

TITLE PAGE

**DESIGN AND IMPLEMENTATION OF MOBILE BASED
STUDENTS TIMETABLE MANAGEMENT SYSTEM**

**(A CASE STUDY OF DEPARTMENT OF COMPUTER SCIENCE
AKANU IBIAM FEDERAL POLYTECHNIC UNWANA, AFIKPO,
EBONYI STATE)**

BY

ONUWA, NNACHI ISAAC

2013/HND/15268/CS

SUBMITTED TO

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF SCIENCE

**AKANU IBIAM FEDERAL POLYTECHNIC, UNWANA, AFIKPO,
EBONYI STATE.**

**IN PARTIAL FULFILMENT FOR THE AWARD OF HIGHER
NATIONAL DIPLOMA (HND) IN COMPUTER SCIENCE**

DECEMBER 2015

CERTIFICATION / APPROVAL PAGE

This is to certify that this project work is carried out by ONUWA, NNACHI ISAAC (2013/HND/15268/CS) has been scrutinized and approved by the undersigned as part of the requirements for the award of the Higher National Diploma in Computer Science, Akanu Ibiam Federal Polytechnic Unwana.

.....

MR. J. O IDEMUDIA

PROJECT SUPERVISOR

.....

DATE

.....

DR. A. N. EZEANO

HEAD OF DEPARTMENT

.....

DATE

.....

EXTERNAL SUPERVISOR

.....

DATE

DEDICATION

To my parents.

ACKNOWLEDGMENT

My profound gratitude goes to Jehovah the creator of the universe.

This work is a synergistic product of many minds and I feel a deep sense of gratitude to my parents, Mr and Mrs Onuwa U. Onuwa for their encouragement and for being ever supportive. My sincere thanks goes to my supervisor Mr J. O. Idemudia for his thorough assistance with this work and to Mr Chibuike Madubike and Mr Okwara K. K for their encouragement and advice.

I appreciate the effort and care of the Head of Computer Science Department Dr. A. N. Ezeano (Ph.D.). My thanks goes to Chief Mrs. V. N. Ezeano (J.P), Mr E. U. Ezeorah and Mr U. C. Ugboajah. I also acknowledge the Computer Science Students (set 2013/2014) for their active verbal participation and suggestions towards the evolvement of this project work.

I am greatly indebted to my Uncle, Mr Arua Smart and my Port Harcourt parents Mr and Mrs Moses Ikekwem who did not count on the hard economy but tried their best to send me to school and supported me both financially, materially and morally. I cannot as well fail to acknowledge my siblings – Chinedu, Nnamdi and Onyinyechi, my aunties Aunty Chidinma, Aunty Chinyere and Aunty Nkechi, my cousins Chinyere Ikekwem, Obinna Ikekwem, Helen Ude, Ukachukwu Ukpai. I love you all.

ABSTRACT

Lecture timetabling is a very important process in any educational institution. It is an open-ended program in which courses must be arranged around a set of time slot 'T' and remains so that some constraints are satisfied. It constitutes a class of difficult-to-solve optimization problems that lacks analytical solution method. Data gathering on the current system was analysed to create a requirement definition for the improved timetable system. Literature review was carried out to search the best approach that can help to solve the problem in the timetable system. Genetic Algorithm has been implemented in the Timetable Management System. This is because Genetic Algorithm is able to produce a feasible timetable system. Java, XML and PHP programming languages were used in developing the solution. MySQL database was used as the back-end for the solution. The front-end solution will be implemented in an android mobile operating system for easier accessibility and proximity to users.

TABLE OF CONTENTS

TITLE PAGE	i
APPROVAL PAGE	ii
DEDICATION	iii
ACKNOWLEDGEMENT	iv
ABSTRACT	v
TABLE OF CONTENT	vi
CHAPTER ONE: INTRODUCTION	
1.1 BACKGROUND OF THE STUDY	3
1.2 STATEMENT OF THE PROBLEM	4
1.3 OBJECTIVES OF THE STUDY	4
1.4 SIGNIFICANCE OF THE STUDY	5
1.5 SCOPE OF THE STUDY	6
1.6 LIMITATION OF THE STUDY	6
1.7 DEFINITION OF TERMS	6
CHAPTER TWO: REVIEW OF RELATED LITERATURE	
2.1 REVIEW OF RELEVANT THEORIES AND TECHNOLOGIES	9
2.2 TIMETABLING AS A NP-COMPLETE PROBLEM	14
2.3 BRIEF HISTORY OF GENETIC ALGORITHMS	15
2.4 BASIS FOR A GENETIC ALGORITHM	19
2.5 METHODS OF REPRESENTATION	21
2.6 METHODS OF SELECTION	23
2.7 METHODS OF CHANGE	26
2.8 STRENGTHS OF GENETIC ALGORITHMS	27
2.9 LIMITATIONS OF GENETIC ALGORITHMS	35
CHAPTER THREE: SYSTEMS INVESTIGATION AND ANALYSIS	
3.1 ORGANOGRAM FOR COMPUTER SCIENCE DEPARTMENT	46
3.2 FACTS FINDING	48
3.3 ANALYSIS	48
3.4 PROBLEM OF THE CURRENT SYSTEM	49
3.5 PROPOSING A NEW SYSTEM	50
3.6 ADVANTAGES OF THE PROPOSED SYSTEM	50
CHAPTER FOUR: SYSTEM DESIGN	
4.1 OBJECTIVES OF THE DESIGN	52
4.2 SYSTEM BLOCK DIAGRAM	53
4.3 OUTPUT DESIGN	54
4.4 INPUT DESIGN	54
4.5 PROGRAM DESIGN	54
4.6 DATABASE	57
4.7 DATABASE SPECIFICATION	57
4.8 PROGRAM FLOWCHART	58
4.9 MODELLING THE SYSTEM	59
4.10 CHOICE OF PROGRAMMING LANGUAGE	66

CHAPTER FIVE: SYSTEM DOCUMENTATION AND IMPLEMENTATION	
5.1	SYSTEM REQUIREMENTS67
5.2	HOW TO INSTALL68
5.3	TRAINING OF OPERATORS68
5.4	IMPLEMENTATION METHOD68
5.5	REVIEW AND MAINTENANCE OF THE SYSTEM70
CHAPTER SIX: CONCLUSION, SUMMARY AND RECOMMENDATION	
6.1	SUMMARY72
6.2	PROBLEMS ENCOUNTERED72
6.3	CONCLUSION72
6.4	CONTRIBUTION TO KNOWLEDGE72
6.5	RECOMMENDATION73
REFERENCES74	
APPENDICES78	

CHAPTER ONE

INTRODUCTION

Timetabling concerns all activities with regard to producing a schedule that must be subjective to different constraints. Timetable can be defined as the optimization of given activities, actions or events to a set of objects in space-time matrix to satisfy a set of desirable constraints.

A key factor in running an educational center or basically an academic environment is the need for a well-planned, well-thoroughout and clash-free timetable. Back in the days when technology was not in wide use, (lecture) timetables were manually created by the academic institution.

Every school year, tertiary institutions are faced with the tedious task of drawing up academic timetables that satisfies the various courses and the respective examination being offered by the different departments.

Timetable development process starts when each Head of Department provide the following information to be used for timetable scheduling. The information provides the modules with dates, time and venues suitable in a particular semester:

- Examinable courses in a particular semester.
- Dates for lectures to be held (Lectures can be scheduled between Monday and Friday).

- Specified time for lectures (i.e. Between 8am and 4pm)
- The venue of the scheduled lectures.

A timetabling problem consists of four (4) parameters and they are: T (set of time), R (set of available resources), M (set of scheduled contacts) and C (set of constraints). This problem assigns time and resources to the contacts on such a way that the constraints will be satisfied. In various timetabling problems, educational timetabling has been generally examined from practical standpoint. Academic timetable is very crucial but it consumes time due to its frequent occurrences and usage among higher institution of learning. Another reason for the difficulty is because of the great complexity of the construction of size of lectures and examinations, due to the scheduling size of the lectures and examinations periods and high number of constraints and criteria of allocation which is usually circumvented with the use of little strict heuristics, based on solutions from previous year (Jose, 2008).

The quality of the timetable determines the quality of time dedicated by lecturers, students and administrators to academic activities. Various academic timetabling includes:

- i. School timetable
- ii. Lecture timetable
- iii. Examination timetable and
- iv. Assignment timetable (Qu, Burke, McCollum, Merlot and Lee, 2004).

This academic timetable must meet a number of requirements and should satisfy the desires of all entities involved simultaneously as well as possible. The timings of events must be such that nobody has more than one event at the same time (Roberts, 2002).

1.1 BACKGROUND OF THE STUDY

The Department of Computer Science was carved out from the defunct Systems Science (Two departments were created out of Systems Sciences. Computer Science and Mathematics/Statistics) in the year 1997 with Mr. C.J.C. Ayatalumo as her first Head of Department.

The department's mission and vision are as follows

- To procure department portal CNET.
- To make the Hardware Maintenance Laboratory fully functional.
- To undertake repairs, maintenance, installation, assembling of computers within and outside the institution in order to generate revenue for the polytechnic.
- To start off new Computer Engineering Technology Department.
- To become Center of Excellence in Computer Science and Engineering.
- To resuscitate staff development programmer in the department.
- To increase the computing equipment in use in both ND and HND laboratories.
- To ensure regular supply of consumables.

- Procurement of up-to-date software (licensed).
- To provide adequate staff offices and facilities.
- Provision of dedicated power supply.

The department of Computer Science is in the School of Sciences and has been accredited to award National Diploma (ND) and Higher National Diploma (HND).

1.2 STATEMENT OF THE PROBLEM

The available system currently builds or generates a set of timetables, but most times have issues with generating a clash-free and complete timetable. The tedious tasks of data introduction and revision of usually incomplete solutions are the bottlenecks in this case (Luisa et.al, 2006). Most educational institutions have resorted to manual generation of their timetables which according to statistics takes much time to get completed and optimal. Even at the optimal stage of the manually generated timetable, there are still a few clashes and it is the lecturer that takes a clashing course that works out the logistics of the course so as to avoid the clash.

1.3 AIM AND OBJECTIVES OF THE STUDY

The literature on and implementation of educational timetabling problem is scattered, vast and far-fetched. Different research papers that have been brought out on timetable may refer to the same type of institution but they mostly deal

with different kinds of assignments, i.e. decisions like timing of events, sectioning of students into groups, or assigning events to locations.

Moreover, each institution has its own characteristics which are reflected in the problem definition (Robertus, 2002). Yet, there have been no leveling ground for developing a system that can work for most of these institutions.

The aim of this work is to generate a timetable while demonstrating the possibility of building the schedules automatically through the use of computers in a way that they are optimal and complete with little or no redundancy.

The objectives of this work are as follows

- To be able to optimize the algorithm used in today's timetable systems to generate the best of timetabling data with fewer or no clashes.
- To bring approved timetable closer to users especially to those who use android phones.

1.4 SIGNIFICANCE OF THE STUDY

The reasons for this work are outlined below

- i. The proposed system will provide an attractive graphical front-end for the administrators and students (mobile platform).
- ii. It will improve flexibility in timetable construction.
- iii. The system will save time.
- iv. Productivity will be improved.

- v. The system can be revised i.e. its backend can be revised.
- vi. Proper recording of class size, number of courses offered, number and capacity of available lecture halls.
- vii. Efficient execution of academic activities.

1.5 SCOPE OF THE STUDY

This study will only cover the management and allocation of spaces and time for lectures in the Department of Computer Science, Akanu Ibiam Federal Polytechnic Unwana.

1.6 LIMITATIONS OF STUDY

The researcher outlined some of the limitations as follows

- i. Incomplete data from data collation officers.
- ii. Wrong data input from technical user.
- iii. Wrong constraint specification.

1.7 DEFINITION OF TERMS

Allocate	To set apart for a specific purpose
Android	This is a mobile operating system based on the Linux Kernel and currently developed by Google.
Backend Application	Serves indirectly in support of front-end services, usually by being closer to the required

resource or having the capability to communicate with the required resource.

Frontend Application This is an application that users interact with directly.

Genetic Algorithm GA is a model of machine learning which derived its behavior from metaphor of the process (es) of EVOLUTION in natural sciences.

Google Cloud Messaging GCM is a tool from Google that allows developers to send data from their server(s) to users' device(s) and receive message(s) from devices on the same connection.

JavaScript Object Notation JSON is a lightweight data-interchange format. It is easy for human to read and write. It is a collection of name/value pairs.

Hypertext Preprocessor PHP is a server-side scripting language used in building dynamic content for the web.

Extensible Markup Language XML is a language used in designing android layouts.

Timetable

This is a table of events arranged according to the time when they take place.

CHAPTER TWO

REVIEW OF RELATED LITERATURE

A timetable is an organized list, usually set out in tabular form, providing information about a series of arranged events in particular, the time at which it is planned these events will take place. They are applicable to any institution where activities have to be carried out by various individuals in a specified time frame. From the time schools became organized environments, timetables have been the framework for all school activities. As a result, schools have devoted time, energy and human capital to the implementation of nearly optimal timetables which must be to satisfy all required constraints as specified by participating entities (Robertus, 2002).

The lecture timetabling problem is a typical scheduling problem that appears to be a tedious job in every academic institute once or twice a year. The problem involves the scheduling of classes, students, teachers and rooms at a fixed number of time-slots, subject to a certain number of constraints. An effective timetable is crucial for the satisfaction of educational requirements and the efficient utilization of human and space resources, which make it an optimization problem. Traditionally, the problem is solved manually by trial and hit method, where a valid solution is not guaranteed. Even if a valid solution is found, it is likely to miss far better solutions. These uncertainties have motivated for the scientific study of the problem, and to develop an automated solution technique for it. The

problem is being studied for last more than four decades, but a general solution technique for it is yet to be formulated (Datta D. et.al, 2006).

Timetabling problem is one of the hardest problem areas already proven to NP-complete and it is worthy of note that as educational institutions are challenged to grow in number and complexity, their resources and events are becoming harder to schedule (Ossam Chohan, 2009).

2.1 REVIEW OF RELEVANT THEORIEDS AND TECHNOLOGIES

Solutions to timetabling problems have been proposed since the 1980s. Research in this area is still active as there are several recent related papers in operational research and artificial intelligence journals. This indicates that there are many problems in timetabling that need to be solved in view of the availability of more powerful computing facilities and advancement of information technology (S.B. Deris et.al, 1997).

The problem was first studied by Gotlieb (1962), who formulated a class-teacher timetabling problem by considering that each lecture contained one group of students, one teacher, and any number of times which could be chosen freely. Since then the problem is being continuously studied using different methods under different conditions. Initially it was mostly applied to schools (de Gans, 1981; Tripathy, 1984). Since the problem in schools is relatively simple because of their simple class structures, classical methods, such as linear or integer programming approaches (Lawrie, 1969; Tripathy, 1984), could be used easily.

However, the gradual consideration of the cases of higher secondary schools and universities, which contain different types of complicated class-structures, is increasing the complexity of the problem. As a result, classical methods have been found inadequate to handle the problem, particularly the huge number of integer and/or real variables, discrete search space and multiple objective functions.

This inadequacy of classical methods has drawn the attention of the researchers towards the heuristic-based non-classical techniques. Worth mentioning non-classical techniques that are being applied to the problem are Genetic Algorithms (Alberto Colomi et al., 1992), Neural Network (Looi C., 1992), and Tabu Search Algorithm (Costa D., 1994). However, compared to other non-classical methods, the widely used are the genetic/evolutionary algorithms (GAs/EAs). The reason might be their successful implementation in a wider range of applications. Once the objectives and constraints are defined, EAs appear to offer the ultimate free lunch scenario of good solutions by evolving without a problem solving strategy (Al-Attar A., 1994). A few worth mentioning EAs, used for the school timetabling problem, are those of Abramson et al. (1992), Piola R.(1994), and Bufe et al. (2001). Similarly, EAs, used for the university class timetabling problem, are those of Carrasco et al. (2001), Srinivasan et al. (2002) and Datta et al...

Since 1995, a large amount of timetabling research has been presented in the series of international conferences on Practice and Theory of Automated

Timetabling (PATAT). Papers on this research have been published in conference proceedings, see e.g., (Burke & Carter, 1997) and (Burke & Erben, 2000), and three volumes of selected papers in the Lecture Notes in Computer Science series, see (Burke & Ross, 1996), (Burke & Carter, 1998), and (Burke & Erben, 2001). Additionally, there is a EURO working group on automated timetabling (EURO-WATT) which meets once a year regularly sends out a digest via e-mail, and maintains a website with relevant information on timetabling problems, e.g., a bibliography and several benchmarks.

Fang (1994), in his doctoral thesis, investigates the use of genetic algorithms to solve a group of timetabling problems. He presents a framework for the utilization of genetic algorithms in solving of timetabling problems in the context of learning institutions. This framework has the following important points, which give you considerable flexibility: a declaration of the specific constraints of the problem and use of a function for evaluation of the solutions, advising the use of a genetic algorithm, since it is independent of the problem, for its resolution.

Gröbner (1997) presents an approach to generalize all the timetabling problems, describing the basic structure of this problem. Gröbner proposes a generic language that can be used to describe timetabling problems and its constraints.

Chan (1997) discusses the implementation of two genetic algorithms used to solve class-teacher timetabling problem for small schools.

Oliveira (Oliveira and Reis, 2000) presents a language for representation of the timetabling problem, the UniLang. UniLang intends to be a standard suitable as

input language for any timetabling system. It enables a clear and natural representation of data, constraints, quality measures and solutions for different timetabling (as well as related) problems, such as school timetabling, university timetabling and examination scheduling.

Fernandes (2002) classified the constraints of class-teacher timetabling problem in constraints strong and weak. Violations to strong constraints (such as schedule a teacher in two classes at the same time) result in an invalid timetable. Violations to weak constraints result in valid timetable, but affect the quality of the solution (for example, the preference of teachers for certain hours). The proposed algorithm, evolutionary, has been tested in a university comprising 109 teachers, 37 rooms, 1131 a time interval of one hour each and 472 classes. The algorithm proposed in resolving the scheduling without violating the strong constraints in 30% of executions.

Eley (2006) in PATAT'06 presents a solution to the exam timetable problem, formulating it as a problem of combinatorial optimization, using algorithms Ant, to solve.

Analyzing the results obtained by the various works published, we can say what the automatic generation of schedules is capable of achieving. Some works show that when compared with the schedules manuals in institutions of learning real, the times obtained by the algorithms for solving the class-teacher timetabling problem are of better quality, since, uses some function of evaluation.

There are two main problems in timetabling. The first one is related to the combinatorial nature of the problems, where it is difficult to find an optimal solution because it is impossible to enumerate all nodes in such a large search space. The second one is related to the dynamic nature of the problems where variables and constraints are changing in accordance with the development of an organization (S.B. Deris et al., 1997). Therefore, a timetabling system must be flexible, adaptable and portable, otherwise the users will not use the system optimally or even as decision aids such as for storing, retrieving, and printing timetables, when the timetable planning decisions are made manually. In addition, most of the universities adopting a semester system give freedom to students to choose subjects provided that all pre-requisites are satisfied. This situation further complicates the construction of a timetable.

Various techniques have been proposed to solve timetabling problems. These techniques are neural networks (Gianoglio P, 1990), heuristics (Wright M, 1996), graph coloring, integer programming, Genetic Algorithms (Burke E. et al., 1994; Paechter B. et al., 1994), knowledge-based, and constraint logic programming (Lajos, 1995). The models formulated by some of these techniques cannot be easily reformulated or customized to support changes, hence the selection of the genetic algorithm for the implementation of this project.

2.2 TIMETABLING AS A NP-COMPLETE PROBLEM

In computational complexity theory, the complexity class NP-complete (abbreviated NP-C or NPC, NP standing for Nondeterministic Polynomial time) is a class of problems having two properties:

- Any given solution to the problem can be verified quickly (in polynomial time); the set of problems with this property is called NP.
- If the problem can be solved quickly (in polynomial time), then so can every problem in NP.

Although any given solution to the timetabling problem can be verified quickly, there is no known efficient way to locate a solution in the first place; indeed, the most notable characteristic of NP-complete problems is that no fast solution to them is known. That is, the time required to solve the problem using any currently known algorithm increases very quickly as the size of the problem grows (Ossam Chohan; 2009).

When solving the timetabling problem, we are usually looking for some solution, which will be the best among others. The space of all feasible solutions (series of desired solutions with some more desirable than others) is called search space (also state space). Each point in the search space represents one feasible solution which can be "marked" by its value or fitness for the problem. The solution is usually one point in the search space (Ossam Chohan; 2009).

As a result of comparative fact finding and exhaustive study of existing systems, Genetic Algorithms have been the most prominently used in generating near-

optimal solutions to timetabling problems, hence its usage in the implementation of this project.

2.3 A BRIEF HISTORY OF GENETIC ALGORITHMS

The earliest instances of what might today be called genetic algorithms appeared in the late 1950s and early 1960s, programmed on computers by evolutionary biologists who were explicitly seeking to model aspects of natural evolution. It did not occur to any of them that this strategy might be more generally applicable to artificial problems, but that recognition was not long in coming: "Evolutionary computation was definitely in the air in the formative days of the electronic computer" (Mitchell Melanie, 1996). By 1962, researchers such as G.E.P. Box, G.J. Friedman, W.W. Bledsoe and H.J. Bremermann had all independently developed evolution-inspired algorithms for function optimization and machine learning, but their work attracted little follow-up. A more successful development in this area came in 1965, when Ingo Rechenberg, then of the Technical University of Berlin, introduced a technique he called evolution strategy, though it was more similar to hill-climbers than to genetic algorithms. In this technique, there was no population or crossover; one parent was mutated to produce one offspring, and the better of the two was kept and became the parent for the next round of mutation (Haupt et. al., 1998). Later versions introduced the idea of a population. Evolution strategies are still employed today by engineers and scientists, especially in Germany.

The next important development in the field came in 1966, when L.J. Fogel, A.J. Owens and M.J. Walsh introduced in America a technique they called evolutionary programming. In this method, candidate solutions to problems were represented as simple finite-state machines; like Rechenberg's evolution strategy, their algorithm worked by randomly mutating one of these simulated machines and keeping the better of the two (Mitchell Melanie, 1996; Goldberg David, 1989). Also like evolution strategies, a broader formulation of the evolutionary programming technique is still an area of ongoing research today. However, what was still lacking in both these methodologies was recognition of the importance of crossover.

As early as 1962, John Holland's work on adaptive systems laid the foundation for later developments; most notably, Holland was also the first to explicitly propose crossover and other recombination operators. However, the seminal work in the field of genetic algorithms came in 1975, with the publication of the book *Adaptation in Natural and Artificial Systems*. Building on earlier research and papers both by Holland himself and by colleagues at the University of Michigan, this book was the first to systematically and rigorously present the concept of adaptive digital systems using mutation, selection and crossover, simulating processes of biological evolution, as a problem-solving strategy. The book also attempted to put genetic algorithms on a firm theoretical footing by introducing the notion of schemata (Mitchell Melanie, 1996; Haupt et. al., 1998). That same year, Kenneth De Jong's important dissertation established the potential of GAs

by showing that they could perform well on a wide variety of test functions, including noisy, discontinuous, and multimodal search landscapes (Goldberg David, 1989).

These foundational works established more widespread interest in evolutionary computation. By the early to mid-1980s, genetic algorithms were being applied to a broad range of subjects, from abstract mathematical problems like bin-packing and graph coloring to tangible engineering issues such as pipeline flow control, pattern recognition and classification, and structural optimization (Goldberg David, 1989).

At first, these applications were mainly theoretical. However, as research continued to proliferate, genetic algorithms migrated into the commercial sector, their rise fueled by the exponential growth of computing power and the development of the Internet. Today, evolutionary computation is a thriving field, and genetic algorithms are "solving problems of everyday interest" (Haupt et. al., 1998) in areas of study as diverse as stock market prediction and portfolio planning, aerospace engineering, microchip design, biochemistry and molecular biology, and scheduling at airports and assembly lines. The power of evolution has touched virtually any field one cares to name, shaping the world around us invisibly in countless ways, and new uses continue to be discovered as research is ongoing. And at the heart of it all lies nothing more than Charles Darwin's simple, powerful insight: that the random chance of variation, coupled with the

law of selection, is a problem-solving technique of immense power and nearly unlimited application.

Genetic algorithms (GAs) are numerical optimization algorithms that are as a result of both natural selection and natural genetics. The method which is general in nature is capable of being applied to a wider range of problems unlike most procedural approaches. Genetic algorithms help to solve practical problems on a daily basis. The algorithms are simple to understand and the required computer code easy to write. The Genetic Algorithm (GA) technique has never attracted much attention like the artificial neural networks, hill climbing, simulate annealing amongst many others although it has a growing number of disciples. The reason for this is certainly not because of any inherent limits it has or its lack of powerful metaphors. The phenomenon that evolution is the concept resulting in the bio-diversity we see around us today is a powerful and inspiring paradigm for solving any complex problem. The use of GAs have been evident from the very beginning characterized by examples of computer scientists having visions of systems that mimics and duplicate one or more of the attributes of life. The idea of using a population of solutions to solve practical engineering optimization problems was considered several times during the 1950's and 1960's. However, the concept of GAs were essentially invented by one man—John Holland—in the 1960's. His reasons for developing such algorithms were to solve problems of generalized concerns. He itemized this concept in his book in 1975, *Adaptation in Natural and Artificial Systems* (recently re-issued with additions) which is

particularly worth reading for its visionary approach. Its application has proven it to be more than just a robust method for estimating a series of unknown parameters within a model of a physical system (David, 1999).

However its robustness cuts across many different practical optimization problems especially those that concern us most like the timetable problem in the context of this project.

2.4 BASIS FOR A GENETIC ALGORITHM

1. A number, or population, of guesses of the solution to the problem.
2. A way of determining the states of generated solutions i.e. calculating how well or bad the individual solutions within the population are.
3. A method for mixing fragments of the better solutions to form new, on average even better solutions.
4. A mutation operator to avoid permanent loss of diversity within the solutions.

Concisely stated, a genetic algorithm is a programming technique that mimics biological evolution as a problem-solving strategy. Given a specific problem to solve, the input to the GA is a set of potential solutions to that problem, encoded in some fashion, and a metric called a fitness function that allows each candidate to be quantitatively evaluated. These candidates may be solutions already known to work, with the aim of the GA being to improve them, but more often they are generated at random.

The GA then evaluates each candidate according to the fitness function. In a pool of randomly generated candidates, of course, most will not work at all, and these will be deleted. However, purely by chance, a few may hold promise - they may show activity, even if only weak and imperfect activity, toward solving the problem.

These promising candidates are kept and allowed to reproduce. Multiple copies are made of them, but the copies are not perfect; random changes are introduced during the copying process. These digital offspring then go on to the next generation, forming a new pool of candidate solutions, and are subjected to a second round of fitness evaluation. Those candidate solutions which were worsened, or made no better, by the changes to their code are again deleted; but again, purely by chance, the random variations introduced into the population may have improved some individuals, making them into better, more complete or more efficient solutions to the problem at hand. Again these winning individuals are selected and copied over into the next generation with random changes, and the process repeats. The expectation is that the average fitness of the population will increase each round, and so by repeating this process for hundreds or thousands of rounds, very good solutions to the problem can be discovered.

As astonishing and counterintuitive as it may seem to some, genetic algorithms have proven to be an enormously powerful and successful problem-solving strategy, dramatically demonstrating the power of evolutionary principles.

Genetic algorithms have been used in a wide variety of fields to evolve solutions to problems as difficult as or more difficult than those faced by human designers. Moreover, the solutions they come up with are often more efficient, more elegant, or more complex than anything comparable a human engineer would produce. In some cases, genetic algorithms have come up with solutions that baffle the programmers who wrote the algorithms in the first place (Adam, 2004).

2.5 METHODS OF REPRESENTATION

- Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form that a computer can process. One common approach is to encode solutions as binary strings: sequences of 1's and 0's, where the digit at each position represents the value of some aspect of the solution (Fleming et. al., 2002).
- Another, similar approach is to encode solutions as arrays of integers or decimal numbers, with each position again representing some particular aspect of the solution. This approach allows for greater precision and complexity than the comparatively restricted method of using binary numbers only and often "is intuitively closer to the problem space" (Fleming et. al., 2002).
- A third approach is to represent individuals in a GA as strings of letters, where each letter again stands for a specific aspect of the solution. One example of this technique is Hiroaki Kitano's "grammatical encoding" approach, where a GA was put to the task of evolving a simple set of rules

called a context-free grammar that was in turn used to generate neural networks for a variety of problems (Mitchell, 1996).

The advantage of the three methods above is that they make it easy to define operators that cause the random changes in the selected candidates: flip a 0 to a 1 or vice versa, add or subtract from the value of a number by a randomly chosen amount, or change one letter to another.

- Another strategy, developed principally by John Koza of Stanford University and called genetic programming, represents programs as branching data structures called trees (Koza et. al., 2003). In this approach, random changes can be brought about by changing the operator or altering the value at a given node in the tree, or replacing one sub-tree with another.

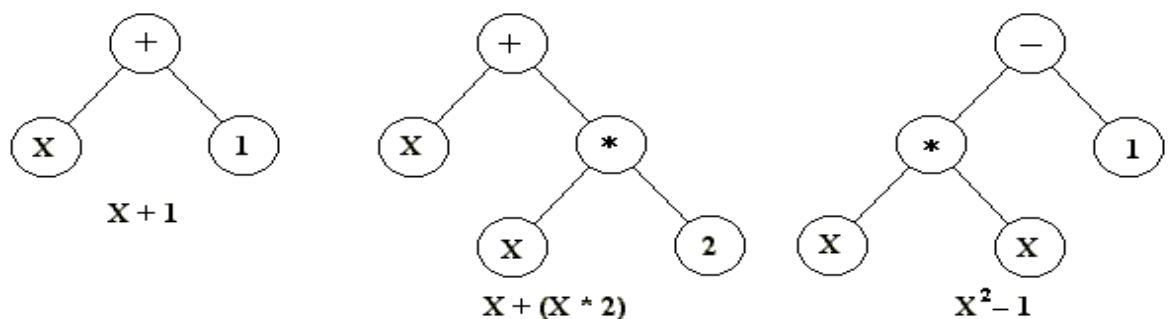


Figure 2.5: Three simple program trees of the kind normally used in genetic programming. The mathematical expression that each one represents is given underneath it (Adapted from Adam Marczyk 2004).

It is important to note that evolutionary algorithms do not necessarily represent candidate solutions as data strings of fixed length. Though some represent them this way, but others do not; e.g. Kitano's grammatical encoding discussed above

can be efficiently scaled to create large and complex neural networks, and Koza's genetic programming trees can grow arbitrarily large as necessary to solve whatever problem they are applied to.

2.6 METHODS OF SELECTION

There are many different techniques which a genetic algorithm can use to select the individuals to be copied over into the next generation, but listed below are some of the most common methods. Some of these methods are mutually exclusive, but others can be and often are used in combination.

- **Elitist selection:** The fittest members of each generation are guaranteed to be selected. (Most GAs doesn't use pure elitism, but instead use a modified form where the single best or a few of the best individuals from each generation are copied into the next generation just in case nothing better turns up.)
- **Fitness-proportionate selection:** More fit individuals are more likely, but not certain, to be selected.
- **Roulette-wheel selection:** A form of fitness-proportionate selection in which the chance of an individual's being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness. (Conceptually, this can be represented as a game of roulette - each individual gets a slice of the wheel, but more fit ones get larger slices than less fit ones. The wheel is then spun, and whichever individual "owns" the section on which it lands each time is chosen).

- **Scaling selection:** As the average fitness of the population increases, the strength of the selective pressure also increases and the fitness function becomes more discriminating. This method can be helpful in making the best selection later on when all individuals have relatively high fitness and only small differences in fitness distinguish one from another.
- **Tournament selection:** Subgroups of individuals are chosen from the larger population, and members of each subgroup compete against each other. Only one individual from each subgroup is chosen to reproduce.
- **Rank selection:** Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking rather than absolute difference in fitness. The advantage of this method is that it can prevent very fit individuals from gaining dominance early at the expense of less fit ones, which would reduce the population's genetic diversity and might hinder attempts to find an acceptable solution.
- **Generational selection:** The offspring of the individuals selected from each generation become the entire next generation. No individuals are retained between generations.
- **Steady-state selection:** The offspring of the individuals selected from each generation go back into the pre-existing gene pool, replacing some of the less fit members of the previous generation. Some individuals are retained between generations.

- **Hierarchical selection:** Individuals go through multiple rounds of selection each generation. Lower-level evaluations are faster and less discriminating, while those that survive to higher levels are evaluated more rigorously. The advantage of this method is that it reduces overall computation time by using faster, less selective evaluation to weed out the majority of individuals that show little or no promise, and only subjecting those who survive this initial test to more rigorous and more computationally expensive fitness evaluation.

2.7 METHODS OF CHANGE

- Once selection has chosen fit individuals, they must be randomly altered in hopes of improving their fitness for the next generation. There are two basic strategies to accomplish this. The first and simplest is called **mutation**. Just as mutation in living things changes one gene to another, so mutation in a genetic algorithm causes small alterations at single points in an individual's code. Refer to Figure.



Figure 2.7a: Diagram showing the effect of mutation on an individual in a population of 8-bit strings where mutation occurs at position 4, changing the 0 at that position in its genome to a 1 (Adapted from Adam Marczyk 2004).

- The second method is called **crossover**, and entails choosing two individuals to swap segments of their code, producing artificial "offspring" that are combinations of their parents. This process is intended to simulate the analogous process of recombination that occurs to chromosomes during sexual reproduction (Adam, 2004). Common forms of crossover include single-point crossover, in which a point of exchange is set at a random location in the two individuals' genomes, and one individual contributes all its code from before that point and the other contributes all its code from after that point to produce an offspring, and uniform crossover, in which the value at any given location in the offspring's genome is either the value of one parent's genome at that location or the value of the other parent's genome at that location, chosen with 50/50 probability. Refer to Figure 2.7b.

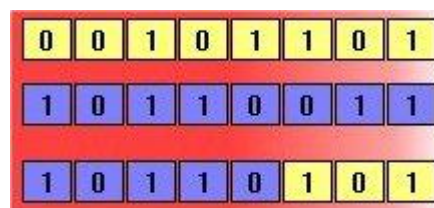


Figure 2.7b: Diagram showing the effect of mutation on individuals in a population of 8-bit strings showing two individuals undergoing single-point crossover; the point of exchange is set between the fifth and sixth positions in the genome, producing a new individual that is a hybrid of its progenitors (Adapted from Adam Marczyk 2004).

2.8 STRENGTHS OF GENETIC ALGORITHMS

- The first and most important point is that genetic algorithms are intrinsically parallel. Most other algorithms are serial and can only explore the solution space to a problem in one direction at a time, and if the solution they discover turns out to be suboptimal, there is nothing to do but abandon all work previously completed and start over. However, since GAs have multiple offspring, they can explore the solution space in multiple directions at once. If one path turns out to be a dead end, they can easily eliminate it and continue work on more promising avenues, giving them a greater chance each run of finding the optimal solution (Adam, 2004; John, 1992).
- However, the advantage of parallelism goes beyond this. Consider the following: All the 8-digit binary strings (strings of 0's and 1's) form a search space, which can be represented as `*****` (where the `*` stands for "either 0 or 1"). The string `01101010` is one member of this space. However, it is also a member of the space `0*****`, the space `01*****`, the space `0*****0`, the space `0*1*1*1*`, the space `01*01**0`, and so on. By evaluating the fitness of this one particular string, a genetic algorithm would be sampling each of these many spaces to which it belongs. Over many such evaluations, it would build up an increasingly accurate value for the average fitness of each of these spaces, each of which has many members. Therefore, a GA that explicitly evaluates a small number of

individuals is implicitly evaluating a much larger group of individuals - just as a pollster who asks questions of a certain member of an ethnic, religious or social group hopes to learn something about the opinions of all members of that group, and therefore can reliably predict national opinion while sampling only a small percentage of the population. In the same way, the GA can "home in" on the space with the highest-fitness individuals and find the overall best one from that group. In the context of evolutionary algorithms, this is known as the Schema Theorem, and is the "central advantage" of a GA over other problem-solving methods (John, 1992; Mitchell, 1996; Goldberg, 1989).

- Due to the parallelism that allows them to implicitly evaluate many schemas at once, genetic algorithms are particularly well-suited to solving problems where the space of all potential solutions is truly huge - too vast to search exhaustively in any reasonable amount of time. Most problems that fall into this category are known as "nonlinear". In a linear problem, the fitness of each component is independent, so any improvement to any one part will result in an improvement of the system as a whole. Needless to say, few real-world problems are like this. Nonlinearity is the norm, where changing one component may have ripple effects on the entire system, and where multiple changes that individually are detrimental may lead to much greater improvements in fitness when combined. Nonlinearity results in a combinatorial explosion: the space of 1,000-digit binary strings

can be exhaustively searched by evaluating only 2,000 possibilities if the problem is linear, whereas if it is nonlinear, an exhaustive search requires evaluating 21000 possibilities - a number that would take over 300 digits to write out in full (Adam, 2004).

- Fortunately, the implicit parallelism of a GA allows it to surmount even this enormous number of possibilities, successfully finding optimal or very good results in a short period of time after directly sampling only small regions of the vast fitness landscape (Forrest, 1993). For example, a genetic algorithm developed jointly by engineers from General Electric and Rensselaer Polytechnic Institute produced a high-performance jet engine turbine design that was three times better than a human-designed configuration and 50% better than a configuration designed by an expert system by successfully navigating a solution space containing more than 10387 possibilities. Conventional methods for designing such turbines are a central part of engineering projects that can take up to five years and cost over \$2 billion; the genetic algorithm discovered this solution after two days on a typical engineering desktop workstation (John, 1992).
- Another notable strength of genetic algorithms is that they perform well in problems for which the fitness landscape is complex - ones where the fitness function is discontinuous, noisy, changes over time, or has many local optima. Most practical problems have a vast solution space, impossible to search exhaustively; the challenge then becomes how to

avoid the local optima - solutions that are better than all the others that are similar to them, but that are not as good as different ones elsewhere in the solution space. Many search algorithms can become trapped by local optima: if they reach the top of a hill on the fitness landscape, they will discover that no better solutions exist nearby and conclude that they have reached the best one, even though higher peaks exist elsewhere on the map (Adam, 2004).

- Evolutionary algorithms, on the other hand, have proven to be effective at escaping local optima and discovering the global optimum in even a very rugged and complex fitness landscape. (It should be noted that, in reality, there is usually no way to tell whether a given solution to a problem is the one global optimum or just a very high local optimum. However, even if a GA does not always deliver a provably perfect solution to a problem, it can almost always deliver at least a very good solution.) All four of a GA's major components - parallelism, selection, mutation, and crossover - work together to accomplish this. In the beginning, the GA generates a diverse initial population, casting a "net" over the fitness landscape. (Koza et. al., 2003) compares this to an army of parachutists dropping onto the landscape of a problem's search space, with each one being given orders to find the highest peak.) Small mutations enable each individual to explore its immediate neighborhood, while selection focuses progress, guiding the

algorithm's offspring uphill to more promising parts of the solution space (John, 1992).

- However, crossover is the key element that distinguishes genetic algorithms from other methods such as hill-climbers and simulated annealing. Without crossover, each individual solution is on its own, exploring the search space in its immediate vicinity without reference to what other individuals may have discovered. However, with crossover in place, there is a transfer of information between successful candidates - individuals can benefit from what others have learned, and schemata can be mixed and combined, with the potential to produce an offspring that has the strengths of both its parents and the weaknesses of neither. This point is illustrated in Koza et.al. (1999), where the authors discuss a problem of synthesizing a low pass filter using genetic programming. In one generation, two parent circuits were selected to undergo crossover; one parent had good topology (components such as inductors and capacitors in the right places) but bad sizing (values of inductance and capacitance for its components that were far too low). The other parent had bad topology, but good sizing. The result of mating the two through crossover was an offspring with the good topology of one parent and the good sizing of the other, resulting in a substantial improvement in fitness over both its parents.

- The problem of finding the global optimum in a space with many local optima is also known as the dilemma of exploration vs. exploitation, "a classic problem for all systems that can adapt and learn" (John, 1992). Once an algorithm (or a human designer) has found a problem-solving strategy that seems to work satisfactorily, should it concentrate on making the best use of that strategy, or should it search for others? Abandoning a proven strategy to look for new ones is almost guaranteed to involve losses and degradation of performance, at least in the short term. But if one sticks with a particular strategy to the exclusion of all others, one runs the risk of not discovering better strategies that exist but have not yet been found. Again, genetic algorithms have shown themselves to be very good at striking this balance and discovering good solutions with a reasonable amount of time and computational effort (Adam, 2004).
- Another area in which genetic algorithms excel is their ability to manipulate many parameters simultaneously (Forrest, 1993). Many real-world problems cannot be stated in terms of a single value to be minimized or maximized, but must be expressed in terms of multiple objectives, usually with tradeoffs involved: one can only be improved at the expense of another. GAs are very good at solving such problems: in particular, their use of parallelism enables them to produce multiple equally good solutions to the same problem, possibly with one candidate solution optimizing one parameter and another candidate optimizing a different one (Haupt et.al.,

1998), and a human overseer can then select one of these candidates to use.

If a particular solution to a multi-objective problem optimizes one parameter to a degree such that that parameter cannot be further improved without causing a corresponding decrease in the quality of some other parameter, that solution is called Pareto optimal or non-dominated (Coello, 2000).

- Finally, one of the qualities of genetic algorithms which might at first appear to be a liability turns out to be one of their strengths: namely, GAs know nothing about the problems they are deployed to solve. Instead of using previously known domain-specific information to guide each step and making changes with a specific eye towards improvement, as human designers do, they are "blind watchmakers" (Dawkins, 1996); they make random changes to their candidate solutions and then use the fitness function to determine whether those changes produce an improvement.
- The virtue of this technique is that it allows genetic algorithms to start out with an open mind, so to speak. Since its decisions are based on randomness, all possible search pathways are theoretically open to a GA; by contrast, any problem-solving strategy that relies on prior knowledge must inevitably begin by ruling out many pathways a priori, therefore missing any novel solutions that may exist there (Koza et. al., 1999). Lacking preconceptions based on established beliefs of "how things should be done" or what "couldn't possibly work", GAs do not have this problem.

Similarly, any technique that relies on prior knowledge will break down when such knowledge is not available, but again, GAs is not adversely affected by ignorance (Goldberg, 1989). Through their components of parallelism, crossover and mutation, they can range widely over the fitness landscape, exploring regions which intelligently produced algorithms might have overlooked, and potentially uncovering solutions of startling and unexpected creativity that might never have occurred to human designers. One vivid illustration of this is the rediscovery, by genetic programming, of the concept of negative feedback - a principle crucial to many important electronic components today, but one that, when it was first discovered, was denied a patent for nine years because the concept was so contrary to established beliefs (Koza et. al., 2003). Evolutionary algorithms, of course, are neither aware nor concerned whether a solution runs counter to established beliefs - only whether it works.

2.9 LIMITATIONS OF GENETIC ALGORITHMS

Although genetic algorithms have proven to be an efficient and powerful problem-solving strategy, they are not a panacea. GAs does have certain limitations which are outlined below:

- The first, and most important, consideration in creating a genetic algorithm is defining a representation for the problem. The language used to specify candidate solutions must be robust; i.e., it must be able to tolerate random changes such that fatal errors or nonsense do not consistently result.

There are two main ways of achieving this. The first, which is used by most genetic algorithms, is to define individuals as lists of numbers - binary-valued, integer-valued, or real-valued - where each number represents some aspect of a candidate solution. If the individuals are binary strings, 0 or 1 could stand for the absence or presence of a given feature. If they are lists of numbers, these numbers could represent many different things: the weights of the links in a neural network, the order of the cities visited in a given tour, the spatial placement of electronic components, the values fed into a controller, the torsion angles of peptide bonds in a protein, and so on. Mutation then entails changing these numbers, flipping bits or adding or subtracting random values. In this case, the actual program code does not change; the code is what manages the simulation and keeps track of the individuals, evaluating their fitness and perhaps ensuring that only values realistic and possible for the given problem result.

In another method, genetic programming, the actual program code does change. As discussed in the section Methods of representation, GP represents individuals as executable trees of code that can be mutated by changing or swapping sub-trees. Both of these methods produce representations that are robust against mutation and can represent many different kinds of problems, and both have had considerable success in various examples on which they have been applied.

This issue of representing candidate solutions in a robust way does not arise in nature, because the method of representation used by evolution, namely the genetic code, is inherently robust: with only a very few exceptions, such as a string of stop codons, there is no such thing as a sequence of DNA bases that cannot be translated into a protein. Therefore, virtually any change to an individual's genes will still produce an intelligible result, and so mutations in evolution have a higher chance of producing an improvement. This is in contrast to human-created languages such as English, where the number of meaningful words is small compared to the total number of ways one can combine letters of the alphabet, and therefore random changes to an English sentence are likely to produce nonsense (Adam, 2004).

- The problem of how to write the fitness function must be carefully considered so that higher fitness is attainable and actually does equate to a better solution for the given problem. If the fitness function is chosen poorly or defined imprecisely, the genetic algorithm may be unable to find a solution to the problem, or may end up solving the wrong problem. (This latter situation is sometimes described as the tendency of a GA to "cheat", although in reality all that is happening is that the GA is doing what it was told to do, not what its creators intended it to do.) This is not a problem in nature, however. In the laboratory of biological evolution there is only one fitness function, which is the same for all living things - the drive to survive

and reproduce, no matter what adaptations make this possible. Those organisms which reproduce more abundantly compared to their competitors are fitter; those which fail to reproduce are unfit (Adam, 2004).

- In addition to making a good choice of fitness function, the other parameters of a GA - the size of the population, the rate of mutation and crossover, the type and strength of selection - must be also chosen with care. If the population size is too small, the genetic algorithm may not explore enough of the solution space to consistently find good solutions. If the rate of genetic change is too high or the selection scheme is chosen poorly, beneficial schema may be disrupted and the population may enter error catastrophe, changing too fast for selection to ever bring about convergence (Adam, 2004).

Living things do face similar difficulties, and evolution has dealt with them. It is true that if a population size falls too low, mutation rates are too high, or the selection pressure is too strong (such a situation might be caused by drastic environmental change), then the species may go extinct. The solution has been "the evolution of evolvability" - adaptations that alter a species' ability to adapt. For example, most living things have evolved elaborate molecular machinery that checks for and corrects errors during the process of DNA replication, keeping their mutation rate down to acceptably low levels; conversely, in times of severe environmental stress, some bacterial species enter a state of hyper mutation where the rate of

DNA replication errors rises sharply, increasing the chance that a compensating mutation will be discovered. Of course, not all catastrophes can be evaded, but the enormous diversity and highly complex adaptations of living things today show that, in general, evolution is a successful strategy. Likewise, the diverse applications of and impressive results produced by genetic algorithms show them to be a powerful and worthwhile field of study (John, 1975).

- One type of problem that genetic algorithms have difficulty dealing with are problems with "deceptive" fitness functions (Mitchell, 1996), those where the locations of improved points give misleading information about where the global optimum is likely to be found. For example, imagine a problem where the search space consisted of all eight-character binary strings, and the fitness of an individual was directly proportional to the number of 1s in it - i.e., 00000001 would be less fit than 00000011, which would be less fit than 00000111, and so on - with two exceptions: the string 11111111 turned out to have very low fitness, and the string 00000000 turned out to have very high fitness. In such a problem, a GA (as well as most other algorithms) would be no more likely to find the global optimum than random search.

The resolution to this problem is the same for both genetic algorithms and biological evolution: evolution is not a process that has to find the single global optimum every time. It can do almost as well by reaching the top of

a high local optimum, and for most situations, this will suffice, even if the global optimum cannot easily be reached from that point. Evolution is very much a "satisfier" - an algorithm that delivers a "good enough" solution, though not necessarily the best possible solution, given a reasonable amount of time and effort invested in the search. The Evidence for Jury-Rigged Design in Nature FAQ gives examples of this very outcome appearing in nature. (It is also worth noting that few, if any, real-world problems are as fully deceptive as the somewhat contrived example given above. Usually, the location of local improvements gives at least some information about the location of the global optimum.)

- One well-known problem that can occur with a GA is known as premature convergence. If an individual that is more fit than most of its competitors emerges early on in the course of the run, it may reproduce so abundantly that it drives down the population's diversity too soon, leading the algorithm to converge on the local optimum that that individual represents rather than searching the fitness landscape thoroughly enough to find the global optimum (Forrest, 1993; Mitchell, 1996). This is an especially common problem in small populations, where even chance variations in reproduction rate may cause one genotype to become dominant over others.

The most common methods implemented by GA researchers to deal with this problem all involve controlling the strength of selection, so as not to

give excessively fit individuals too great of an advantage. Rank, scaling and tournament selection, discussed earlier, are three major means for accomplishing this; some methods of scaling selection include sigma scaling, in which reproduction is based on a statistical comparison to the population's average fitness, and Boltzmann selection, in which the strength of selection increases over the course of a run in a manner similar to the "temperature" variable in simulated annealing (Mitchell, 1996).

Premature convergence does occur in nature (where it is called genetic drift by biologists). This should not be surprising; as discussed above, evolution as a problem-solving strategy is under no obligation to find the single best solution, merely one that is good enough. However, premature convergence in nature is less common since most beneficial mutations in living things produce only small, incremental fitness improvements; mutations that produce such a large fitness gain as to give their possessors dramatic reproductive advantage are rare.

- Finally, several researchers (John, 1992; Forrest, 1993; Haupt et. al., 1998) advise against using genetic algorithms on analytically solvable problems. It is not that genetic algorithms cannot find good solutions to such problems; it is merely that traditional analytic methods take much less time and computational effort than GAs and, unlike GAs, are usually mathematically guaranteed to deliver the one exact solution. Of course,

since there is no such thing as a mathematically perfect solution to any problem of biological adaptation, this issue does not arise in nature.

2.10 APPLICATION OF GENETIC ALGORITHMS IN THIS RESEARCH

Having considered the basis for a genetic algorithm, the outline below highlights the applications of the proposed system in generating timetables (with Department of Computer Science, Akanu Ibiam Federal Polytechnic, Unwana as case study).

The timetabling problem is a combinatorial optimization problem (COP) and in order to find a very comprehensive mathematical framework to describe it (and also tackle its NP-hardness), hence the introduction of a highly abstract concept of heuristics (genetic algorithms). The basic property of the timetable problem is the attempt of the genetic algorithm to optimize a function over a discrete structure with many independent variables. The relation between the choices made in the discrete domain and the effects on the objective function value are usually complex and frequently not easy to trace. The unifying framework for COP's is the Constraint Satisfaction Problem (CSP) in conjunction with the optimization of an objective function (Kostuch, 2003). It is worthy of note that even though the timetabling problem is treated as an optimization problem, there is actually no fixed objective function, the function that exists is used as an arbitrary measure to check for optimized solutions and degree of constraints satisfaction (Abramson, 1992). Once the objectives and constraints are specified,

genetic algorithms offer the ultimate scenarios of good timetable solutions through evolution processes even though the complexity of assignment is totally dependent on the number of instances and number of constraints.

Hence the algorithm considered for use in the proposed system is a scaled down version of the Hybrid Genetic algorithm for the construction of examination timetables developed for the University of Nottingham. The concept though developed for examination timetabling, can be adapted to fit the construction of course timetables. The genetic algorithm employed combines two heuristic algorithms, the first finding a non-conflicting set of exams and the second assigning the selected exam to rooms. The process is repeated until all exams have been scheduled without conflicts (Rupert, 1995).

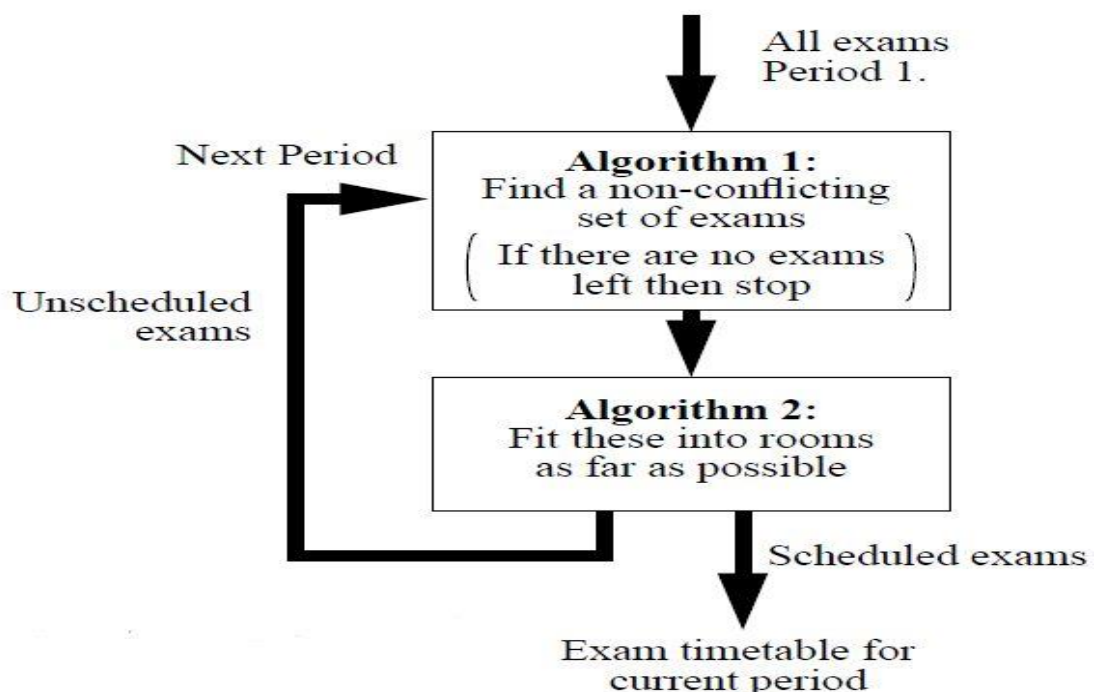


Figure 2.10: Diagram depicting the Hybrid Genetic Algorithm used in University of Nottingham (Adapted from Rupert Weare et. al. 1995).

Like every other genetic algorithms, this algorithm can quickly produce large populations of random feasible exam timetables. Uniquely, the process takes each member of the course population and assigns it to the first period in which the exam may be placed without conflict. The mutation and crossover procedures are then applied to the population so that constraints associated with each course in the assignment are satisfied. The timetables generated by the algorithm with a starting population size of 200 had an average fitness of 0.986 (Rupert Weare et. al., 1995).

CHAPTER THREE

SYSTEM INVESTIGATION AND ANALYSIS

A system can be defined as a group of components consisting of sub system or procedures that works in a coordinated fashion to achieve said objective.

This chapter reviews how the existing system works as well as to produce a better alternative for its improvement.

The relationship amongst entities and information flows within the organization is very important; in a nutshell, system investigation and analysis is the study of an existing system with the view of improving on it or developing an entirely new system to replace the existing one.

The major task here is to design a new system with a mobile frontend for students.

FUNCTIONS OF THE PERSONNEL

1. **H.O.D:** He is the head of the department. He plans and assigns work schedule for each staff and students, and he oversees their duties.
2. **SECRETARY/TYPIST:** She is a subordinate staff and responsible to the head of department. She is responsible for typing official document needed for the academic duties of the department, receiving of incoming mails, and issuing of course forms to students.
3. **LECTURERS:** A lecturer lectures students professionally.
4. **TECHNOLOGIST:** The technologist uses scientific knowledge to solve practical problems.
5. **ADMINISTRATION OFFICICER:** He is responsible for the administrative management of the office.
6. **EXECUTIVE OFFICIER:** He is also responsible for the administration of business, he has the function of carrying out plans orders etc.
7. **LAB ASSISTANT:** The lab assistant is in charge of taking care of the laboratory.
8. **CLERICAL OFFICERS/MESSENGER:** They are in charge of clerical duties and running errands for the department.

3.2FACTS FINDING

Fact finding is an approach taken to acquire data about a specific or subject with the aim of analyzing and synthesizing the analyzed data to come up with a better

system. The researcher used different techniques in data collection in this research and they include: interview, examination of existing documents, internet browsing and use of questionnaires.

A set of well-articulated questions were used to interview a number of persons.

During the investigation, some documents used for drafting the lecture timetable were examined. Some of them include:

- i. List of examinable and non-examinable courses offered by the department.
- ii. List of existing lecture halls and their carrying capacity.
- iii. List of periods or intervals.

The internet is the largest pool of information where virtually everything can be found. The researcher in addition to examining existing documents equally used the internet. Some relevant and related documents were downloaded.

The researcher used the questionnaire to take conclusion on how the user interface will be. It was developed using Google Forms.

3.3 ANALYSIS

In this section, thorough studying and analysis of the gathered data and fact were done on the existing system.

In Akanu Ibiam Federal Polytechnic Unwana, timetable scheduling is done by:

- The submission of examinable and non-examinable courses by the timetable officer of each department.
- The submitted documents are taken by the chairman space and timetable committee.
- Previous school scheduled timetable and blank templates are passed round from one department to another for the timetable officers to allocate course to halls.
- After the exercise, tentative timetables are released for staff and students to make inputs.
- At the end of receiving inputs from students and lectures alike, the final timetable is presented to the school and it will become a working document for that semester.

3.4 PROBLEMS OF THE CURRENT SYSTEM

The traditional manual generations of timetables encounters a lot of problems which may include the following:

- Repeated time allocations may be made for a particular course thereby leading to data redundancy.
- A lot of administrative error may occur as a result of confusing time requirements.
- Timetable generation by center staff may have a slow turnaround.

- Final generated timetable may not be near optimal as a result of clashing course requirements and allocations.
- It generates a lot of paperwork and is very tasking.
- It is not flexible as changes may not be easily made.

3.5 PROPOSING A NEW SYSTEM

The proposed systems was developed to solve the timetabling problem being faced by universities every academic year and reduce high cost and slow turnaround involved in the generation of near-optimal timetables.

The system has capabilities for input of the various courses, halls of lectures, departments, programs, buildings, lecturers and the specification of a few constraints from which the timetable is constructed. The proposed timetabling system for this project seeks to generate near optimal timetables using the principles of genetic algorithm (selection and crossover).

3.6 ADVANTAGES OF THE PROPOSED SYSTEM

The timetable generation process by the education center staff is:

- Unlike the manual timetabling system, the system offers flexibility.
- It utilizes minimal processing/computing power.
- It greatly reduces the time needed to generate near-optimal timetables.
- It provides an easy means for data entry and revision through an intuitive interface.
- It increases productivity.

- Timetables generated are between to 60% - 80% optimum.
- It almost eliminates paperwork.
- It simplifies the timetabling process.

CHAPTER FOUR

SYSTEM DESIGN

System design is the specification or construction of a technical, computer-based solution for the business requirements identified in system analysis. It gives the overall plan or model of a system consisting of all specification that give the system its form and structure (i.e. the structural implementation of the system analysis).

4.1 OBJECTIVES OF THE DESIGN

The objective of designing the new system is to provide an automated timetabling system that will generate a feasible timetable and broadcast to listening devices (android).

FACTORS CONSIDERED IN THE DESIGN

The following are the factors put into consideration during the design of the new system;

To design

- A user friendly system.
- A system that is able to consider all given constraints before producing a feasible (timetable) output.
- A time effective system.
- A system that will broadcast generated timetable to listening android

devices once it has been published.

4.2 SYSTEM BLOCK DIAGRAM

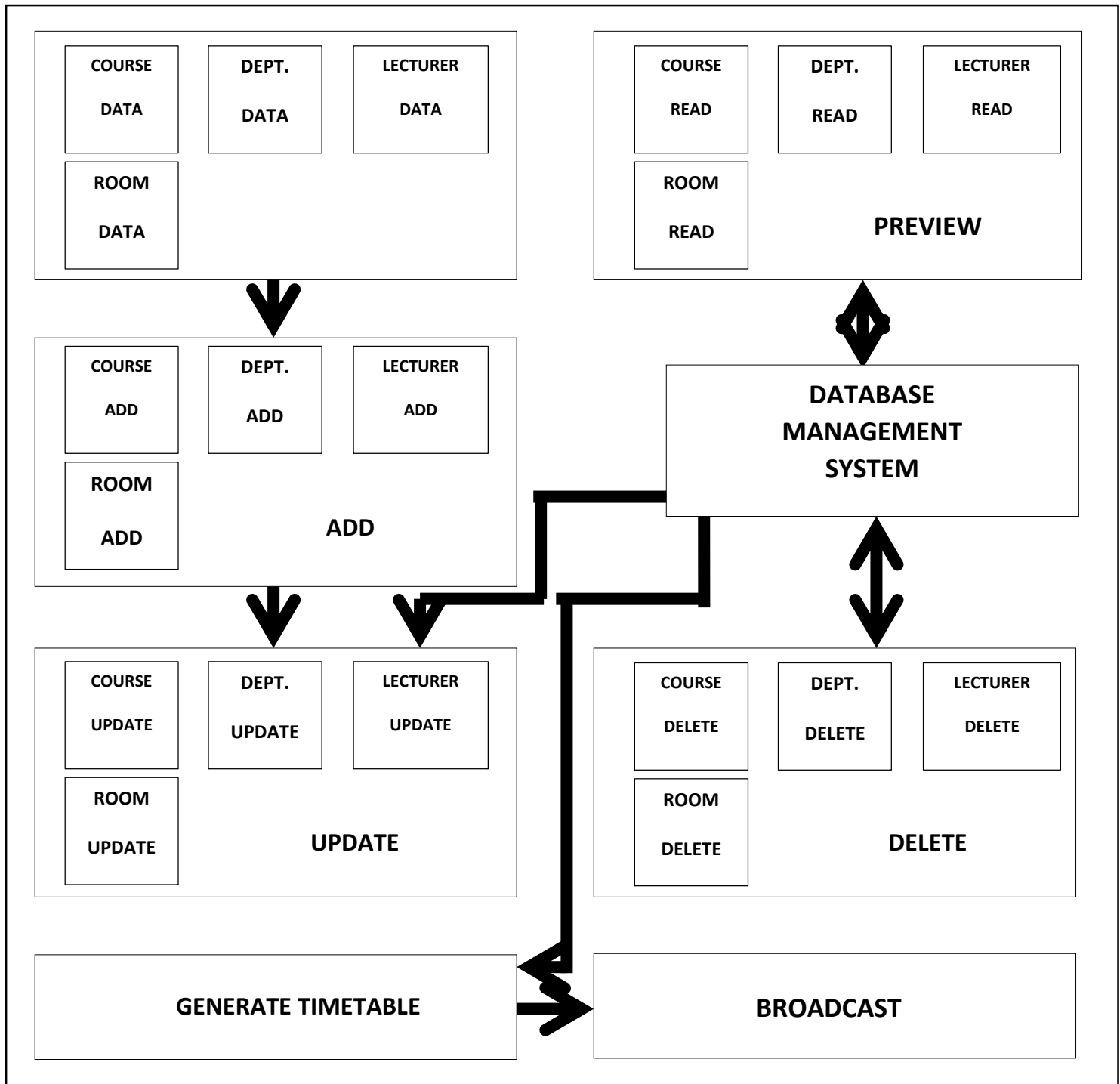


Fig 4.2: System Block Diagram

4.3 OUTPUT DESIGN

The output design contains all the necessary details of the processed input by the system which is drawn from different tables of the database. In this case, the generated timetable is the output of the new system.

4.4 INPUT DESIGN

The input design is the interaction point between the user and the system. There are three major input forms/design

- Courses input design.
- Room input design.
- Lecturer input design.
- Building input design.
- Class input design.

4.5 PROGRAM DESIGN

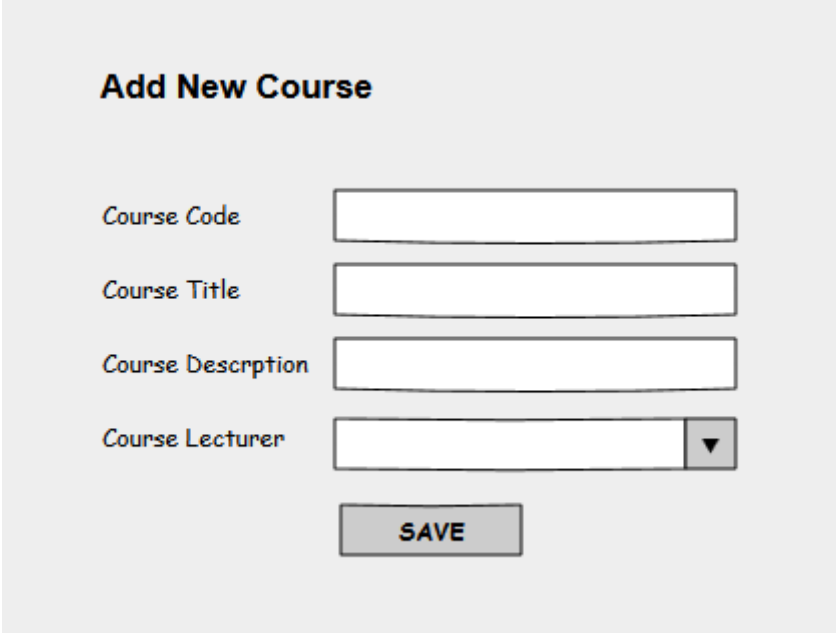
This is where the programs that will run the modules identified in the control center are specified. This will enable the researcher to capture the complete working picture of the application, how each component is related to another, the calling program modules and the ones called.

The timetabling system consists of various program modules. Each module of the system is a complete module and part of the entire system.

Below are the various modules contained in the application:

- Course Module

The course module is the interface (subsystem) that manages the courses available in the database. To create a new course, the course name (title), course code, course description and course lecturer are required fields. The course lecturer field is fetched from the lecturers table.




The image shows a web form titled "Add New Course". It contains four input fields: "Course Code", "Course Title", "Course Description", and "Course Lecturer". The "Course Lecturer" field is a dropdown menu. Below the fields is a "SAVE" button.

Add New Course	
Course Code	<input type="text"/>
Course Title	<input type="text"/>
Course Description	<input type="text"/>
Course Lecturer	<input type="text" value="▼"/>
<input type="button" value="SAVE"/>	

Fig 4.5a: Course Input Module

- Building Module

This subsystem manages the existing building on the database. To create a new building record, the building name is a required field.




The image shows a web form titled "Add New Building". It has a single text input field labeled "Building Name". Below the input field is a button labeled "SAVE".

Fig 4.5b: Building Input Module

- Room Module

The room module/subsystem manages available rooms in the database. To create a new record, the room name, building name and room capacity are required fields. The building name field is fetched from the buildings table.



The image shows a web form titled "Add New Room". It has three input fields: "Room Name" (text), "Building" (dropdown menu), and "Capacity" (text). Below the input fields is a button labeled "SAVE".

Fig 4.5b: Room Input Module

- Lecturer Module

This subsystem manages available lecturers in the database. To add a new lecturer, the lecturer name and department is required field.

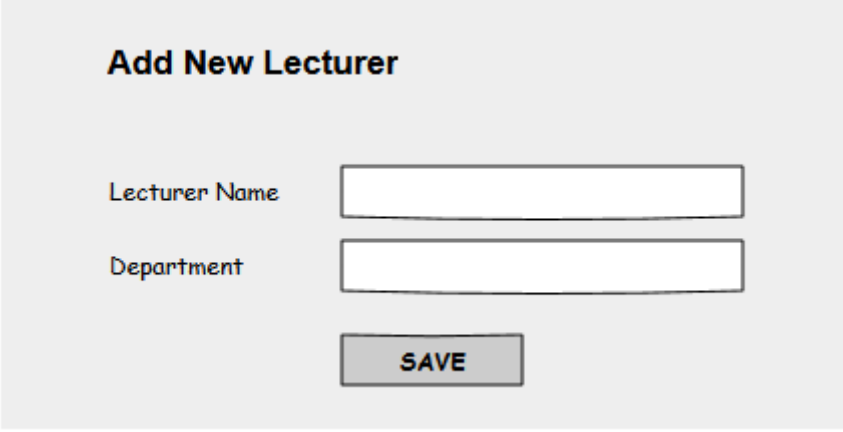
A screenshot of a web form titled "Add New Lecturer". The form is set against a light gray background. It contains two text input fields: the first is labeled "Lecturer Name" and the second is labeled "Department". Below these fields is a rectangular button with the word "SAVE" in capital letters.

Fig 4.5b: Lecturer Module

- Class Module

The class subsystem manages available classes in the database. To insert new record, the level, class size and department are required.

4.6 DATABASE

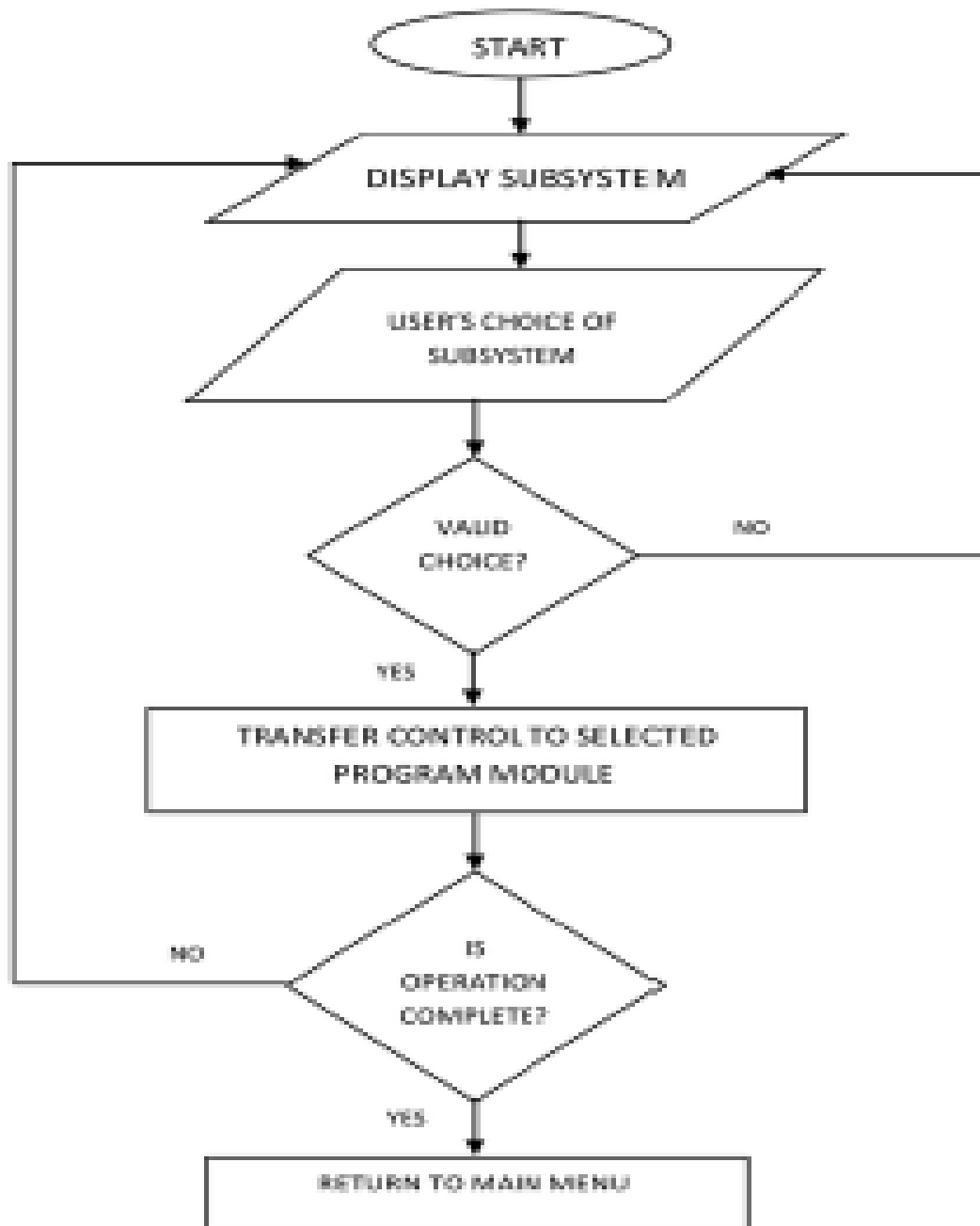
This is an organized collection and manipulation of data. The data are typically organized to model relevant aspect of reality in a way that supports the process requiring the information.

4.7 DATABASE SPECIFICATION

The program's database is built using MySQL.

4.8 PROGRAM FLOWCHART

This describes the diagrammatic representation of the logical paths to be



followed.

Fig 4.8: Program Flowchart

LOGIN MODULE

This is the first module that is loaded once the web timetable interface is launched. It accepts three information (login ID, login PIN and remember me - optional) that will be validated against the one already stored in the users database table. If the login information is valid, a new session will be created for the user and the dashboard page will be loaded, else an error message based on the offence committed will be displayed and the login page will be reloaded.

DASHBOARD

This module is loaded if and only if a session was created for the timetable officer. This is the module that housed other subsystems – Course module, Lecturer module, Rooms module and Timetable Scheduling module.

4.9 MODELING THE SYSTEM

Modeling a system is the process of abstracting and organizing significant features of how the system would look like. Modeling is the designing of the software application before coding. Unified Modeling Language (UML) tools were used in modeling the system.

UML (Unified Modeling Language)

This is the object-oriented system notation that provides a set of modeling conventions that is used to specify or describe a software system in terms of

objects. The UML has become an object modeling standard and adds a variety of techniques to the field of system analysis and development hence its choice for this project.

UML offers ten different diagrams to model a system. These diagrams are listed below:

- Use case diagram
- Class diagram
- Object diagram
- Sequence diagram
- Collaboration diagram
- State diagram
- Activity diagram
- Component diagram
- Deployment diagram
- Package Diagram

In this project, the Use case diagram, Class diagram, Sequence diagram, Activity diagram and State diagram will be used for system modeling.

USE CASE DIAGRAM

Use case diagrams describe what a system does from an external observer's standpoint. The emphasis of use case diagrams is on what a system does rather than how. They are used to show the interactions between users of the system and the system. A use case represents the several users called actors and the different ways in which they interact with the system.

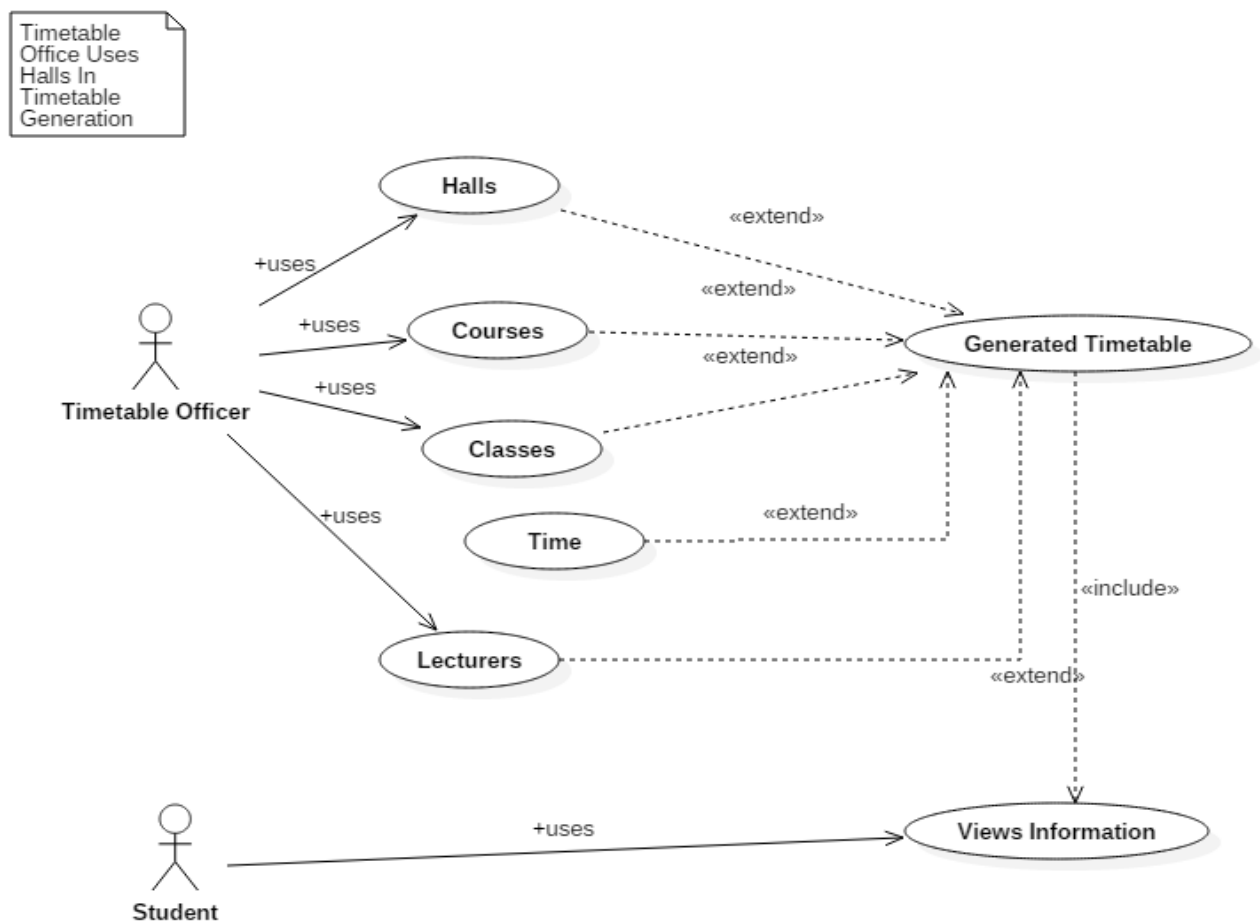


Fig 4.9a: Use Case Diagram

CLASS DIAGRAM

This is an organization of related objects. It gives an overview of a system by showing its classes and their relationships. Class diagrams only displays what interacts and not what happens during the interaction.

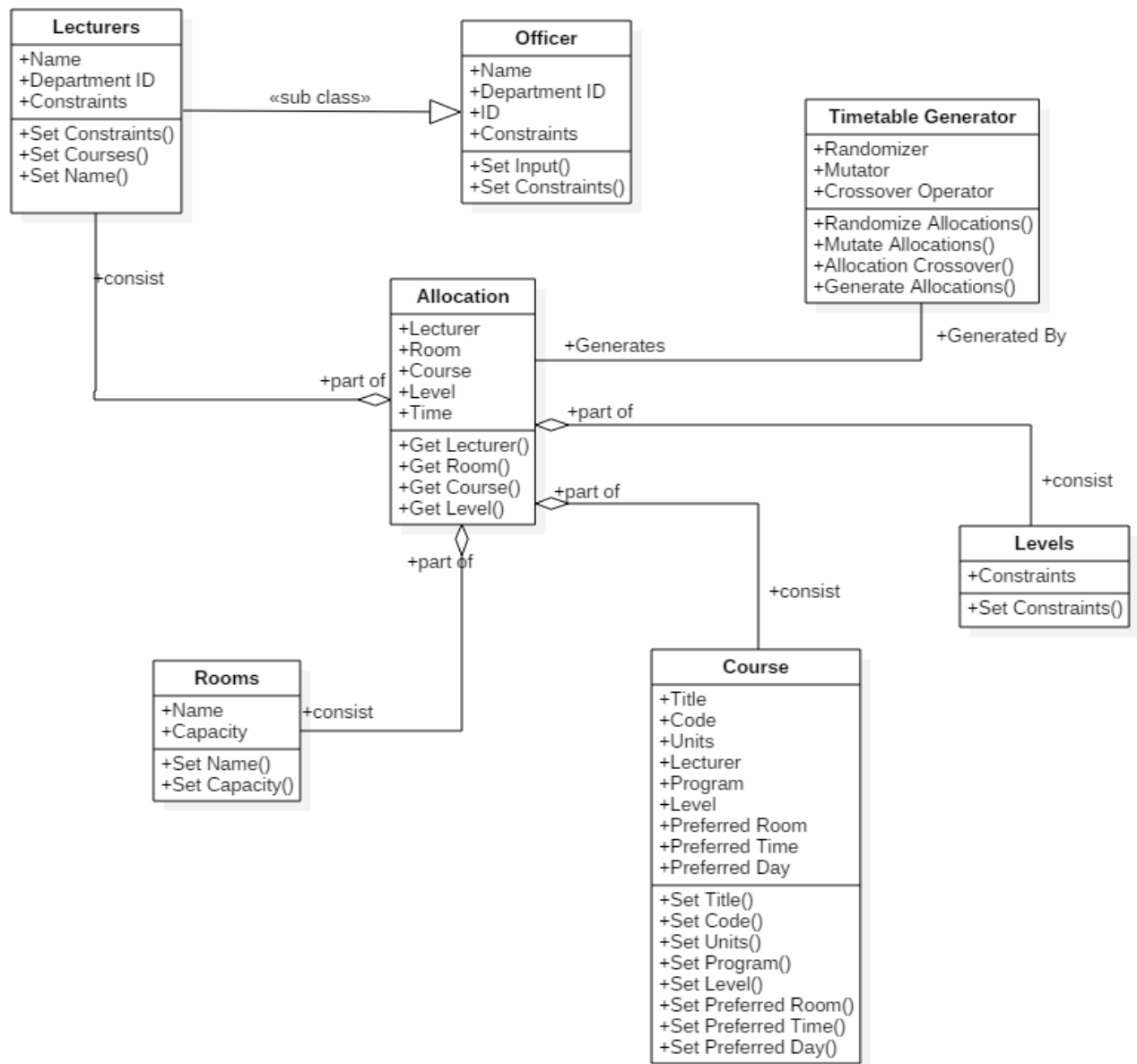


Fig 4.9b: Class Diagram

SEQUENCE DIAGRAM

This describes how objects interact with each other through messages during the execution of a use case or any operation. They illustrate how messages are sent and received between objects and the sequence of message transfer. It also describes how operations are carried out according to the time of operation.

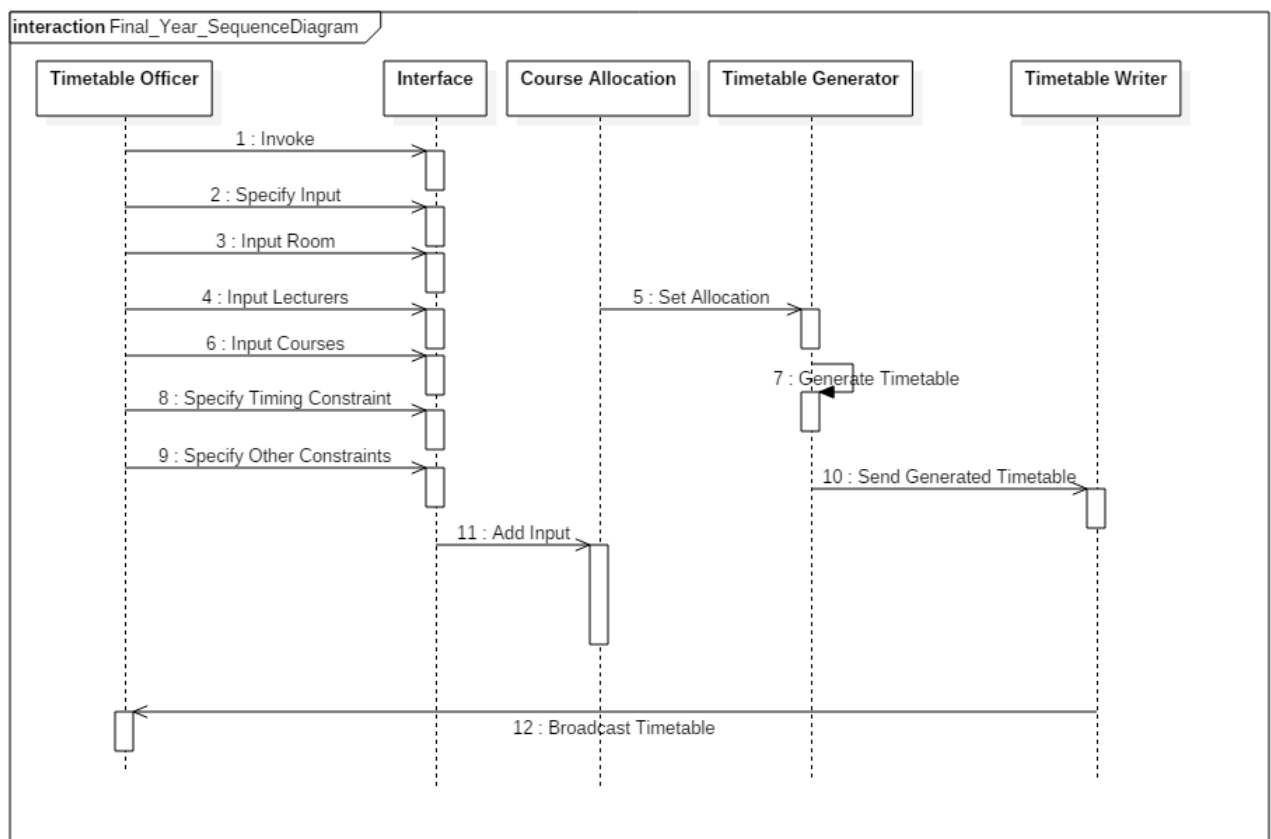


Fig 4.9c: Sequence Diagram

ACTIVITY DIAGRAM

Activity diagrams describe the sequential flow of activities of either a business process or use case. They can also be used to model actions that will be performed when an operation is executed as well as the result of those actions. It shows how activities depend on one another.

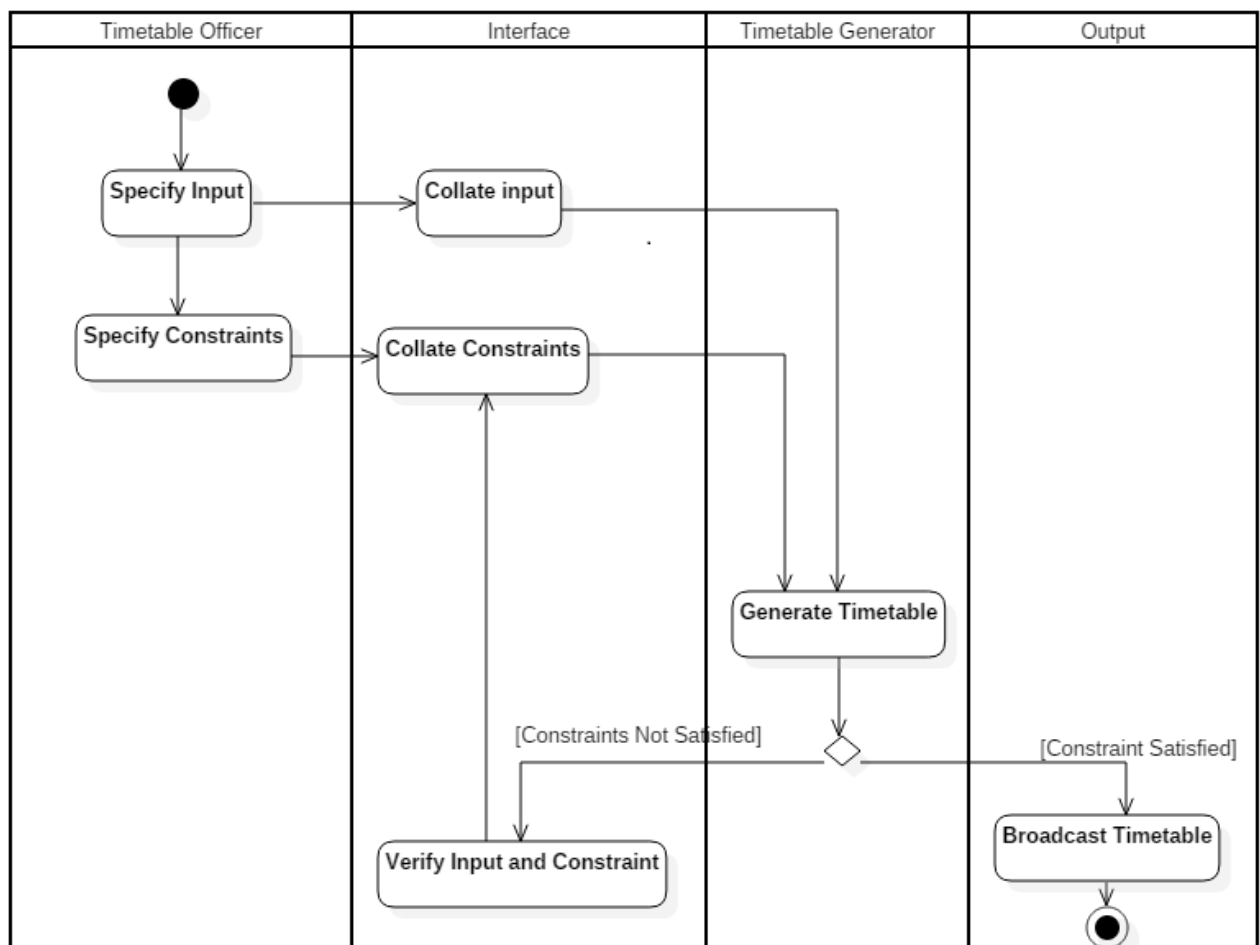
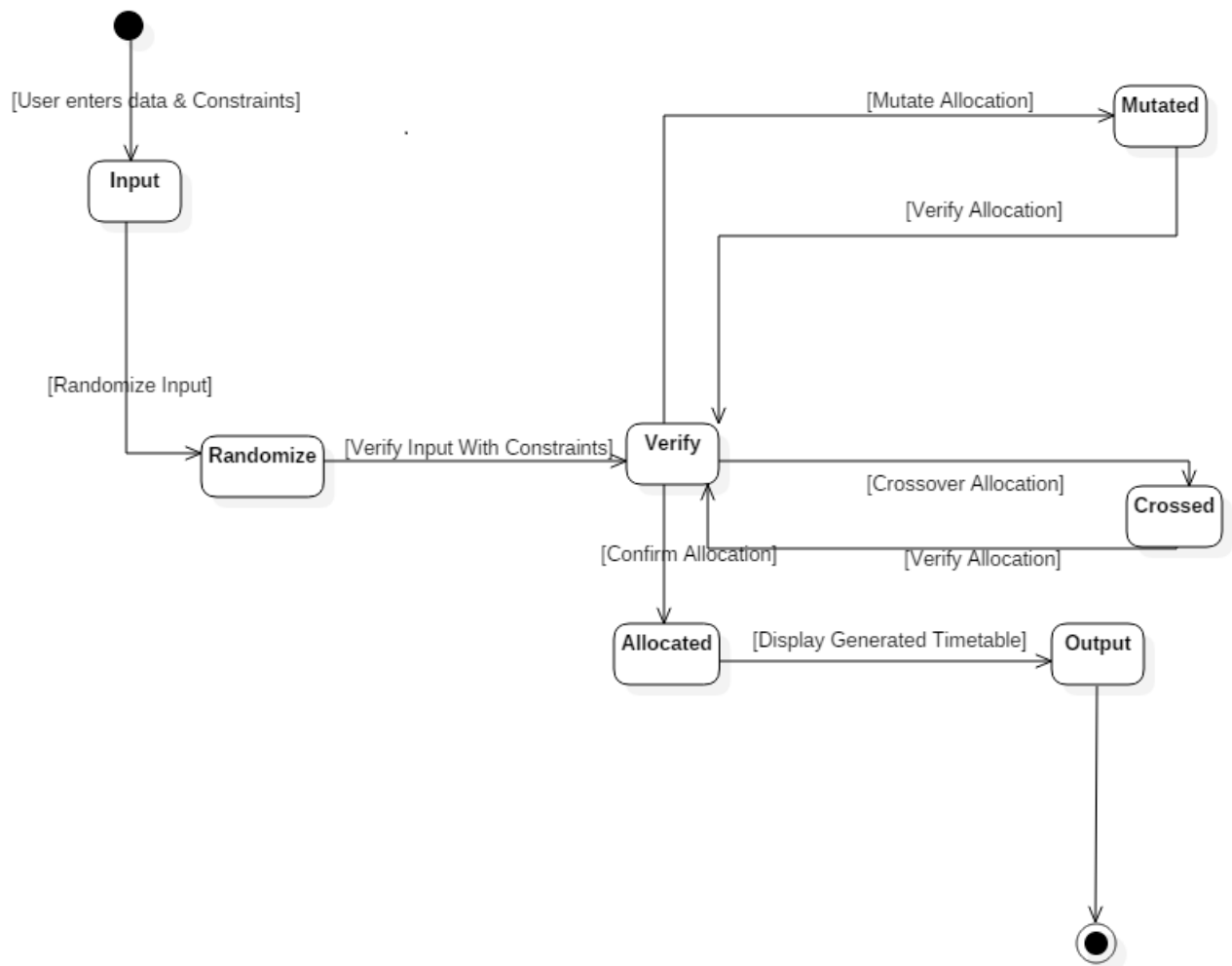


Fig 4.9d: Activity Diagram

STATE DIAGRAM



This is used to model the dynamic behavior of a particular object. They illustrate an object's lifecycle (i.e. the various states that an object can assume) and the events that cause the object to move from one state to another.

4.10 CHOICE OF PROGRAMMING LANGUAGE

The web backend is heavily built using HTML programming language. The HTML is a native language for preparing document for presentation on the internet. In addition to HTML, PHP was used also. PHP is an open source server-side HTML-embedded scripting language. PHP here serves as the server-side

processor to process constraints, generate feasible output and make available to all listening devices.

CHAPTER FIVE

SYSTEM DOCUMENTATION AND IMPLEMENTATION

The system implementation defines the construction, installation, testing and delivery of the proposed system. After thorough analysis and design of the system, the system implementation incorporates all other development phases to produce a functional system.

5.1 SYSTEM REQUIREMENTS

Below are the conditions required on a computer system for the timetable to run;

HARDWARE REQUIREMENTS

- Processor should be Pentium V and above
- A minimum of 1Gigabyte of RAM
- Hard Disk space of 80Gigabyte and above.

For the mobile interface,

- Android device
- A minimum of 512megabyte of RAM.

SOFTWARE REQUIREMENTS

- Linux, Windows, Mac Operating Systems.
- PC Web Browser – Mozilla, Google Chrome, Internet Explorer, Safari

etc.

For the mobile interface,

- A minimum version of Android 4.2.2 (Kitkat)

5.2 HOW TO INSTALL

The software installation process requires a live hosting of the web application on either a public or a private web server. The android application must also be installed on an android device. Provided the web and mobile application has been hosted and installed successfully, then it is ready for use.

5.3 TRAINING OF OPERATORS

This is very important for the new system. For the new system to be successful the steps involved in the training of timetable officers and site administrators are as follows;

- Educating the users on how to use the internet.
- Educate the users on the objectives and benefits of the new system.
- The users will also be educated on roles management as well as their duties in the new system.
- In other to actualize the above and enhance the user training, workshops will be organized for operators.
- Making the system documentation available to users.

5.4 IMPLEMENTATION METHOD

This is concerned with the way the system will be implemented. The activities involved in system implementation and programming, testing the system, training of operators and conversion from the old system to the new system.

a. PROGRAMMING

The conversion of the design specification into computer readable language is performed here. Subsystem modules were developed and compiled to produce the new system. Many frameworks were employed to build the web application backend.

- jQuery (JavaScript framework)
- Bootstrap (CSS framework)
- ionicframework (CSS framework)
- CodeIgniter (PHP framework)

For the mobile frontend, java and xml were used for the development.

b. TESTING THE SYSTEM

The new system was tested with the required constraints and the output was a feasible timetable solution.

c. CHANGE OVER PROCEDURES

This process involves changing from the existing procedure to an online timetable management system. The parallel conversion method was chosen.

This is to allow both the old and new system run simultaneously for some time

to ensure that the new system meets the requirements it is introduced to. Other conversion methods include

- Direct Changeover Method

This is mainly used where there is a strong confidence in the design and implementation of the new system. It involves complete changeover of the old system when the new system has been implemented.

- Pilot Changeover Method

This involves pre-testing the new system in a separate or isolated testing environment using the same form of data. This method provides an opportunity of testing the systems operational reliability and risk of changing everything at once.

- Phased Changeover Method

Here, conversion is carried out gradually until the entire conversion is completed. This is used when the old and new systems look alike.

5.5 REVIEW AND MAINTENANCE OF THE SYSTEM

It is necessary to check the new system periodically to present unforeseen problems that might arise in the cause of using the system and to give room for system upgrade when necessary. File maintenance operations includes:

- Record deletion operation.

- Record insertion operational.
- Record rearrangement and amendment.

Due to numerous advantages offered to this new system through its index-sequential method of file accessing, file maintenance operations are easily done.

FILE SECURITY

The researcher used the following items to ensure adequate security of system.

- Encryption of files using CodeIgniter's encryption key.
- Backup copies of database.
- Effective use of passwords for user authentication.
- Restricting access for non-administrative positions.

CHAPTER SIX

SUMMARY, CONCLUSION AND RECOMMENDATION

6.1 SUMMARY

This study was carried out as is to reduce the intense manual effort being put into creating and developing school lecture timetables. The timetable automation system currently is a conceptual work in progress but has the capability to generate near-optimal timetables based on three unit courses with minimized course constraints.

6.2 PROBLEMS ENCOUNTERED

Timing constraints was a major issue in the development of this system due to the robustness of the system. Given the time allotment for this project, the system developed could not meet up to the intended robustness.

6.3 CONCLUSION

Timetabling problem being the hard combinatorial problem that is would take more than just the application of only one principle. The timetable problem may only be solved when the constraints and allocations are clearly defined and simplified thoroughly and more than one principle is applied to it (i.e. A hybrid solution – a combination of different solution techniques).

6.4CONTRIBUTION TO KNOWLEDGE

Timetable problem is **NP-Complete**. Timetable management has been made easier with this solution. It includes a web app to generate timetable and push to android devices.

6.5RECOMMENDATIONS

Having understudied the challenges that are contained in the manual timetable system, the following are recommended

1. The timetable system developed should be broadened to accommodate the timetable management of the entire department (academic) in higher institution and made open to all students.
2. Further works on developing a timetable system should be based this research work so as to utilize the incremental model of software development.
3. A collaborative model of timetabling system which utilizes mobile operating systems should be built.

REFERENCES

- A. Cornelissen, M.J. Sprengers and B.Mader(2010)"OPUS-College Timetable Module Design Document" Journal of Computer Science 1(1), 1-7.
- Abramson D., & Abela J. (1992). "A parallel genetic algorithm for solving the school timetabling problem." In Proceedings of the 15th Australian Computer Science Conference, Hobart, 1-11.
- Adam, M. (2004). "Genetic Algorithms and Evolutionary Computation ". Available online at <http://www.talkorigins.org/faqs/genalg/genalg.html>.
- Al-Attar, A.(1994). White Paper: "A hybrid GA-heuristic search strategy." AI Expert, USA.
- Alberto, C., Marco D., Vittorio, M. (1992). "A Genetic Algorithm to Solve the Timetable Problem" Journal of Computational Optimization and Applications, 1, 90-92.
- Bufe, M., Fischer, T., Gubbels H., Hacker C., Hasprich O., Scheibel C., Weicker K., Weicker N., Wenig M., & Wolfangel C. (2001). Automated solution of a highly constrained school timetabling problem - preliminary results. EvoWorkshops, Como-Italy.
- Burke E, Elliman D and Weare R (1994)."A genetic algorithm for university timetabling system." Presented at the East-West Conference on Computer Technologies in Education, Crimea, Ukraine .
- Carrasco M.P., & Pato M.V.(2001). "A multiobjective genetic algorithm for the class/teacher timetabling problem." In Proceedings of the Practice and Theory of Automated Timetabling (PATAT'00), Lecture Notes In Computer Science, Springer, 2079, 3-17.
- Chan, H. W. (1997). "School Timetabling Using Genetic Search." 2th International Conference on the Practice and Theory of Automated Timetabling, PATAT'97.
- Coello, C. (2000). "An updated survey of GA-based multiobjective optimization techniques." ACM Computing Surveys, 32(2), 109-143.
- Costa, D. (1994). "A tabu search algorithm for computing an operational timetable." European Journal of Operational Research, 76(1), 98-110.

- Datta, D., Deb K., & Fonseca, C.M.(2006). Multi-objective evolutionary algorithm for university class timetabling problem, In *Evolutionary Scheduling*, Springer-Verlag Press.
- David, A. C.(1999). *An Introduction to Genetic Algorithms for Scientists and Engineers*, 1st ed. World Scientific Publishing Co. Pte. Ltd.
- Dawkins, R (1996). *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design*. W.W. Norton.
- de Gans O.B.(1981). "A computer timetabling system for secondary schools in the Netherlands". *European Journal of Operations Research*,7, 175-182.
- Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons Ltd, England.
- Deb, K., Agarwal S., Pratap A., & Meyarivan T.(2002). "A fast and elitist multi-objective genetic algorithm: NSGA-II." *IEEE Transactions on Evolutionary Computation*, 6(2), 182-197.
- Eley, M. (2006). "Ant Algorithms for the Exam Timetabling Problem." 6th International Conference on the Practice and Theory of Automated Timetabling, PATAT'06.
- Fang, H. L. (1994). "Genetic Algorithms in Timetabling Problems." PhD Thesis, University of Edinburgh.
- Fernandes, C. (2002). "Infected Genes Evolutionary Algorithm for School Timetabling." WSES International Conference.
- Fleming, P. and R.C. Purshouse(2002). "Evolutionary algorithms in control systems engineering: a survey." *Control Engineering Practice*, 10,1223-1241.
- Forrest, S. (1993). "Genetic algorithms: principles of natural selection applied to computation." *Journal of Science*, 261, 872-878.
- Gianoglio, P. (1990). "Application of neural networks to timetable construction." Presented at the 3rd International Workshop on Neural Network and Their Applications, Nimes, France.
- Goldberg, D.(1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

- Gotlieb, C .C. (1962). "The construction of class-teacher timetables." Proceedings of IFIP Congress, North-Holland Pub. Co., Amsterdam, 73-77.
- Gröbner, M., Wilke, P (2002). "A General View on Timetabling Problems." 4th International Conference on the Practice and Theory of Automated Timetabling - PATAT'02.
- Haupt, R. and Sue, E. H. (1998). Practical Genetic Algorithms. John Wiley & Sons.
- Holland, J. (1975). Scheduling, Adaptation in Natural and Artificial Systems. The University of Michigan Press.
- John, H. (1992). "Genetic algorithms." Scientific American, 66-72.
- Jose, J. M (2008) "A system for Automatic Construcion of Exam Timetable Using Genetic Algorithms" Journal of Algorithm and Computations 6(9), 1-18.
- Kostuch, P.A (2003), University of Oxford - University Course Timetabling. St. Anne's College.
- Koza, J., Forest, B., David, A. and Martin, K. (1999). Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers.
- Koza, J., Martin, K., Matthew, S., William, M., Jessen, Y and Guido, L.(2003). Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers.
- Lajos, G. (1995). "Complete university modular timetabling using constraint logic programming." Presented at the Ist International Conference on the Practice and Theory of Automated Timetabling, Scotland, UK.
- Lawrie, N.L. (1969). "An integer programming model of a school timetabling problem." The Computer Journal, 12, 307-316.
- Looi, C. (1992). "Neural network methods in combinatorial optimization" Journal of Computers and Operations Research, 19(3/4), 191-208.
- Luisa, C, Ana Ceira-Pena, Guillermo de Bernardo, Diego Seco" An Integrated System for School Timetabling" Journal of Heuristics 1(1), 1.
- Mitchell, M. (1996). An Introduction to Genetic Algorithms. MIT Press.

- Paechter B, Cumming A, Luchian H and Petriuc M (1994). "Two solutions to the general timetable problem using evolutionary methods." Presented at the 3rd World Conference on Evolutionary Computing. Florida, USA.
- PATAT (1995). The International Series of Conferences on the Practice and Theory of Automated Timetabling (PATAT) - <http://www.asap.cs.nott.ac.uk/patat/patat-index.shtml>
- Piola, R. (1994). "Evolutionary solutions to a highly constrained combinatorial problem." In Proceedings of IEEE Conference on Evolutionary Computation (FirstWorld Congress on Computational Intelligence), Orlando, Florida, 1, 446-450
- Robertus J. W. (2002). School timetable construction-Algorithm and Complexity, 1st ed. Netherlands: University of Eindhoven Press
- S. B. Deris, S. Omatu, H. Ohta, P. A. B. D. Samat(1997). "University timetabling by constraint-based reasoning: A case study" Journal of the Operational Research Society, 48(12), 2-4.
- Srinivasan D., Seow T.H., & Xu J.X.,(2002). "Automated time table generation using multiple context reasoning for university modules." In Proceedings of IEEE International Conference on Evolutionary Computation (CEC'02), 1751-1756.
- Tripathy A. (1984). "School timetabling - A case in large binary integer linear programming." Management Science, 30(12), 1473-1489.
- Wright M. (1996). "School timetabling using heuristic search." Journal of Operational Research Society, 47, 347-357.
- Ossam, C. (2009). "University Scheduling using Genetic Algorithm." Master Thesis - Department of Computer Engineering, Dalarna Univeristy, 3-4.

APPENDIX I

PROGRAM SOURCE CODE

Index page

```
<?php defined('BASEPATH') OR exit('No direct script access allowed'); ?>

<div class="login-box">
    <div class="login-logo">
        <a href="<?= base_url();?>"><b>STS</b> CPanel</a>
    </div>
    <?php if($this->session->flashdata('msg') != ""){?>
        <div class="callout callout-danger">
            <h4>Login Failure</h4>
            <?php echo $this->session->flashdata('msg');?>
        </div><?php }?>
    <div class="login-box-body">
        <fieldset>
            <form action="<?= base_url();?>" method="post" autocomplete="off">
                <div class="form-group has-feedback">
                    <input type="text" name="email" class="form-control" placeholder="Login ID">
                    <span class="glyphicon glyphicon-lock form-control-feedback"></span>
                </div>
                <div class="has-feedback">
                    <input type="password" name="password" class="form-control"
placeholder="Login PIN">
                    <span class="glyphicon glyphicon-pushpin form-control-feedback"></span>
                </div>
                <button type="submit" class="btn btn-success btn-block btn-flat">Sign
In</button>
            </form>
        </fieldset>
    </div>
</div>
```

DASHBOARD

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

$this->load->view('themes/navbar');
$this->load->view('themes/sidebar');

?>    <section class="content-header">
        <h1>        Dashboard
    </h1>        <ol class="breadcrumb">
            <li><a href="#"><i class="fa fa-dashboard"></i> Home</a></li>
            <li class="active">Dashboard</li>        </ol>
    </section>
```

```

<section class="content">
    <div class="row">
        <div class="col-md-12">
            <div class="col-md-6">
                </div>
            <div class="box box-info">
                <div class="box-header with-border">
                    <h3 class="box-title">Site
Back Up</h3>
                <div class="box-tools pull-right">
                    <button class="btn btn-box-tool" data-widget="collapse"><i class="fa fa-
minus"></i></button>
                    <button class="btn btn-box-tool" data-widget="remove"><i class="fa fa-
times"></i></button>
                </div>
            </div>
        </div>
    </div>
    <div class="box">
        <div class="box-header">
        </div>
        <form method="POST" action="<?= base_url('backend/home/error');?>">
        <div class="box-body">
            <table id="datatables" class="table table-bordered table-striped">
                <thead>
                    <tr>
                        <th width="1%" align="left"><input type="checkbox" name="chkall"
id="chkall" onclick="return checkall('selector[]');"> </th>
                        <th>Error Type</th>
                        <th>Error Title</th>
                        <th>Location</th>
                        <th>Error Line</th>
                        <th>Time</th>
                    </tr>
                </thead>
                <tbody>
                    <?php foreach($error_log as $key => $value):
                        echo '<tr>
                            <td width="1%"><input type="checkbox" name="selector[]"
id="selector[]" value="">'.$value->id. "'>
                            <td>'.$value->errtype.'</td>

```

```
<td>'{$value->errstr}'</td>

<td>'{$value->errfile}'</td>

<td>'{$value->errline}'</td>

<td>'{$value->time}'</td>;

endforeach; ?>

</tr>

</tbody>

</table>

</div>

<div class="box-footer clearfix">

<div class="btn-group">

    <button type="submit" class="btn btn-danger" onclick="return
confirm('Deleted item(s) cannot be undone. Are you sure?');" name="delete"><span
class="glyphicon glyphicon-trash"></span> Delete Selected</button>

</div>

<div class="pull-right"></div>

</div>

</form>

</div>

</div>

</div>

</div>

</section>

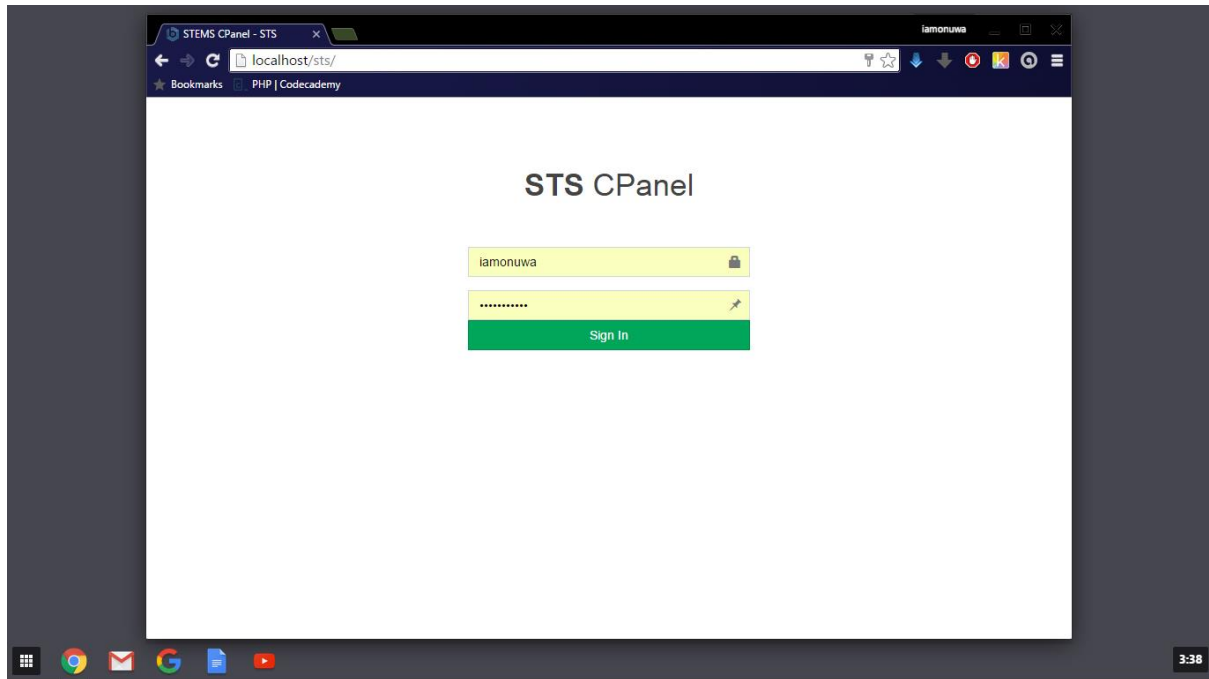
</div>

<?php $this->load->view('themes/footer');?>
```

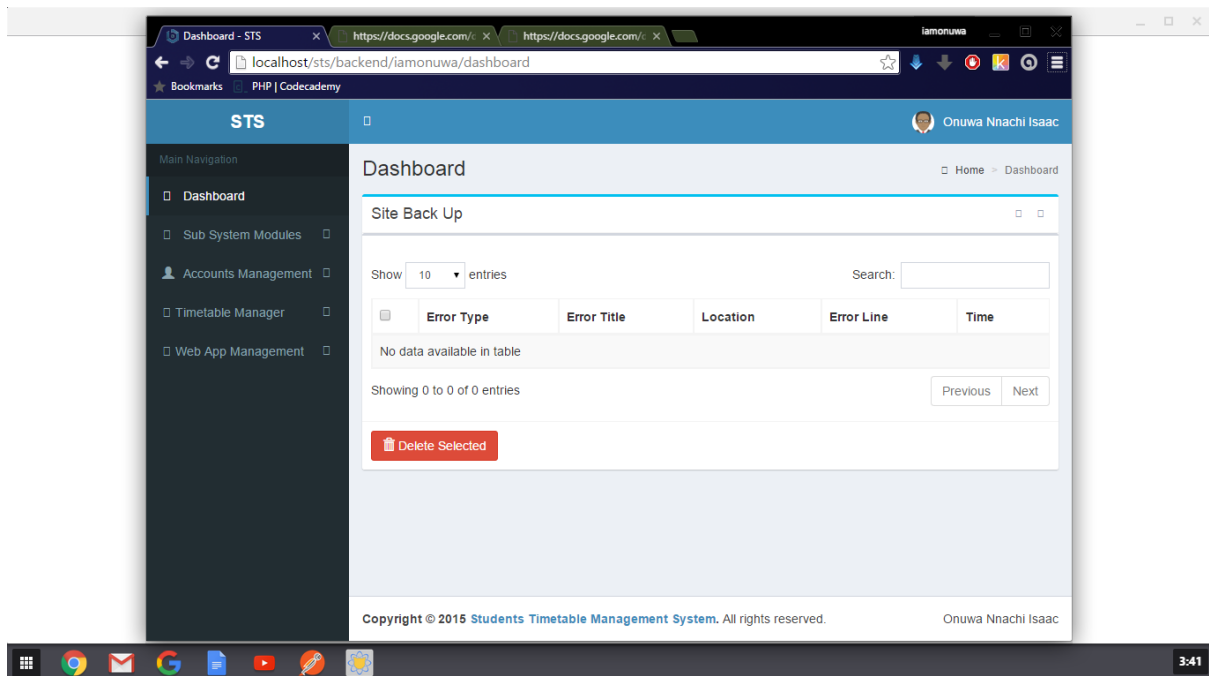
APPENDIX II

PROGRAM OUTPUT

INDEX PAGE



DASHBOARD



APPENDIX III

LIST OF FIGURES

Figure 2.7a:	Diagram showing the effect of mutation on an individual...	26
Figure 2.7b:	Diagram showing the effect of mutation on individuals	27
Figure 2.10:	Diagram depicting the Hybrid Genetic Algorithm	44
Figure 3.1	Organogram for Department of Computer Science	46
Figure 4.2:	System Block Diagram	53
Figure 4.5a:	Course Input Module	55
Figure 4.5b:	Building Input Module	56
Figure 4.5b:	Room Input Module	56
Figure 4.5b:	Lecturer Module	57
Figure 4.8:	Program Flowchart	58
Figure 4.9a:	Use Case Diagram	61
Figure 4.9b:	Class Diagram	62
Figure 4.9c:	Sequence Diagram	63
Figure 4.9d:	Activity Diagram	64
Figure 4.9e:	State Diagram	65