number of student groups multiplied by the number of modules each student group is enrolled in.

For each class scheduled by this application the following hard constraints will be considered.

- Classes can only be scheduled in free classrooms.
- A professor can only teach one class at any one time.
- Classrooms must be big enough to accommodate the student group.

When encoding the class schedule, certain class properties are needed. They are: the timeslot the class is scheduled for, the professor teaching the class, and the classroom required for the class.

The encoding used must be able to encode all the class properties that are required. The class properties are, the timeslot the course is scheduled for, the professor teaching the course and the classroom for the course.

A numerical code can be allocated to each timeslot, professor, and the classroom. A chromosome can then be used that encodes an array of integers to represent each class.

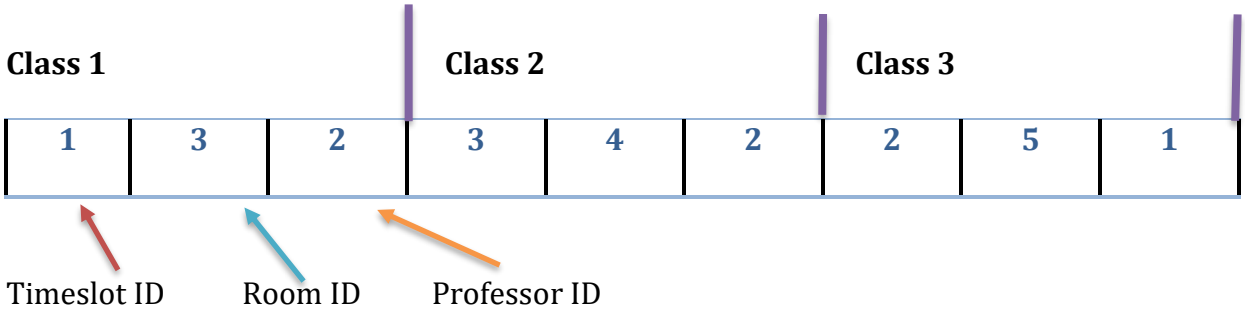


Figure X: Array is split into three to retrieve information for each class.

3.2. Initialization

A timetable needs to be built around the following criteria : the rooms, professors, timeslots, courses/modules, and student groups.

The room class will contain information about the classroom, such as the roomID, room number and the capacity of students that can be accommodated. This class will accept a roomId, a room number and the capacity as well as provide methods to get the room's properties.

The timeslot class represents the day of the week and time that a class takes place. It contains the timeslotId and the timeslot details.

The professor class accepts a professorID and professor name properties. It also makes an allowance to retrieve this information as well.

A module class will store the information on the course modules. Each module can have several sections and groups of students taking the course at different times of the week with different professors. The module class accepts a moduleID, module code ("CSCI111" or "ENGL 101"), module name, an array of professorID's (professors who teach the module).

A group class will hold the information about the student groups. This class accepts groupID, a group size, module IDs the group is taking.

The Class class combines all of the above information. It will take a student group that takes a section of a module at a given timeslot, in a specific room with a specific professor.

A timetable class will encapsulate all these objects and will co-ordinate how different constraints interact with each other. This class will also parse a chromosome and create a candidate timetable to be appraised and recorded. The timetable class serves two purposes. First, it is aware of all the available rooms, timeslots, professors, modules and groups. Second, it can read a chromosome, generate a subset of classes from that chromosome, and help evaluate the fitness of the chromosome. This class consists of two significant methods. The create class method and the calculation of clashes method. When creating a class for the timetable, an individual (chromosome) must be accepted, read and assigned information (timeslot, room, professor) to each class. As a result, the create class method uses a genetic algorithm and the subsequent chromosome to try different combinations of timeslots, rooms, and professors. The timetable class stores this information for future use. The clashes method then checks each one of the classes that has been built and count the number of clashes. A clash is a hard constraint violation. Examples

of clashes: room is too small, conflict with professor and timeslot, conflict with room and timeslot.

Clashes are used later in the genetic algorithm to calculate the fitness value. As each class is compared to all other classes, a "clash" is added, if any of the hard constraints are violated. The total number of clashes are calculated. This is then used to calculate the fitness value. The fitness value is the inverse of the number of clashes (1/clashes+1). If there are no clashes then the fitness value will be 1.

The main method in the timetable class creates the timetable and initializes it with all of the available courses, timeslots, rooms, modules, groups and professors. As a result, tournament selection and uniform crossover is used for the genetic algorithm.

A termination check is set up such that the deciding factors are the number of generations and the fitness factor. Combining both of these factors will terminate the genetic algorithm either after a certain number of generations or if it finds a valid solution. As such the fitness value depends on the number of broken constraints. As a result, the perfect solution will have a fitness value of 1. The number of generations is set to 1000.

Uniform crossover is applied to guarantee that chromosomes are selected at random and are swapped with a parent, within the collections of genes.

Mutation is implemented in such a way that a new random valid individual is created. The random individual created is used to select genes to copy into the individual to be mutated. This is called *uniform mutation*. This technique ensures that all the mutated individuals are valid.

CHAPTER 4

EXPERIMENTS AND RESULTS

The hard constraints were tested to ensure that all the solutions obtained were valid.

The optimized solution for the timetable consisted of the following factors and their values. The population size was 100 with a mutation rate of 0.01%, a crossover rate of 0.9%, the number of elite individuals was 2 and the tournament size was 5. With these values the resulting timetable had zero number of clashes with the fitness value of 1.

4.1 Architectural specifications

JAVA version 8 was used on a 6th Generation Intel(R) Core (TM) i7 Quad Core (6M Cache, upto 3.5 GHz) 16GB Dual Channel DDR4 256GB PCIe Solid State Drive with NVIDIA GTX960M 2GB DDR5 for the design of the Course Scheduler.

4. 2 Performance Analysis

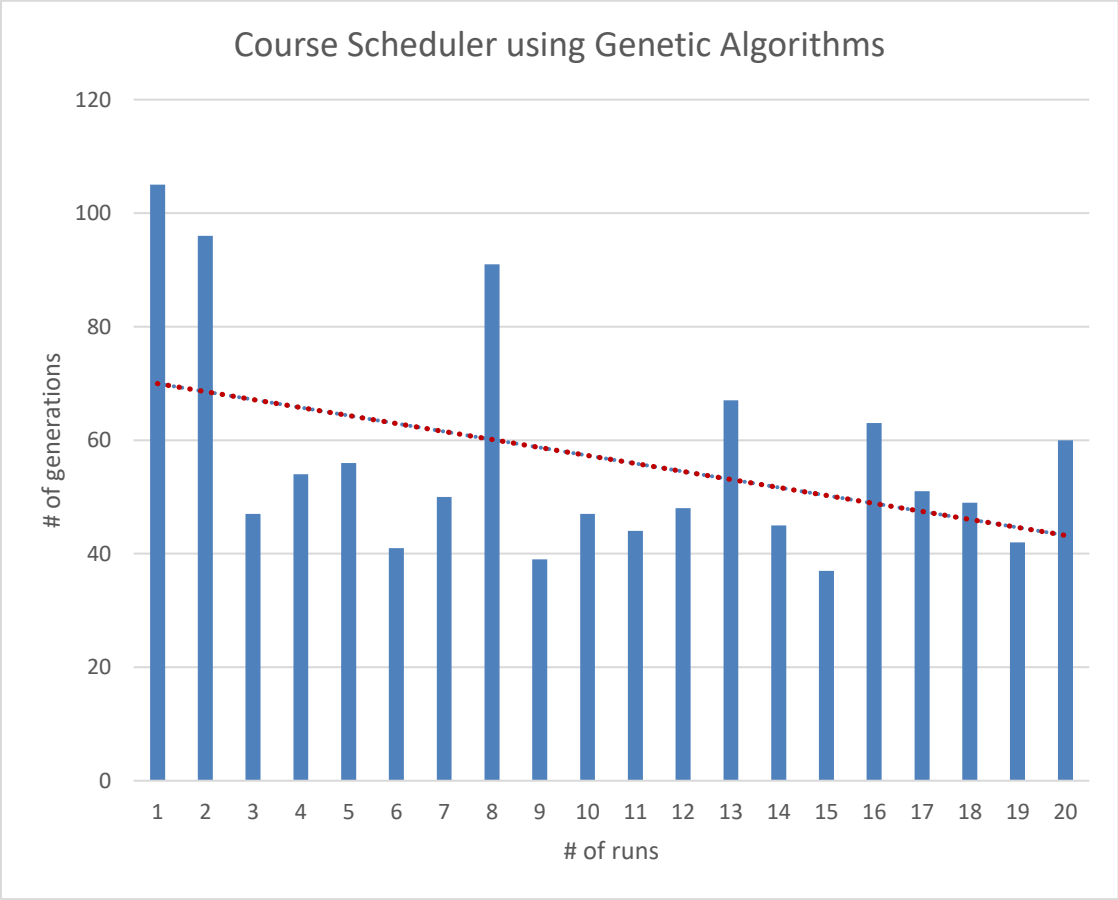


Figure XI: The number of generations the Course Scheduler takes to reach an optimal solution.

The increase in population size shows a steady increase in the number of generations that takes to reach a valid solution.

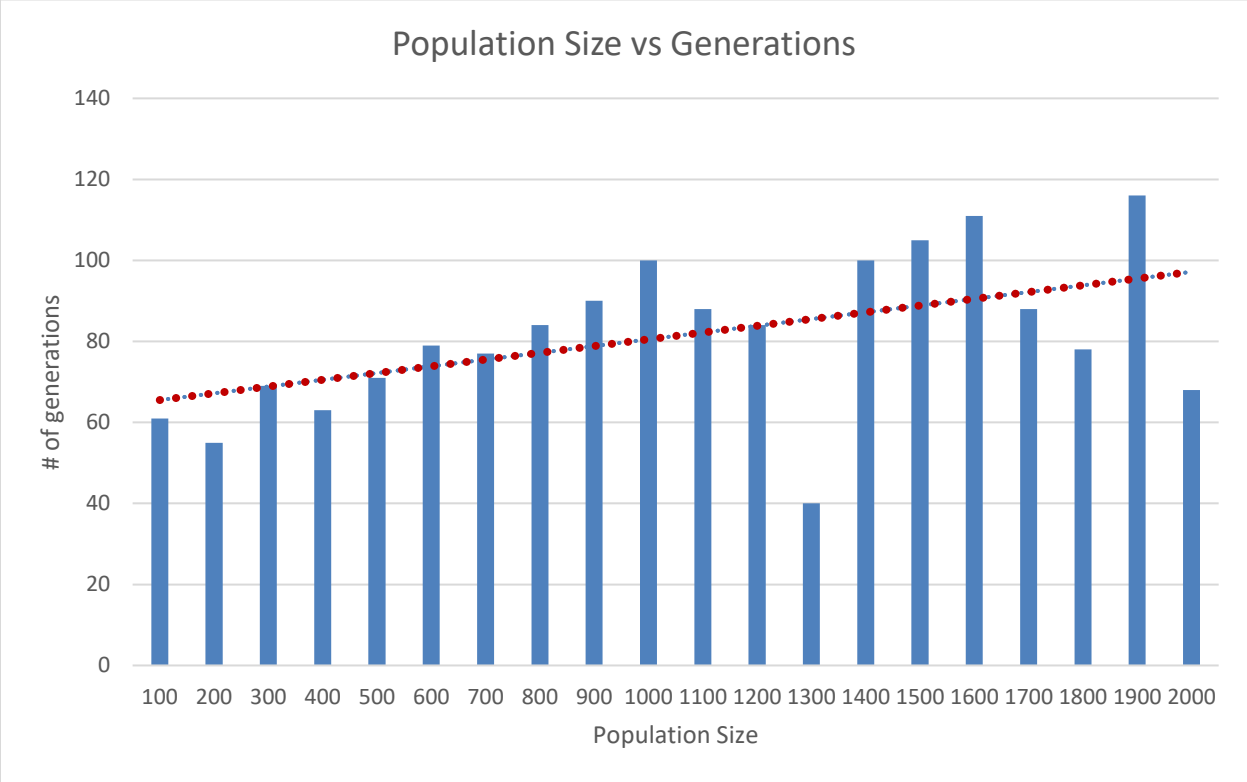


Figure XII: Population size increase vs number of generations to reach a solution.

Mutation rate plays an important role in genetic algorithms. As a result, the effect of clashes and the fitness value were checked with the course scheduler. Here, the number of generations were limited to 1000, the mutation rate range was from 0.01 to 0.21.

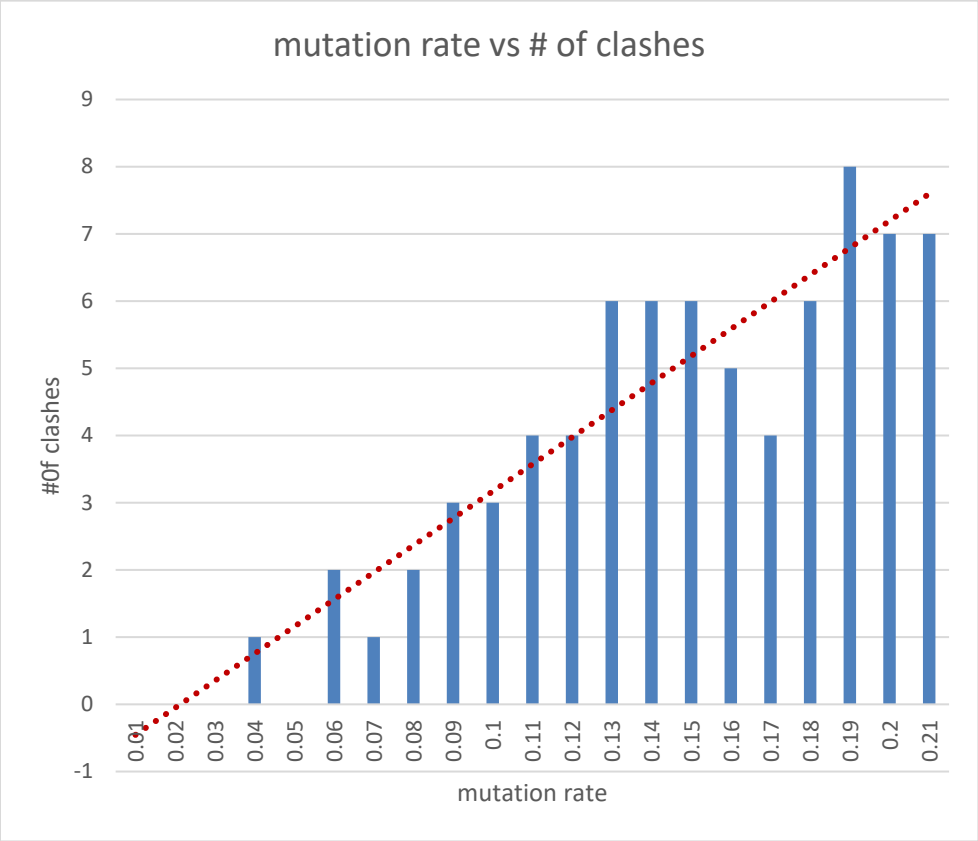


Figure XIII: mutation rate vs number of clashes.

As seen from the graph above, the increase in mutation rate increases the number of clashes. As a result, the mutation rate is optimal at 0.01.

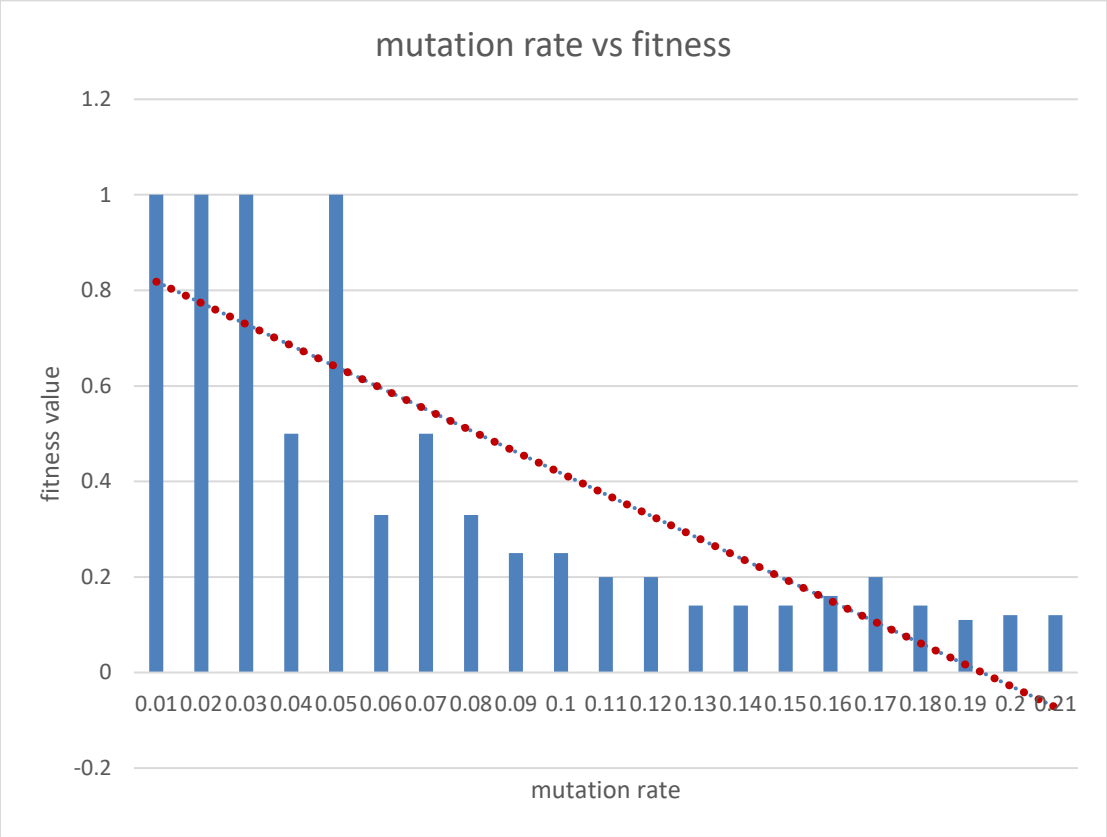


Figure XIV: mutation rate vs fitness value.

The increase in mutation rate decreases the fitness value as seen from the graph in Figure XIV.

CHAPTER 5

CONCLUSION

The course scheduler using genetic algorithms finds the best solution that satisfies a number of hard constraints. Also, the mutation technique used guarantees that the mutated chromosomes remain valid. This was done by creating a known valid random individual, swapping genes with it similar to uniform crossover. Use of uniform crossover and tournament selection completed the algorithm. The factors such as population size, mutation rate, crossover rate, elite individuals, and tournament size were considered for the course scheduler. The optimum values for these factors were obtained through several runs of the system. Also, the effect of mutation rate and the population size were studied for the course scheduler.

BIBILOGRAPHY

Bibilography:

[1] Khaled Elleithy, Tarek Sobh, Innovations and Advances in Computer, Information , Systems Sciences, Part 1, Ed., Springer, New York, Heidelberg Dordrecht, London, ISSN 1876-1100 (2013).

[2] S. Daskalaki, T. Birbas, E. Housos, An integer programming formulation for a case study in university timetabling, *Eur. J. Oper. Res.*, **153**, 117-135 (2004).

[3] E. K. Burke, S. Petrovic, Recent research directions in automated timetabling, *Eur. J. Oper. Res.*, **140**, 266–280 (2002).

[4] A. Schaerf, Local search techniques for large high-school timetabling problems, *IEEE Transactions on Systems Man. And Cybernetics, Part A*, **29** (4), 368-377 (1999).

[5] H.P. Hariyadi, T. Widiyaningtyas, et. al., Implementation of Genetic Algorithm to Academic Scheduling System, *IEEE*, Region 10 Conference, (2016).

[6] A.F. AbouElhamayed, A.S. Mahmodd, et. al., An Enhanced Genetic Algorithm-Based Timetabling System with Incremental Changes, *IEEE*, (2016).

[7] T. Islam, Z. Shahriar, et. al., University Timetable Generator Using Tabu Search, *Journal of Computer and Communications,* **4**,28-37, (2016).

[8] V. Rohini, A.M. Natarajan, Comparison of Genetic Algorithm with Particle Swarm Optimization, Ant Colony Optimization, and Tabu Search based on University Course Scheduling System, *Indian Journal of Science and Technology,* **9**(21), (2016).

[9] M. Dimopoulou, P. Miliotis, An automated university course timetabling system developed in a distributed environment: A case study, *Eur. J. Oper. Res.*, **153** 136–147 (2004).

[10] E. Burke, K. Jackson, J. H. Kingston, R. Weare, Automated university timetabling: The state of the art, *The Computer Journal*, **40** (9): 565-571, (1997).

[11] H. Frangouli, V. Harmandas, P. Stamatopoulos, UTSE: Construction of optimum timetables for university courses - A CLP based approach, Conference on the practical applications of prolog, Paris, France, 225-243 (1995). Publisher: Alinmead

Software Ltd., ISBN: 0952555409, 9780952555407.

[12] C. Gueret, N. Jussien, P. Boizumault, C. Prins, Building University Timetables UsingConstraint Logic Programming. In *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling*, 393-408, (1995).

[13] J. W. Shen, Solving the graph coloring problem using genetic programming, "Genetic Algorithms and Genetic Programming at Stanford 2003, Ed. J. R. Koza, 187-196. Publisher: Stanford Bookstore.

[14] H. Babaei, J. Karimpour, A. Hadidi, A survey of approaches for university course timetabling problem, *Comput. Ind. Eng.*, **86**, 43–59 (2015).

[15] T. A. Redl, On using Graph Colouring to Create University Timetables with Essential and Preferential Conditions, *Advances in Marketing, Management and Finances, Proceedings of the 3rd International Conference on Management, Marketing and Finances (International Conference on Computational and Information Sciences, University of Houston-Downtown*, 162- 167 (2006).

[16] A. Tripathy, School Timetabling - a Case in Large Binary Integer Linear-Programming. *Management Science*, **30**(12): 1473-1489 (1984).

[17] M. A. Badri, A two-stage multiobjective scheduling model for [faculty-coursetime] assignments. *European Journal of Operational Research*, **94**(1),16-28, (1996).

[18] T. Birbas, S. Daskalaki, and E. Housos, Timetabling for Greek high schools. *Journal of the Operational Research Society*, **48**(12): 1191-1200, (1997).

[19] M. Dimopoulou, P. Miliotis, Implementation of a university course and examination timetabling system. *European Journal of Operational Research*, **130**(1): 202-213 (2001).

[20] Papoutsis, K., C. Valouxis, E. Housos, A column generation approach for the timetabling problem of Greek high schools. *Journal of the Operational Research Society*, **54**(3): 230-238 (2003).

[21] S. Daskalaki, T. Birbas, and E. Housos, An integer programming formulation for a casestudy in university timetabling. *European Journal of Operational Research*, **153**(1): 117-135 (2004).

[22] S. M. Al-Yakoob, H. D. Sherali, A mixed-integer programming approach to a class timetabling problem: A case study with gender policies and traffic considerations. *European Journal of Operational Research*, 2007. **180**(3):1028-1044.

[23] S. A. MirHassani, A computational approach to enhancing course timetabling with integer programming. *Applied Mathematics and Computation*, **175**(1):814-822 (2006).

[24] Hinkin, T.R. and G.M. Thompson, SchedulExpert: Scheduling courses in the Cornell University School of Hotel Administration. *Interfaces*, **32**(6), 45-57 (2002).

[25] D. Johnson, A Database approach to course timetabling, *The Journal of the Operational Research Society*, Vol. 44(5), 425-433, (1993).

[26] L. Lalescu, C. Badica, Timetabling experiments using genetic algorithms. In *Proceedings of the International 12th Turkish Symposium on Artificial Intelligence and Neural Networks* (TAINN-2003), Canakkale, Turkey, (2003).

[27] K. F. Man, K. S. Tang, and S. Kwong, Genetic Algorithms: Concepts and Applications, *IEEE Trans. Ind. Electron*. **43** (5), 519 – 534, (1996).

[28] A. Dal Palù, A. Dovier, Federico Fogolari, Constraint Logic Programming approach to protein structure prediction, *BMC Bioinformatics*, 5: 186 (2004).

[29] S. Abdullah, H. Turabieh, "Generating university course timetable using genetic algorithm and local search", *Proc. 3rd Int. Conf. Hybrid Inform. Tech.*, pp. 254-260, (2008).

[30] D. Mitta, H. Doshi, M. Sunasra, R. Nagpur, Automatic Timetable Generation using Genetic Algorithm, *Int. J. Adv. Res. Comput. Commun. Eng.*, **4** (2), 245-248, (2015) .

[31] T. Jain, N. Jamil, Genetic Algorithm Approach to Time Tabling Problem, *European Journal of Business and Management* , **7**(4) 7-11, (2015).

[32] J. E. Smith, T. Fogarty, "Operator and parameter adaptation in genetic algorithms," Soft computing : a fusion of foundations, methodologies and applications, **92**, 81–87, (1997).

[33] M. Mitchell, C. E. Taylor, Evolutionary computation: An overview. Annu. Rev. Ecol. Syst. **30**, 593–616 (1999).

[34] S. Petrovic, E. K. Burke 2004. University Timetabling. In: Leung J. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. Chapter 45. CRC Press.

[35] Rosen, Kenneth H. (2012). *Discrete Mathematics and Its Applications* (7th ed.). New York: McGraw-Hill. p. 119. ISBN 978-0-07-338309-5.

[36] D. Shiffman, The Nature of Code, Chapter 9, 2012, Creative Commons, 444 Castro Street, Suite 900, Mountain View, California 94041, USA. ISBN-13: 978-0985930806.

[37] E. R. Pianka, *Evolutionary Ecology,* 6th ed. San Francisco, CA: Addison-Wesley-Longman, 2000; Deepti Gupta, Shabina Ghafir, International Journal of Emerging Technology and Advanced Engineering**, 2**(5), (2012).

[38] Z. Michalewicz, Genetic Algorithms+Data Structures = Evolution programs, second edition, Springer-Verlag Heidenberg New York, 15 (1996). (ISBN 3-540-60676-9).

[39] D. Shiffman, The Nature of Code, 2012, Creative Commons, 444 Castro Street, Suite 900, Mountain View, California 94041, USA., ISBN-13: 978-0985930806.

[40] A. Marczyk, Genetic Algorithms and Evolutionary Computation, (2004).

[41] A. Jain, S. V. Chande, International Journal on Computational Science & Applications (IJCSA) **5(**6), (2015).

[42] N. M. Razali, John Geraghty, Genetic Algorithm Performance with Different Selection Strategies in Solving TSP, Proceedings of the World Congress on Engineering, Vol II, WCE 2011, July 6 - 8, (2011), London, U.K.

[43] B. L. Miller, D. E. Goldberg, Genetic Algorithms, Tournament Selection, and the Effects of Noise, *Complex Systems* 9, 193- 212 (1995).

[44] H. Xie, M. Zhang, Tuning Selection Pressure in Tournament Selection, Technical Report Series, School of Engineering and Computer Science, Victoria University of Wellington, New Zealand (2009).

[45] P. J. Fleming, R. C. Purshouse (2002), Control Evolutionary Practice, 10: 1223-1241.

[46] J. Koza, M. Keane, M. Streeter, W. Mydlowec, J. Yu, G. Lanza, Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Kluwer Academic Publishers, 35, (2003).

[47] D. Graham-Rowe, "Radio Emerges from the Electronic Soup.", New Scientist, **175**, 19, (2002).

VITA

Achini Kumari Herath

103, Jamie Cove • Oxford, MS 38655 • (662) 915- • akherat1@olemiss.edu

A. EDUCATION

1. Graduate Diploma, British Computer Society (UK), Sri Lanka (2002)
2. BSCS in Computer Science University of Mississippi Oxford, MS (2009)
3. Masters in Computer Science, University of Mississippi, Oxford, MS (2017)
4. Ph.D. in Computer Science, University of Mississippi, Oxford, MS (pending)

B. WORK EXPERIENCE

The University of Mississippi, Oxford, MS

1. **Graduate Instructor**:
    a. Teaching –Intoduction to the Java Programming Language (Spring 2017-).
    b. Teaching – MS Office Applicvations (MS Word, MS Excel, MS Access, MS PowerPoint) undergraduate students (August 2016)
    c. Teaching –Intoduction to the Java Programming Language (June-August 2016).
2. **Computer Department – Head Teaching Assistant** (August 2015 – May 2016):
    a. Selecting and scheduling student TA's for weekly labs.
    b. Coordinating labs between the instructor and student TA's.

c. Preparing and completing grade related paperwork.

d. Meeting with students for scheduling missed labs.

e. Receiving feedback from students, teaching assistants and instructors/faculty to improve the performance.

3. Graduate Instructor

a. Teaching MS Office Applications (August 2013 – May 2015)

4. Teaching Assistant

a. JAVA

5. Technical Skills

a. JAVA
b. C++
c. Javascript
d. MS Office Applications

6. Career Overview

I have worked as a software programmer in Sri Lanka while studying for my Graduate Professional Diploma (British Computer Society -UK) in Sri Lanka. After moving to USA, I worked as a student worker at the University of Mississippi while studying for my bachelors degree in Computer Science. I am currently working as a graduate instructor and reading for my Ph.D.

7. References

Dr. Dawn Wilkins
Professor and Chair
203 Weir Hall
662 915 7309
dwilkins@cs.olemiss.edu

Dr. H. Conrad Cunningham
Professor (Chair 2001 – 2015)
11 Weir Hall
662 915 5358
hcc@cs.olemiss.edu