AUTOMATIC TIME TABLE GENERATOR AND COURSE MANAGEMENT SYSTEM FOR JUIT

Project report submitted in partial fulfillment of the requirement for The degree of Bachelor of Technology in

Computer Science and Engineering

By
Aditya Tyagi (121228)
Sanjay Chaudhary (121220)

Under the supervision of

Ms. AnnieSingla

To

Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Certificate

Candidate's Declaration

I hereby declare that the work presented in this report entitled **"**AUTOMATIC TIME TABLE GENERATOR AND COURSE MANAGEMENT SYSTEM FOR JUIT**"**in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering/Information Technologysubmitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat an authentic record of my own work carried out over a period from August 2015 to December 2015 under the supervision of Miss Annie SinglaAssistant Professor (Grade-I) **.**

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature)
Student Name: Aditya Tyagi
Rollno:121228

(Student Signature)
Student Name: Sanjay Choudhary
Rollno:121220

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Supervisor Name: Miss Annie Singla
Designation: Assistant Professor (Grade-I)**.**
Department name: ICT
(Supervisor Signature)
Dated: MAY2015

ACKNOWLEDGEMENT

This work is a synergistic product of many minds and I feel a deep sense of gratitude tomy parents, for their encouragement and for being ever supportive. My sincere thanks goes to my supervisor Ms. Annie Singla for her thorough assistance with this work. I also acknowledge the Computer Science Students for their active verbal participation and suggestions towards the evolvement of this project work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Scheduling course timetables for a large array of courses is a very complex problem which often has to be solved manually by the center staff even though results are not always fully optimal. Timetabling being a highly constrained combinatorial problem, this work attempts to put into play the effectiveness of evolutionary techniques based on Darwin's theories to solve the timetabling problem if not fully optimal but near optimal.

Genetic Algorithm is a popular meta-heuristic that has been successfully applied to many hard combinatorial optimization problems which includes timetabling and scheduling problems. In this work, the course sets, halls and time allocations are represented by a multidimensional array on which a local search is performed and a combination of the direct representation of the timetable with heuristic crossover is made to ensure that fundamental constraints are not violated.

Finally, the genetic algorithm was applied in the development of a viable timetabling system which was tested to demonstrate the variety of possible timetables that can be generated based on user specified constraint and requirements.

CHAPTER ONE

INTRODUCTION

1.1. BACKGROUND STATEMENT

Timetabling concerns all activities with regard to making a timetable that must be subjective to different constraints. According to Collins Concise Dictionary (4th Edition) "a *timetable* is a *table of events arranged according to the time when they take place*."

A critical factor in running a university or essentially an academic environment is the need for a well-planned and clash-free timetable. Back in the times when technology was not in wide use, academic timetables were manually created by the educational center staff.

Every year, JUIT face the rigorous task of drawing up timetables that satisfies the various courses and their respective examinations being offered by the different department. The difficulty is due to the great complexity of the construction of timetables for lectures and exams, due to the scheduling size of the lectures and examinations periods and the high number of constraints and criteria of allocation, usually circumvented with the use of little strict heuristics, based on solutions from previous years.

A timetable management system is designed and created to handle as much course data as fed while ensuring the avoidance of redundancy.An educational timetable must meet a number of requirements and should satisfy the desires of all entities involved simultaneously as well as possible. The timing of events must be such that nobody has more than one event at the same time.

The proposed timetabling system is designed to handle events of course lectures offered at JUIT.

Based on the above event, the system would have only one module which is the Course Lecture Timetable Module.

1.2. PROBLEMSTATEMENT

The systems available currently can build or generate a set of timetables, but still have issues with generating a clash-free and complete timetable. The tedious tasks of data introduction and revision of usually incomplete solutions are the bottleneck in these cases. JUIT (most educational institutions) have resorted to manual generation of their timetables which according to statistics takes over a month 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. OBJECTIVE

The literature on and implementation of educational timetabling problems is quite scattered. Different research papers that have been brought out on timetabling may refer to the same type of institution but they mostly deal with different kinds of assignments, i.e., decisions like the timing of events, sectioning students into groups, or assigning events to locations. Moreover, each institution has its own characteristics which are reflected in the problem definition. Yet, there have been no leveling grounds for developing a system that can work for most of these institutions.

The aim of this work is the generation of course schedules while demonstrating the possibility of building these schedules automatically through the use of computers in such a way that they are optimal and complete with little or no redundancythrough the development of a viable lecture timetabling software.

The primary objective is 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.
The secondary objective is to expand the scope of timetable automation systems by making it generic thereby bringing about uniformity in the creation of timetables as it applies to different universities or educational institutions i.e. will be able to generate timetables that fit the requirements of any academic institution.

1.4. RESEARCH METHODOLOGY

This research is concerned with the problem of constructing timetables for JUIT. JUIT timetable construction problems are interesting objects to study because neither modeling nor solving them is straightforward. It is difficult to make a clear-cut distinction between acceptable and not acceptable timetables. Because of the large diversity in acceptance criteria, realistic timetable construction problems are multidimensional. Each dimension may introduce its own characteristic aspects that add to the complexity of the problem. Therefore, only heuristic solution approaches without known performance guarantees are practically feasible.

As a result of the large data input a timetable management system is supposed to handle, a linear method or algorithm cannot be employed to handle such validation and generation, hence the usage of a heuristic method. The heuristic method to be used in this study is the genetic algorithm. The genetic algorithm is one that seeks to find the most optimal solutions where the search space is great and conventional methods is inefficient.; it works on a basis of the Darwinian evolution theory.

1.5. SCOPE OF THE STUDY

The boundaries of this work take into consideration all academic institutions from the primary level to the higher institution level. It will only involve the technical skills of one academic personnel and a few data collaborators to handle the data input and constraint specifications. The proposed system will be able to handle as much data input as possible i.e. the course data, halls data, lecturer's data etc.

1.6. SIGNIFICANCE OF THE STUDY

Outlined below is the significance of the proposed system and the improvements on existing systems that are featured in this system:

- The proposed system will provide an attractive graphical front-end which acts as the main point of interaction with user and data collaborators.
- It will also improve flexibility in timetable construction.
- The system will be able to generate reports on conflicting constraint specifications.

- The system will seek to improve on previous versions of timetable generating system.
- Stress in creating timetable manually will be greatly reduced as output would be automated.
- The system will save time.
- Productivity will be improved.
- The system can be revised i.e. its backend can be revised.

1.7. LIMITATIONS

Below are some of the limitations that may hinder the functionalities of the system:

- Unserious collaborative work from staffs in the various departments.
- Incomplete data from data collaborators.
- Wrong data input by technical user: the system will only work with data supplied; hence wrong data input might have to be edited manually.
- Wrong constraints specification.
- Low system memory capacity.

1.8. RESEARCH OUTLINE

This report contains a further four sections. Chapter 2 gives further background information while reviewing in details the workings of existing systems. Chapter 3 discusses the structure, design and internal workings of each module in the project, it also details the tasks required to complete the project, and a timescale to complete them in. Chapter 4 details the backend of the system and shows the development and testing of each stage in the project. Chapter 5 presents my summary, conclusions and recommendations. The final section lists the references used while writing this report

CHAPTER TWO

REVIEW OF RELEVANT LITERATURE

2.1. INTRODUCTION

A Timetable or schedule 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's activities. As a result, schools have devoted time, energy and human capital to the implementation of nearly optimal timetables which must be able to satisfy all requirements constraints as specified by participating entities.

The class lecture timetabling problem is a typical scheduling problem that appears to be a tedious job in JUIT 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.

The automated timetabling and scheduling is one of the hardest problem areas already proven to be NP-Complete and it is worthy of note is that as educational institutions are challenged to grow in number and complexity, their resources and events are becoming harder to schedule, hence the choice of this project topic which entails investigating the performance of Genetic Algorithm on the optimality of timetabling problems under predefined constraints.

2.2. REVIEW OF RELEVANT EXISTING THEORIES 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.

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. Since the problem in schools is relatively simple because of their simple class structures, classical methods, such as linear or integer programming approaches 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, neural network and tabular search algorithm. 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.

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 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).

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 JUIT. 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, JUIT has adopted a semester system which gives 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, heuristics, graph coloring, integer programming, genetic algorithms, knowledge-based, and constraint logic programming. 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.

2.3. TIMETABLING AS AN NP-COMPLETE PROBLEM

In computational complexity theory, the complexity class NP-complete (abbreviated NP-C or NPC, NPstanding 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.

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.

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.

2.4. THOROUGH EXAMINATION OF THE GENETIC ALGORITHM

2.4.1. 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". 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. 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, 1996; Goldberg, 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. 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).

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, 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" 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.

CHAPTER THREE

SYSTEM ANALYSIS AND DESIGN

3.1. INTRODUCTION

System Analysis is the study of a business problem domain to recommend improvements and specify the business requirements and priorities for the solution. It involves the analyzing and understanding a problem, then identifying alternative solutions, choosing the best course of action and then designing the chosen solution.

It involves determining how existing systems work and the problems associated with existing systems. It is worthy to note that before a new system can be designed, it is necessary to study the system that is to be improved upon or replaced, if there is any.

3.2. THE EXISTING SYSTEM

3.2.1. REVIEW OF THEEXISTING SYSTEM

Timetabling is the whole process concerned with making a timetable having events arranged according to a time when they take place which must be subject to the timing constraints of each entity placed in the table. University timetabling in this context refers to the rigorous task educational center staff in a Covenant University undergo to draw up timetables that satisfies various courses that should compulsorily be inherent in the final timetable solution.

These courses are usually taught by varied lecturers in different departments who may also wish to specify some timing constraints on their courses. Given all the courses and course details, the academic staff is charged with the responsibility of creating a near optimal timetable which would serve as a guide for academic activities in the university.

The traditional manual timetabling system is time-consuming, resource-intensive, involves many steps and requires re-processing the same data several times.

3.2.2. ADVANTAGES OF THE EXISTING SYSTEM

The timetable generation process by the education center staff is:

- Subjective and can be made better through collaboration with the different entities involved.

3.2.3. LIMITATIONS OF THE EXISTING SYSTEM

The traditional manual generations of timetables encounter 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.

3.3. THE PROPOSED SYSTEM

3.3.1. REVIEW OF THEPROPOSED SYSTEM

The proposed systems were 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).

3.3.2. ADVANTAGES OF THEPROPOSED SYSTEM

The timetable generation process by the academic 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.

3.3.3. LIMITATIONS OF THE PROPOSED SYSTEM

The following are the challenges in the system due to time constraints:

- The proposed system can only generate timetables based on a few hard course constraints.
- The proposed system can only generate timetables for two streams.
- Timetable generated by this system is still subject to revision by academic center staff.
- Not all of the genetic algorithm principles are implemented in the system.

3.4. SYSTEMS DESIGN

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

3.5. MODELLING 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 applications before coding. Unified Modeling Language (UML) tools were used in modeling this system.

3.5.1. UML (UNIFIED MODELLING LANGUAGE) MODELLING

The Unified modeling language is an 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 Unified Modeling Language (UML) has become an object modeling standard and adds a variety of techniques to the field of systems 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, Collaboration diagram, Component diagram and State diagram will be used for system modeling.

3.5.1.1. Use Case Diagram

Use case diagrams describe what a system does from the standpoint of an external observer. 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.

ACTORS

- User
- Timetable System

USE CASES

- Specify Input
- Specify Constraints
- Input Courses
- Input Course Lecturers
- Input Departments
- Input Programs
- Input Halls
- Apply and Verify Constraints
- Crossover Course Allocation
- Mutate Course Allocations
- Random Generation and Course Allocations
- Write To External File

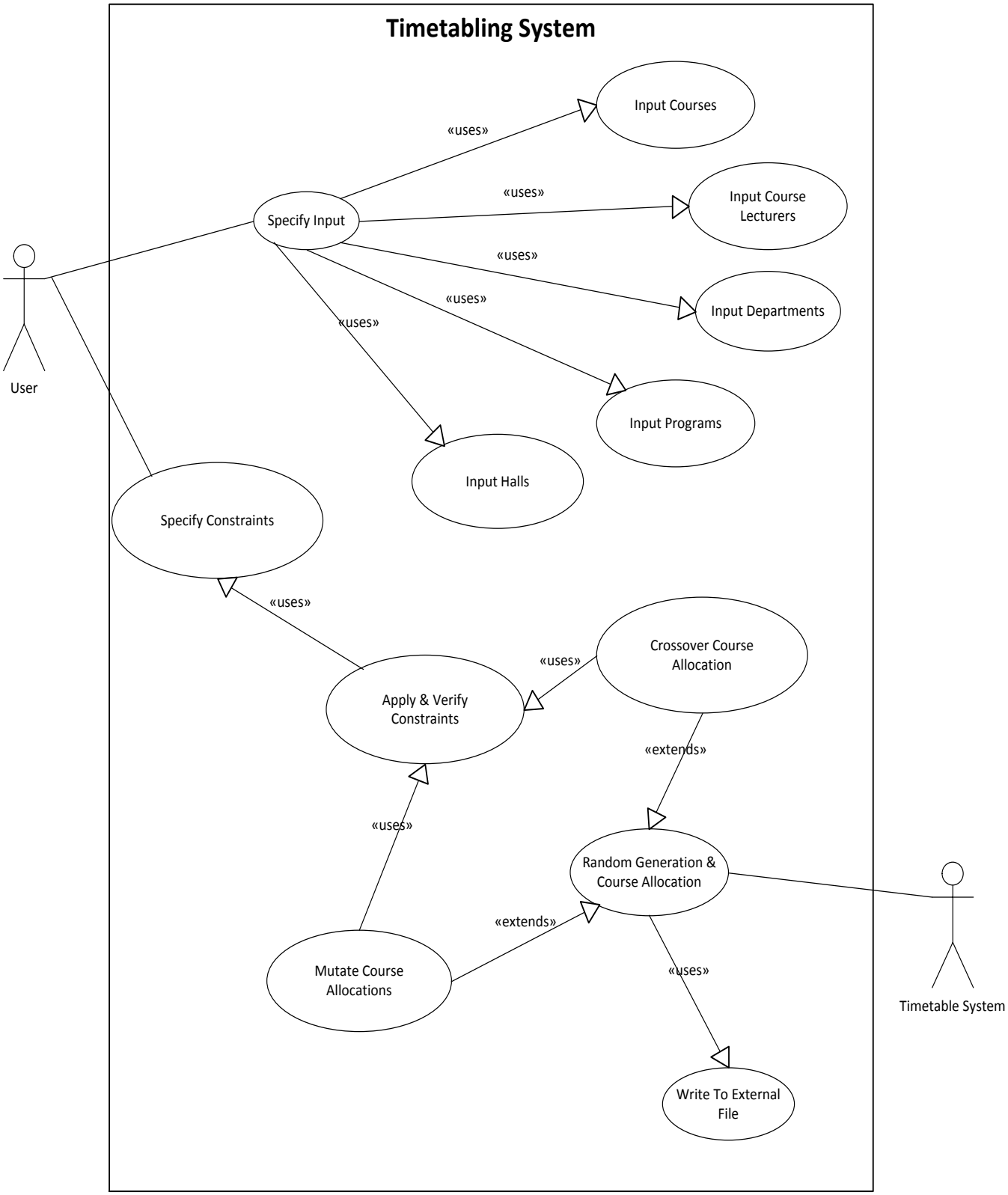


Figure 3.1.: Use Case Diagram to show the interaction between the timetabling system and user

3.5.1.2. Class Diagram

A class diagram is an organization of related objects. It gives an overview of a system by showing its classes and the relationships among them. Class diagrams only display what interacts but not what happens during the interaction hence they are static diagrams.

CLASSES

- Lecturers
- Buildings
- Halls
- Programs
- Courses
- Levels
- Allocations
- Front End Staff
- Generator Module
- File Writer

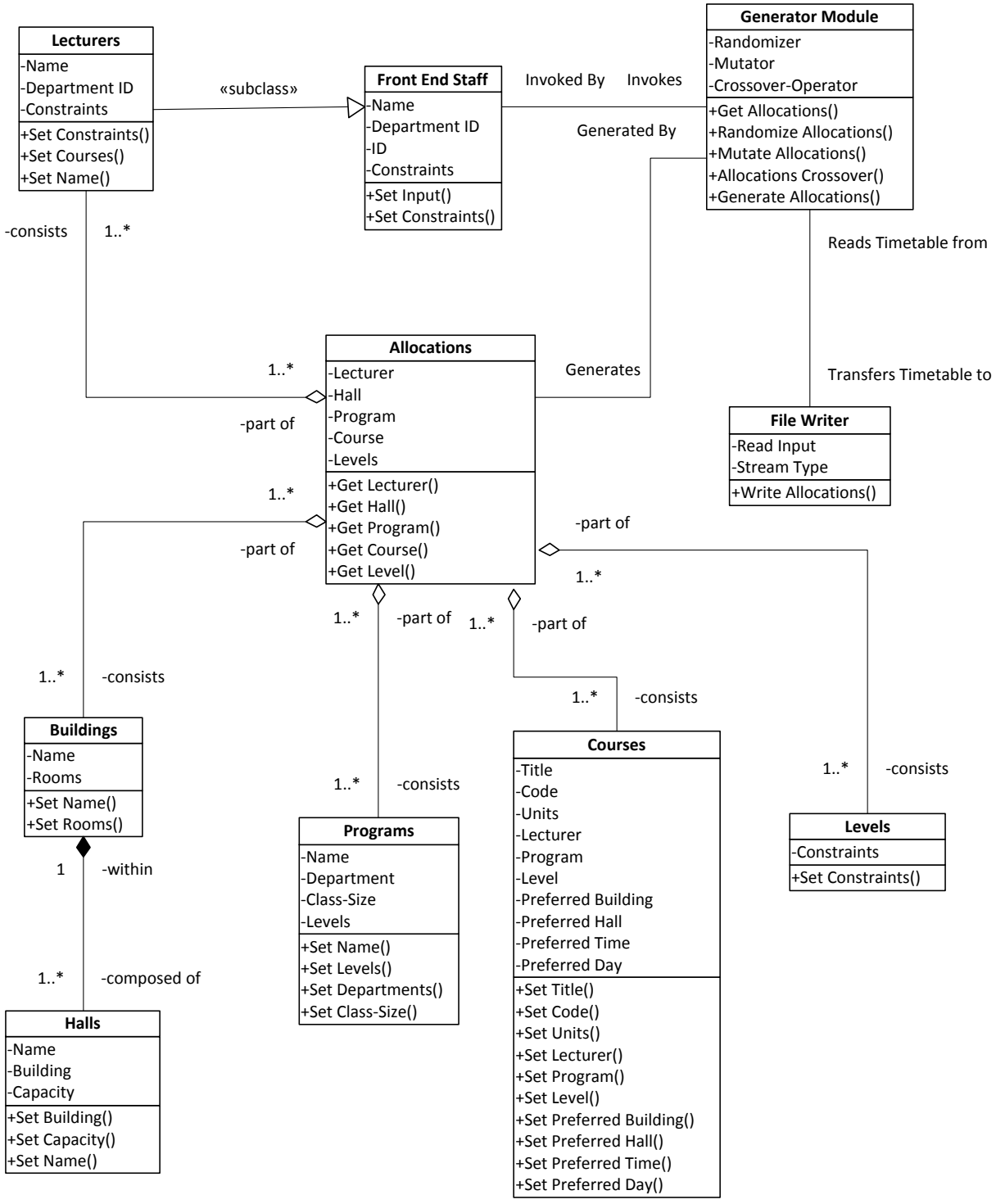


Figure 3.2.: Class Diagram to show the relationships between the different classes associated with the system

3.5.1.3. Sequence Diagram

A sequence graphically depicts how objects interact with each other via messages in the execution of a use case or operation. They illustrate how messages are sent and received between objects and the sequence of message transfer. It also details how operations are carried out according to the time of operation.

CLASSES

- Front End Staff
- Interface
- Course Allocation
- Generator Module
- Timetable Writer

MESSAGES

- Invoke
- Specify Input
- Input Buildings
- Input Halls
- Input Departments
- Input Programs
- Input Lecturers
- Input Courses
- Specify Timing Constraints
- Specify Other Constraints
- Add Input
- Set. allocations
- Generate Timetable
- Send Generated Timetable
- Print Out Generated Timetable

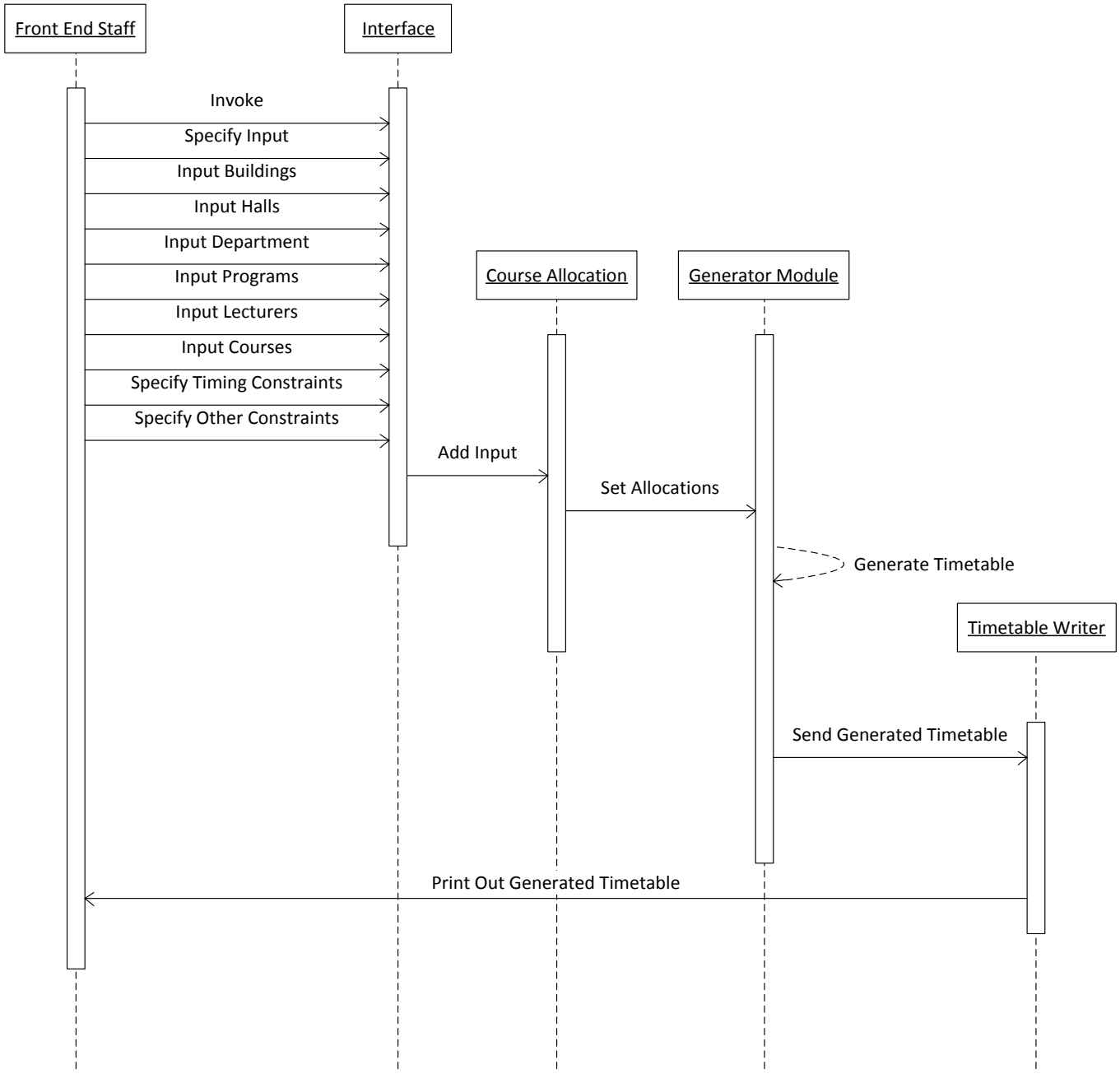


Figure 3.3.: Sequence Diagram to show how the different objects interact during the execution of the system

3.5.1.4. Activity Diagram

Activity diagrams graphically depict the sequential flow of activities of either a business process or a use case. They can also be used to model actions that will be performed when an operation is executed as well as the results of those actions. They focus on the flow of activities involved in a single process. The activity diagram shows the how those activities depend on one another.

SWIMLANES

- Front End User
- Interface
- Timetable Generator Module
- Writer

STATES

- Specify Input
- Specify Constraints
- Collate Input
- Collate Constraints
- Generate Timetable
- Verify Input and Constraints
- Output Timetable to File

GUARD EXPRESSIONS

- Constraints not Satisfied
- Constraints Satisfied

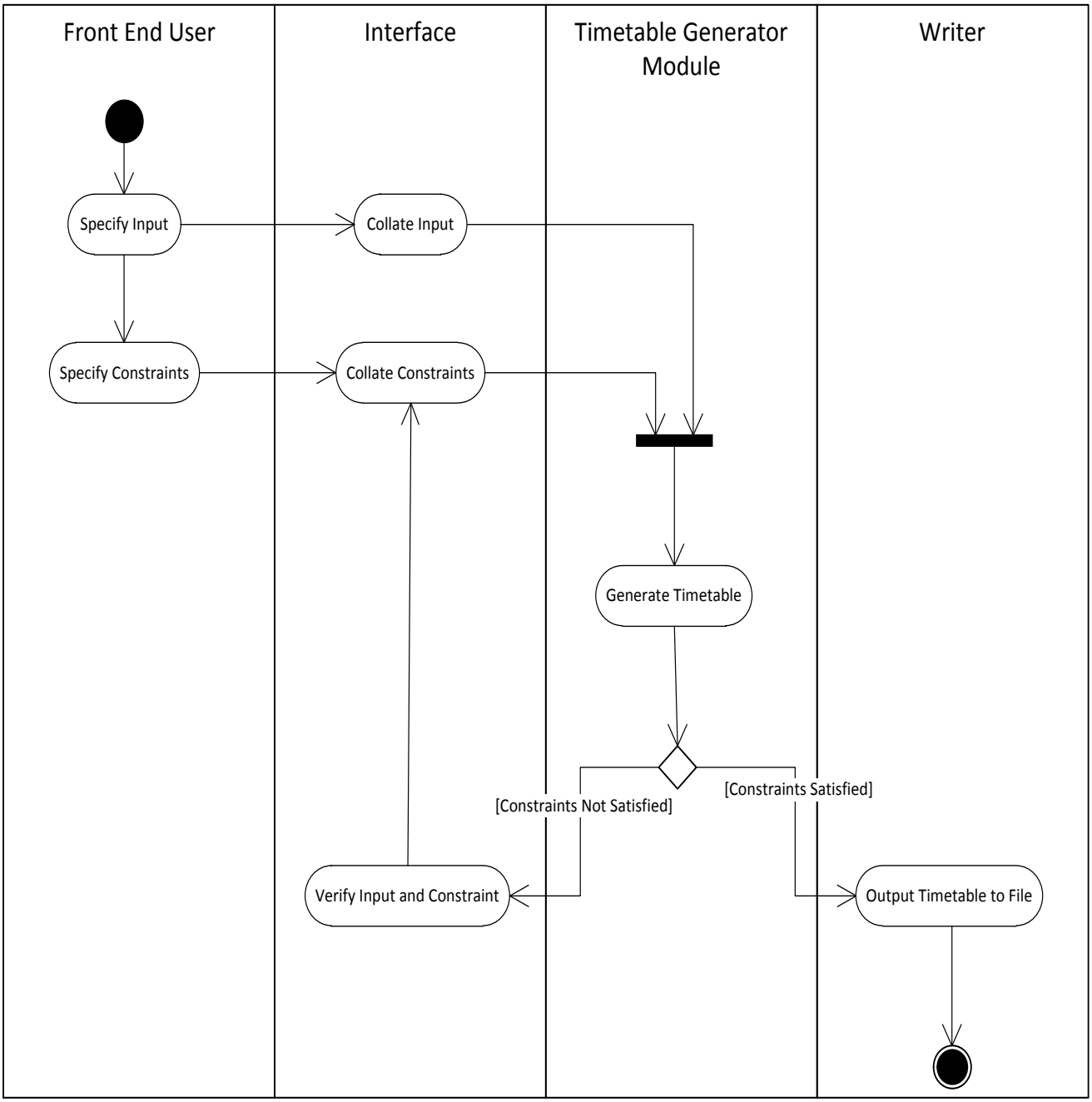


Figure 3.4.: Activity Diagram to model the actions and the output of those actions when an operation is carried out in the system

3.5.1.5. State Diagram

State diagrams are used to model the dynamic behavior of a particular object. They illustrate an object's life cycle i.e. the various states that an object can assume and the events that cause the object to move from one state to another.

STATES

- Input
- Randomize
- Crossed
- Mutated
- Verify
- Group
- Output

TRANSITIONS

- User Enters Inputs & Constraints
- Randomize Input
- Verify Input with Constraints
- Crossover Allocations
- Verify Allocations
- Mutate Allocations
- Confirm Allocations
- Write Allocations to File

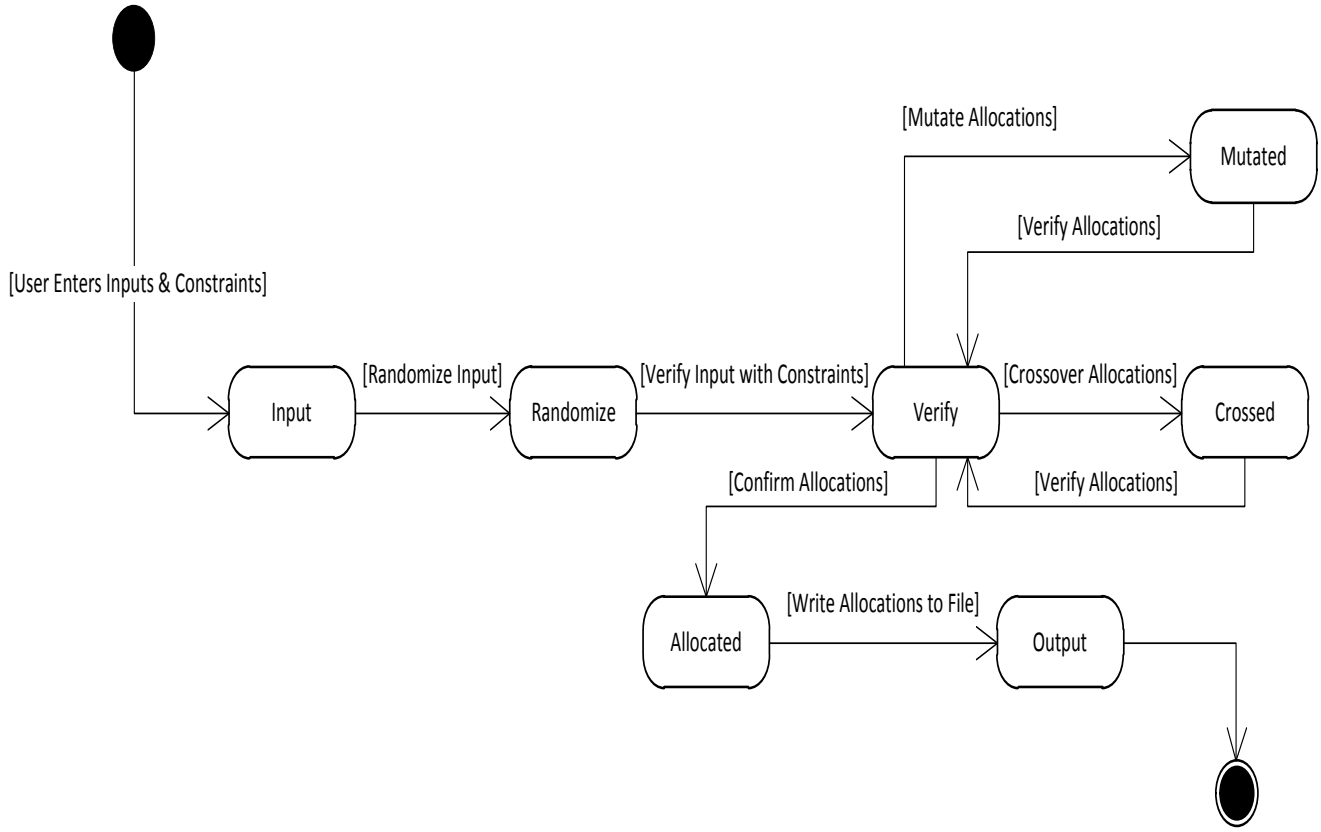


Figure 3.5.: State Diagram to depict the different states of the system during its execution

3.5.1.6. Collaboration Diagram

Collaboration diagrams are similar to sequence diagram but do not focus on the timing or sequence of messages. Instead they present the interaction between objects in a network format.

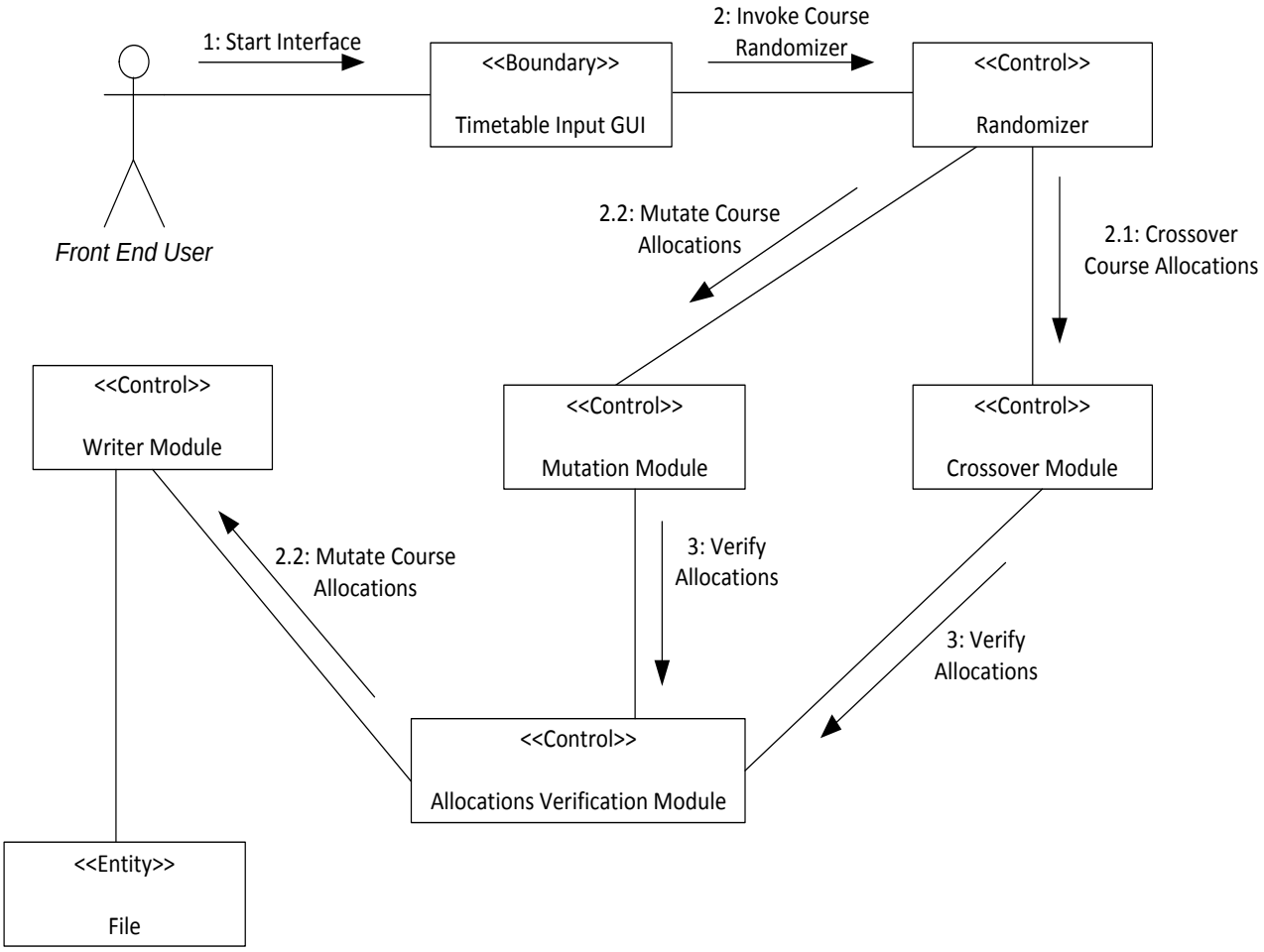


Figure 3.6.: Collaboration Diagram showing the interaction between the objects in the system

3.5.1.7. Component Diagram

Component diagrams describe the organization of physical software components, including source code, run-time (binary) code, and executable.

COMPONENTS

- Input Module
- Randomizer
- Crossover Operator
- Mutation Operator
- Verifier
- File Handler

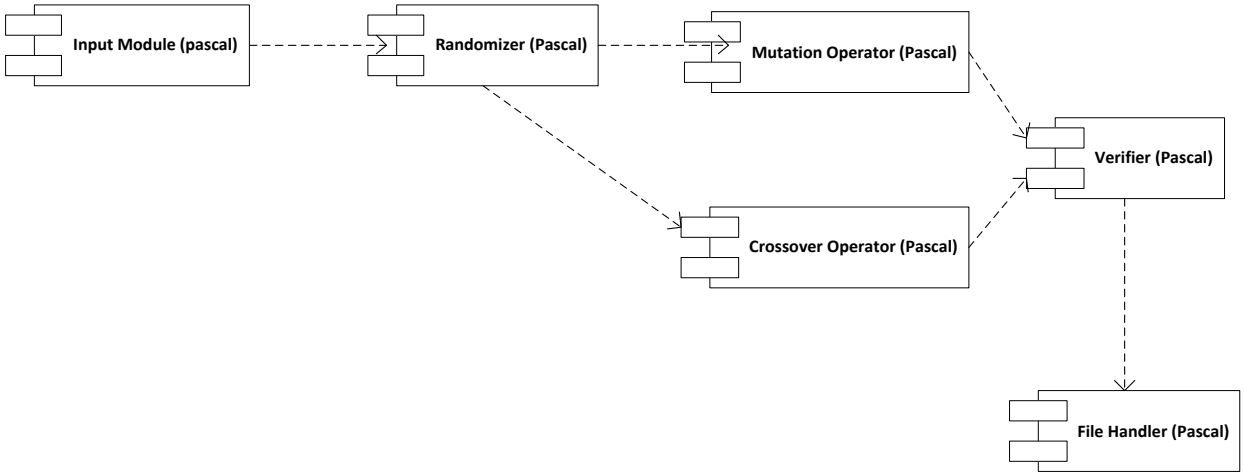


Figure 3.7.: Component Diagram to show the connection between the various software modules in the system

3.6. FILES DESIGN

A file is a collection of similar or related records as a result of this programs complexity, a file processing system is used rather than a database. The timetabling system uses 6 files consisting of:

- **Halls File**: contains the name of the Halls/Rooms that are used in the course allocations within the program. See Table 3.1.

  Table 3.1.: Hall Table

  This table stores the information about each hall used in the timetable allocations.

Field	Type
Hall Name	String
Hall Capacity	Int

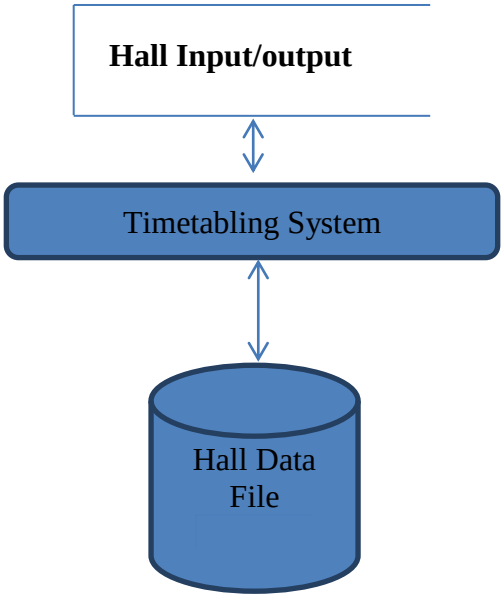


Figure 3.8.: Hall File Processing Diagram showing how the hall configuration file is accessed by the system.

- **Programs File**: contains the different programs in the university entered at input. See Table 3.2.

Table 3.2.: Programs Table

This table stores the program names used in the timetable allocations.

Field	Type
Program Name	String

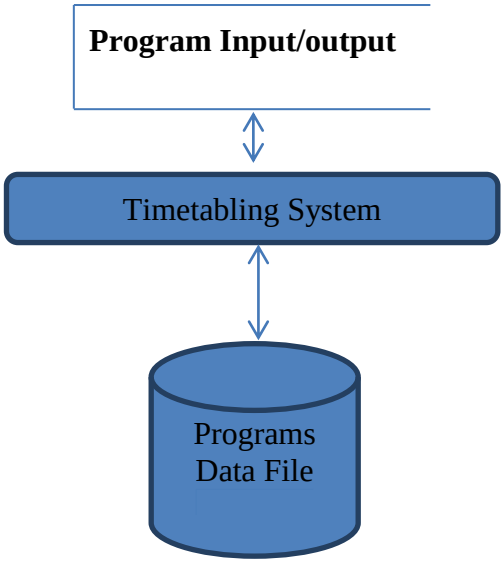


Figure 3.9.: Program File Processing Diagram showing how the programs configuration file is accessed by the system.

- **Buildings File**: contains the names of the building existing in the school for which the timetable is to be generated. See Table 3.3.

Table 3.3.: Buildings Table

This table stores the building names used in the timetable allocations.

Field	Type
Building Name	String

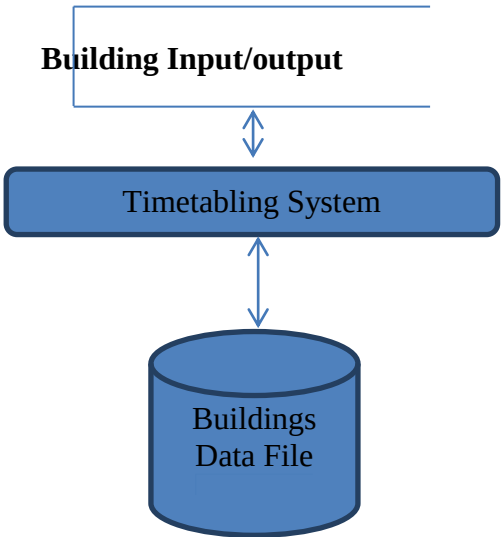


Figure 3.10.: Building File Processing Diagram showing how the buildings configuration file is accessed by the system.

- **Lecturers File**: contains the names of the Lecturers in the school. See Table 3.4.

  Table 3.4.: Lecturers Table

  This table stores the Lecturer names used in the timetable allocations.

Field	Type
Lecturer Name	String

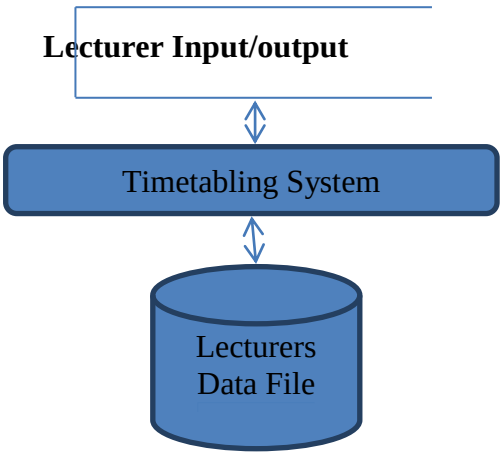


Figure 3.11.: Lecturer File Processing Diagram showing how the lecturer's configuration file is accessed by the system.

- **Departments File**: contains all the department names in the university. See Table 3.5.

  Table 3.5.: Buildings Table

  This table stores the building names used in the timetable allocations.

Field	Type
Department Name	String

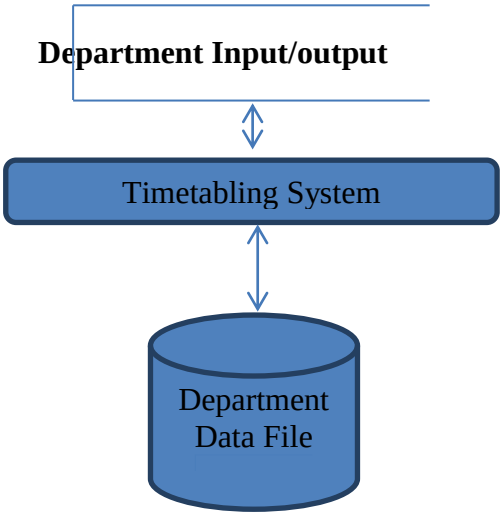


Figure 3.12.: Departments File Processing Diagram showing how the department's configuration file is accessed by the system.

- **Course Codes File**: contains all the course codes being offered in the school including their titles and units. See Table 3.6.

Table 3.6.: Buildings Table

This table stores the building names used in the timetable allocations.

Field	Type
Course Code	String
Unit	Integer
Title	String

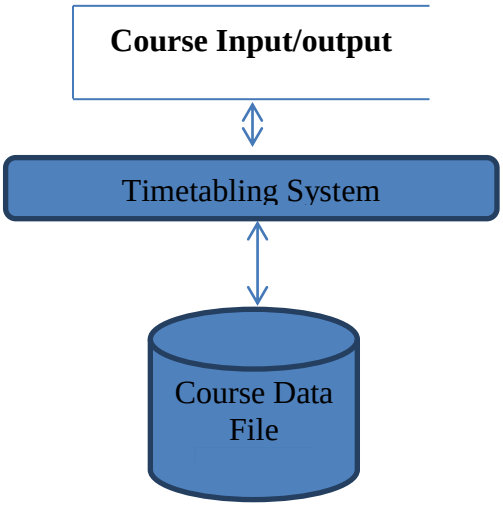


Figure 3.13.: Course File Processing Diagram showing how the course configuration file is accessed by the system.

3.7. 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" .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. 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. 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. 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.

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.

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" 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.

3.7 GENETIC ALGORITHMS

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 has 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 was 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.

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.

3.7.1 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.

3.7.2 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.

- 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".

- 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.

  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 tree. 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 subtree with another.

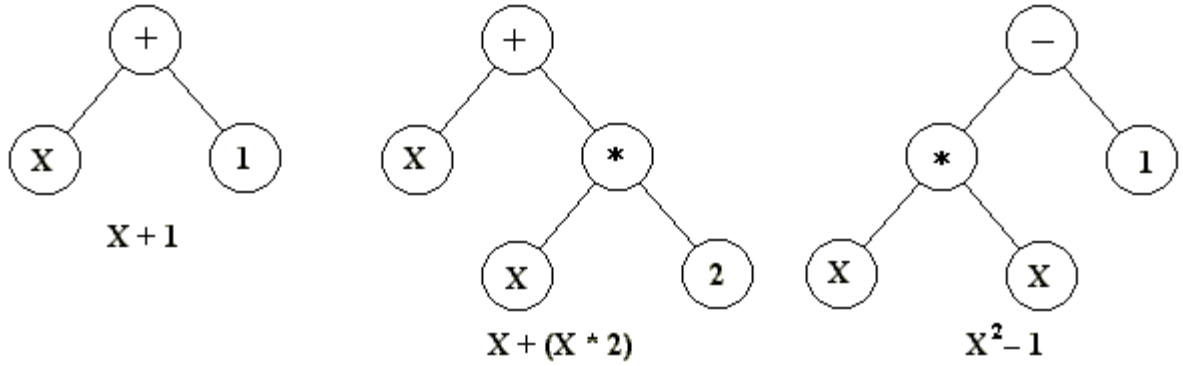


Figure 3.14.: Three simple program trees of the kind normally used in genetic programming. The mathematical expression that each one represents is given underneath it.

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.

3.7.3 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 don’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.

3.7.4 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 2.1.

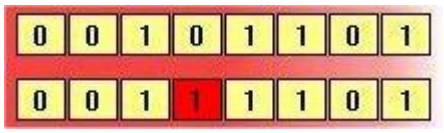


Figure3.15. Diagram showing the effect of mutation on an individual in a population of 8-bit strings where mutation occurs at position 4, changing the 0 at that position in its genome to a 1

- 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 .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 2.2.

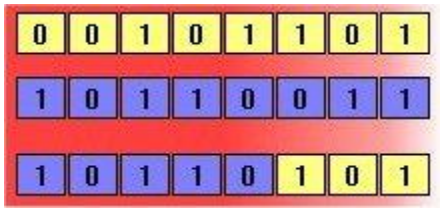


Figure3.16. Diagram showing the effect of mutation on individuals in a population of 8-bit strings showing two individuals undergoing single-point crossover; the point of exchange is set between the fifth and sixth positions in the genome, producing a new individual that is a hybrid of its progenitors.

3.7.5. 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.

- 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.

- 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.

- 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. 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.

- 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.

- 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 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.

- 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". 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.

- Another area in which genetic algorithms excel is their ability to manipulate many parameters simultaneously. 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 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.

- 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", 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 priority, therefore missing any novel solutions that may exist there. 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 are not adversely affected by ignorance. 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. Evolutionary algorithms, of course, are neither aware nor concerned whether a solution runs counter to established beliefs - only whether it works.

3.7.6. LIMITATIONS OF GENETIC ALGORITHMS

Although genetic algorithms have proven to be an efficient and powerful problem-solving strategy, they are not a panacea. GAs do 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.
- 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.

- 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.

- One type of problem that genetic algorithms have difficulty dealing with are problems with "deceptive" fitness functions, 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.

- 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. 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.
- Finally, several researchers 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.

3.7.7. 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.

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. 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. 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.

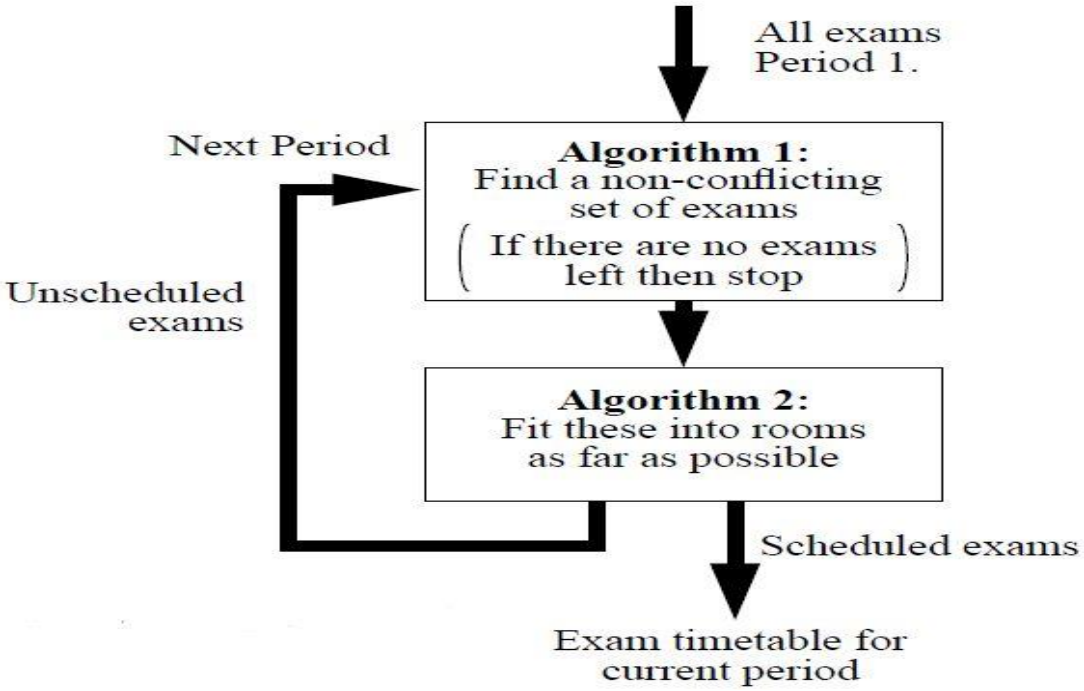


Figure 3.17.: Diagram depicting the Hybrid Genetic Algorithm used in University of Nottingham.

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.

CHAPTER FOUR

SYSTEM IMPLEMENTATION

4.1. INTRODUCTION

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.

4.2. CHOICE OF PROGRAMMING LANGUAGE

Java is the language of choice for this project because of its high speed and low memory usage. The timetabling problem is combinatorial in nature hence the need for a programming language that has enhanced CPU optimizing capabilities for the development of and algorithm like the genetic algorithm which optimizes search space and avoids local optima.

4.3. PROGRAM WRITING

The software implementation contains two major modules:

- **Crossover Module**: has a functionality of simulating the crossover of course population whose constraints are violated. Crossing over individual courses in the population attempts to reduce and or eliminate constraints violations. See Appendix A for source code.
- **Random Population Generation Module**: generates the initial random course population from the input supplied by user.

4.4. SYSTEMS REQUIREMENTS

Below are the conditions a computer system on which the timetable software will be run:

4.4.1. HARDWARE REQUIREMENTS

- Processor should be Pentium 5 and above

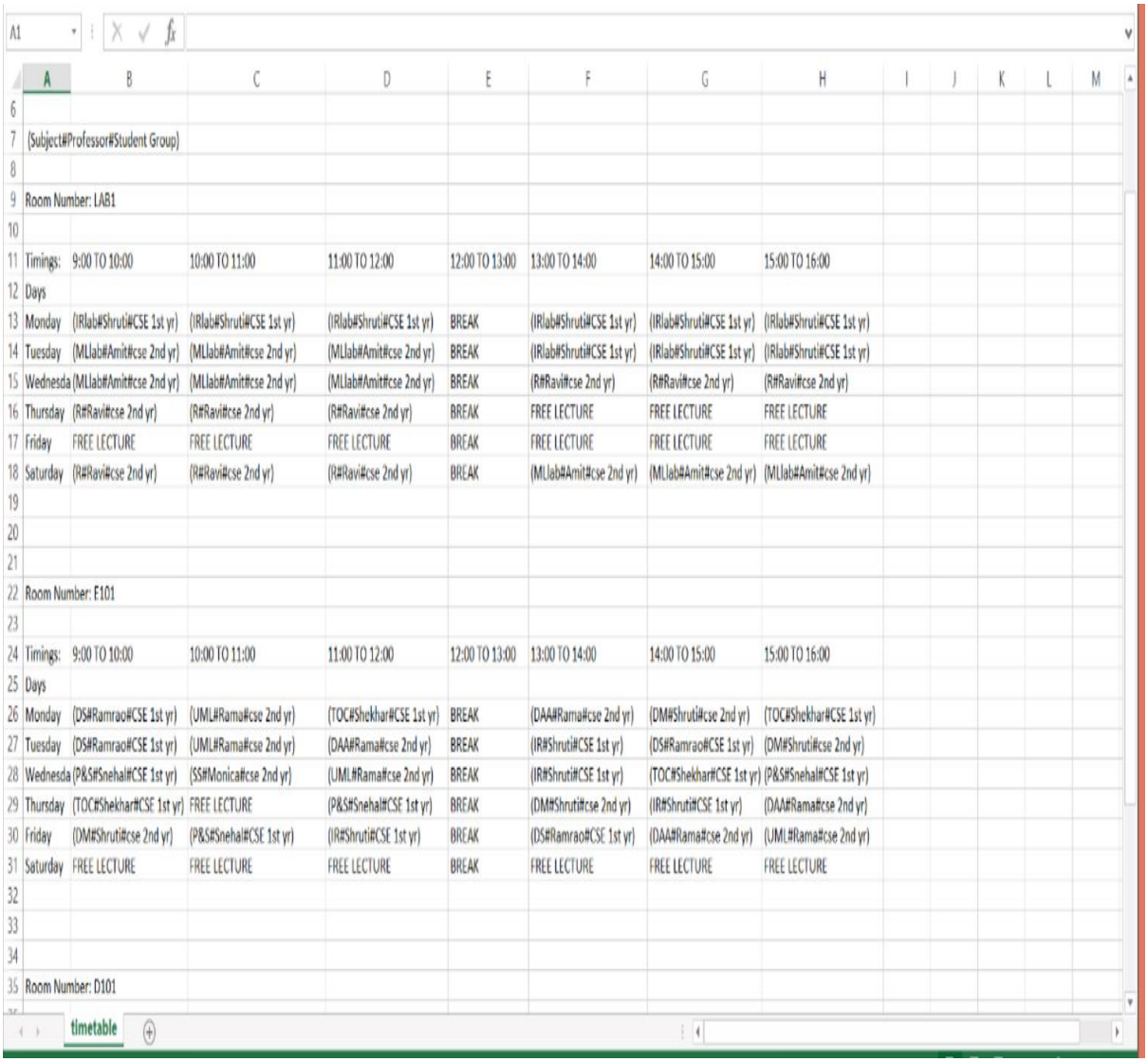
- 128 Megabytes of RAM (or more)
- 1 Gigabyte of Free Disk Space

4.4.2. SOFTWARE REQUIREMENTS

- Windows XP (or higher)

4.5. DOCUMENTATION

4.5.1. PROGRAM MODULES AND INTERFACE



	A	B	C	D	E	F	G	H
6								
7	(Subject#Professor#Student Group)							
8								
9	Room Number: LAB1							
10								
11	Timings:	9:00 TO 10:00	10:00 TO 11:00	11:00 TO 12:00	12:00 TO 13:00	13:00 TO 14:00	14:00 TO 15:00	15:00 TO 16:00
12	Days							
13	Monday	(IRlab#Shruti#CSE 1st yr)	(IRlab#Shruti#CSE 1st yr)	(IRlab#Shruti#CSE 1st yr)	BREAK	(IRlab#Shruti#CSE 1st yr)	(IRlab#Shruti#CSE 1st yr)	(IRlab#Shruti#CSE 1st yr)
14	Tuesday	(MLlab#Amit#cse 2nd yr)	(MLlab#Amit#cse 2nd yr)	(MLlab#Amit#cse 2nd yr)	BREAK	(IRlab#Shruti#CSE 1st yr)	(IRlab#Shruti#CSE 1st yr)	(IRlab#Shruti#CSE 1st yr)
15	Wednesda	(MLlab#Amit#cse 2nd yr)	(MLlab#Amit#cse 2nd yr)	(MLlab#Amit#cse 2nd yr)	BREAK	(R#Ravi#cse 2nd yr)	(R#Ravi#cse 2nd yr)	(R#Ravi#cse 2nd yr)
16	Thursday	(R#Ravi#cse 2nd yr)	(R#Ravi#cse 2nd yr)	(R#Ravi#cse 2nd yr)	BREAK	FREE LECTURE	FREE LECTURE	FREE LECTURE
17	Friday	FREE LECTURE	FREE LECTURE	FREE LECTURE	BREAK	FREE LECTURE	FREE LECTURE	FREE LECTURE
18	Saturday	(R#Ravi#cse 2nd yr)	(R#Ravi#cse 2nd yr)	(R#Ravi#cse 2nd yr)	BREAK	(MLlab#Amit#cse 2nd yr)	(MLlab#Amit#cse 2nd yr)	(MLlab#Amit#cse 2nd yr)
19								
20								
21								
22	Room Number: E101							
23								
24	Timings:	9:00 TO 10:00	10:00 TO 11:00	11:00 TO 12:00	12:00 TO 13:00	13:00 TO 14:00	14:00 TO 15:00	15:00 TO 16:00
25	Days							
26	Monday	(DS#Ramrao#CSE 1st yr)	(UML#Rama#cse 2nd yr)	(TOC#Shekhar#CSE 1st yr)	BREAK	(DAA#Rama#cse 2nd yr)	(DM#Shruti#cse 2nd yr)	(TOC#Shekhar#CSE 1st yr)
27	Tuesday	(DS#Ramrao#CSE 1st yr)	(UML#Rama#cse 2nd yr)	(DAA#Rama#cse 2nd yr)	BREAK	(IR#Shruti#CSE 1st yr)	(DS#Ramrao#CSE 1st yr)	(DM#Shruti#cse 2nd yr)
28	Wednesda	(P&S#Snehal#CSE 1st yr)	(SS#Monica#cse 2nd yr)	(UML#Rama#cse 2nd yr)	BREAK	(IR#Shruti#CSE 1st yr)	(TOC#Shekhar#CSE 1st yr)	(P&S#Snehal#CSE 1st yr)
29	Thursday	(TOC#Shekhar#CSE 1st yr)	FREE LECTURE	(P&S#Snehal#CSE 1st yr)	BREAK	(DM#Shruti#cse 2nd yr)	(IR#Shruti#CSE 1st yr)	(DAA#Rama#cse 2nd yr)
30	Friday	(DM#Shruti#cse 2nd yr)	(P&S#Snehal#CSE 1st yr)	(IR#Shruti#CSE 1st yr)	BREAK	(DS#Ramrao#CSE 1st yr)	(DAA#Rama#cse 2nd yr)	(UML#Rama#cse 2nd yr)
31	Saturday	FREE LECTURE	FREE LECTURE	FREE LECTURE	BREAK	FREE LECTURE	FREE LECTURE	FREE LECTURE
32								
33								
34								
35	Room Number: D101							

The above excel sheet shows us the time tale which has been generated by our software.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONSs

5.1. SUMMARY

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

5.2. CONCLUSION

Timetabling problem being the hard combinatorial problem that it is would take more than just the application of only one principle. The timetabling 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).

This research has been able to actualize a sub-implementation of a genetic algorithm which can be applied to input of 2-units courses.

5.3. RECOMMENDATIONS

In furtherance of this work, the following are recommended:

- The timetable system developed as the outcome of this project should be made open to avid students of computing who can collaborate and improve on the techniques and ideas inherent in this project.
- Further works on developing a timetabling system should be based on this research work so as to utilize the incremental model of software development.
- A collaborative model of timetabling system which utilizes a computer network can also be built which entails different departments and entities allocating courses and constraints concurrently while the system threads and reports clashes.

5.4. 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.

5.5. SCOPE FOR FURTHER WORKS

For a fully functional system, the genetic algorithm should be fully implemented by satisfying the following objectives:

- The mutation function in the genetic algorithm should be implemented in the system.
- The existing crossover module can be restructured to dynamically handle varied course units e.g. 1-units, 3-units and others as may be required.
- Error handling features should be introduced in the full implementation of the algorithm.

REFERENCES

1. A. Cornelissen, M.J. Sprengers and B.Mader (2010). "OPUS-College Timetable Module Design Document" Journal of Computer Science 1(1), 1-7.
2. 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.
3. Adam Marczyk (2004). "Genetic Algorithms and Evolutionary Computation ". Available online at http://www.talkorigins.org/faqs/genalg/genalg.html.
4. Al-Attar A.(1994). White Paper: "A hybrid GA-heuristic search strategy." AI Expert, USA.
5. Alberto Colorni, Marco Dorigo, Vittorio Manniezzo (1992). "A Genetic Algorithm to Solve the Timetable Problem" Journal of Computational Optimization and Applications, 1, 90-92.
6. 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.

7. 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.
8. 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.
9. Chan H. W. (1997). "School Timetabling Using Genetic Search." 2th International Conference on the Practice and Theory of Automated Timetabling, PATAT'97.
10. Coello Carlos (2000). "An updated survey of GA-based multiobjective optimization techniques." ACM Computing Surveys, 32(2), 109-143.
11. Costa D.(1994). "A tabular search algorithm for computing an operational timetable." European Journal of Operational Research, 76(1), 98-110.
12. Datta D., Deb K., & Fonseca, C.M.(2006). Multi-objective evolutionary algorithm for university class timetabling problem, In Evolutionary Scheduling, Springer-Verlag Press.
13. David A Coley (1999). An Introduction to Genetic Algorithms for Scientists and Engineers, 1st ed. World Scientific Publishing Co. Pte. Ltd.
14. Dawkins Richard (1996). The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design. W.W. Norton.
15. De Gans O.B.(1981). "A computer timetabling system for secondary schools in the Netherlands". European Journal of Operations Research,7, 175-182.
16. Deb K. (2001). Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons Ltd, England.
17. 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.
18. Eley M. (2006). "Ant Algorithms for the Exam Timetabling Problem." 6th International Conference on the Practice and Theory of Automated Timetabling, PATAT'06.
19. Fang H. L. (1994). "Genetic Algorithms in Timetabling Problems." PhD Thesis, University of Edinburgh.
20. Fernandes C. (2002). "Infected Genes Evolutionary Algorithm for School Timetabling." WSES International Conference.
21. Fleming Peter and R.C. Purshouse (2002). "Evolutionary algorithms in control systems engineering: a survey." Control Engineering Practice, 10, 1223-1241

APPENDIX

APPENDIX A: PROGRAM LISTINGS

CROSSOVER MODULE SOURCE CODE

```
procedure crossover( day, hall, time : integer);
var j,k,l, temp : integer;
label retry,segment;
begin
    randomize;
    j := random(5);
retry:
    randomize;
    k := random(high(hallrecord));
    randomize;
    l := random(10);
          if (allocations[j][k][l] = 0) then
            begin
                if (time mod 2 <> 0) then
                  begin
                      if (l mod 2 <> 0) then
                        begin
                            allocations[j][k][l] := allocations[day][hall][time];
                            allocations[j][k][l-1] := allocations[j][k][l];
                            allocations[day][hall][time] := 0;
                            allocations[day][hall][time-1] :=0;
                            allocations[day][hall][10] := allocations[day][hall][10]-2;
                            allocations[j][k][10] := allocations[j][k][10] + 2;
                        end
                      else
                         begin
                             allocations[j][k][l] := allocations[day][hall][time];
                             allocations[j][k][l+1] := allocations[j][k][l];
                             allocations[day][hall][time] := 0;
                             allocations[day][hall][time-1] :=0;
                             allocations[day][hall][10] := allocations[day][hall][10]-2;
                             allocations[j][k][10] := allocations[j][k][10] + 2;
                         end;
                  end
                else
                   begin
s         if (l mod 2 <> 0) then
                         begin
                             allocations[j][k][l] := allocations[day][hall][time];
                             allocations[j][k][l-1] := allocations[j][k][l];
                             allocations[day][hall][time] := 0;
                             allocations[day][hall][time+1] :=0;
                             allocations[day][hall][10] := allocations[day][hall][10]-2;
```

```
                    allocations[j][k][10] := allocations[j][k][10] + 2;
                end
            else
                begin
                    allocations[j][k][l] := allocations[day][hall][time];
                    allocations[j][k][l+1] := allocations[j][k][l];
                    allocations[day][hall][time] := 0;
                    allocations[day][hall][time+1] :=0;
                    allocations[day][hall][10] := allocations[day][hall][10]-2;
                    allocations[j][k][10] := allocations[j][k][10] + 2;
                end;
        end;
    end
else
    begin
        if (time mod 2 <> 0) then
            begin
                if (l mod 2 <> 0) then
                    begin
                        temp := allocations[j][k][l];
                        allocations[j][k][l] := allocations[day][hall][time];
                        allocations[j][k][l-1] := allocations[j][k][l];
                        allocations[day][hall][time] := temp;
                        allocations[day][hall][time-1] :=temp;
                    end
                else
                    begin
                        temp := allocations[j][k][l];
                        allocations[j][k][l] := allocations[day][hall][time];
                        allocations[j][k][l+1] := allocations[j][k][l];
                        allocations[day][hall][time] := temp;
                        allocations[day][hall][time-1] :=temp;
                    end;
            end
        else
            begin
                if (l mod 2 <> 0) then
                    begin
                        temp := allocations[j][k][l];
                        allocations[j][k][l] := allocations[day][hall][time];
                        allocations[j][k][l-1] := allocations[j][k][l];
                        allocations[day][hall][time] := temp;
                        allocations[day][hall][time+1] :=temp;
                    end
                else
                    begin
                        temp := allocations[j][k][l];
                        allocations[j][k][l] := allocations[day][hall][time];
                        allocations[j][k][l+1] := allocations[j][k][l];
                        allocations[day][hall][time] := temp;
```

```
                    allocations[day][hall][time+1] :=temp;
                end;
            end;
        end;
end;
```