DEVELOPMENT OF AN EXAMINATION TIMETABLING SYSTEM USING GENETIC ALGORITHM

By

ADESAGBA OLOLADE ELIZABETH

18010301074

A PROJECT SUBMITTED IN THE DEPARTMENT OF COMPUTER SCIENCE AND MATHEMATICS, COLLEGE OF BASIC AND APPLIED SCIENCES, IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF DEGREE OF BACHELOR OF SCIENCE.

2021

DECLARATION

I hereby declare that this project has been written by me is a record of my own research work. It has not been presented in any previous application for a higher degree of this or any other University. All citations and sources of information are clearly acknowledged by means of reference.

____

ADESAGBA, OLOLADE ELIZABETH

____

Date

CERTIFICATION

This Project titled, '**Development Of An Examination Timetabling System Using Genetic Algorithm'**, was prepared and submitted by **ADESAGBA OLOLADE ELIZABETH** in partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE.** The original research work was carried out by her under by supervision and is hereby accepted.

______________________________ (Signature and Date)

Prof. I.O Akinyemi

(Supervisor)

______________________________ (Signature and Date)

Matthew O. Adewole, PhD

Coordinator, Department of Computer Science and Mathematics.

DEDICATION

This project is dedicated to God Almighty.

ACKNOWLEDGEMENTS

I owe my profound gratitude to Almighty God who gave the strength, wisdom and courage, divine help and provision to me from the beginning to the completion of this work. I express gratitude to my supervisor, Prof. I.O Akinyemi, for his guidance and support in ensuring the successful completion of this project work. God bless you Sir.

I sincerely appreciate Mr Jeremiah Balogun, for his guidance and teachings. My heart-felt gratitude goes to the Head of Department, Computer Science and Mathematics – Dr. M.O Adewole and all other members of staff of the Department of Computer Science: Late Dr. Oyetunji M.O., Dr. (Mrs.) Kasali F.A., Dr (Mrs.) Oladejo Bola, Prof. Idowu P.A., Dr. Okunoye O.B., Dr. (Mrs.) Oladeji F.A., Dr. (Mrs.) Igiri, Mr. J.A Balogun, Mr. Ebo I.O and others too that I could not mention.

I acknowledge the constant support of my lecturer Mr. J.A Balogun who had contributed greatly to my academics. I pray God would continually make him a force to reckon with in his area of expertise and that God increases his knowledge.

I will forever be grateful to my parents Major and Mrs. Adesagba, who sacrificed wealth, time and other resources for the sake of my success; and my siblings, Adeola, Mayowa and Temilade then my cousins Sodiq and Rukayat for their love and support. I also want to appreciate my uncles, Mr. Adekunle Adeyeye, Mr. Phillip Adesagba and my aunts Dr. Tayo Adesagba and Mrs. Fadeke Adeyonu for their love and endless support. And my profound gratitude goes to Hunzlah Malik, Saad Bazaz, Abdul Rehman Subhani, Emmanuel Sule, Tolu Odey and Benjamin Akighbe for their help in making this project feasible and all my Mountain Top University colleagues and friends for their prayers and support, and help in one way or the other. God bless them all immensely.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The aim of this study is to apply genetic algorithm to the optimization of the timetable scheduling problem for the examination timetable based on the size of the registered students, the number of courses and the carryover courses of the students.

This was achieved by eliciting knowledge on identifying the requirements of the system, formulating the genetic algorithm model and the simulation of the genetic algorithm model.

A structured interview was conducted with the timetable representative in order to obtain quantitative data for the purpose of testing the system and the data collected consisted of sets of school configurations like the number of examination halls, the capacity of each hall, the number of timeslots per day etc.

Hard and soft constraint of the genetic algorithm were formulated based on the limitations and feedbacks of timetable representative. Anaconda Navigator was used as an Integrated Development Environment (IDE). The system was simulated using comma separated values (CSV) file which served as a storage capacity for all the quantitative data gotten. The program was written using the Jupyter notebook with python being the interpreter.

After representing the different constraints using mathematical modelling and the simulation of the GA using the python interpreter, the result has shown that the system is capable of providing useful solutions. It does not, however, fully automate the process. There are still some circumstances when the operator will need to make changes to some of the entries in order to achieve a flawless result. The enormous number of possible combinations for testing in order to arrive at an appropriate assessment for the application has proven to be impossible. However, considering the number of constraints set on the system, it can be inferred that the system was able to provide findings that, despite being imperfect, are valid and acceptable.

In conclusion, the system was developed to improve timetabling scheduling in general, this study suggested that the approach/technique to tackling the problem of genetic algorithms should be used. Although the experimental results show that a more efficient and dependable schedule can be reached with a properly constructed genetic algorithm. This offers good review schedules without conflicting examinations and in a much quicker time.

Keywords: *University Timetable scheduling, Examination, Genetic Algorithm, Modelling, Constraints.*

CHAPTER ONE

INTRODUCTION

1.1. Background to the study

Timetabling can be defined as the act of scheduling something to happen at a particular time. Timetable scheduling is one of the problems in the educational sector. An example of a scheduling issue is the university course and examination scheduling problem, which is NP-hard. The timetabling procedure must be completed for each semester on a regular basis, which is a time-consuming and demanding activity. (Hamed Babaei, 2015)

There are different algorithms that are used to solve timetabling problems, some examples are the ant colony algorithm, the bee algorithm, the genetic algorithm or any other type of hybrid algorithms. (Kadam, 2015). But in this study, the application of genetic algorithm is employed.

Genetic algorithms can be simply put, as heuristic techniques that are used to provide multiple possible solutions to a particular problem and finding that which optimally solves the problem. Genetic algorithms can be simple or complex. Like the natural evolution process itself, Life has identified a wide range of genetic information sharing methods. Genetic algorithms are known to solve multiple problems such as scheduling problems and optimization problems. (Mallawaarachchi, 2017).

Many studies have been conducted to determine which problems can be solved using genetic algorithms. Some of them are in Risk assessment, Bio-Medical problems, Minimum Dominating Set of Queens problem and the famous timetabling problem.

1.2. Statement of the problem

In the educational sector of today, manual methods of timetable scheduling are employed, which is the use of paper. There are a lot of problems faced in using this manual method due to the limited time slots, limited venues and the availability of the lecturers at that particular time and this poses as problems for course scheduling whereas in examination scheduling a lot of more difficulties are faced. For example, clashes between normal semester courses and carryover courses and having limited timeslots to fit theses examinations into. In recent studies, genetic algorithm is used in the implementation of examination timetabling system but without considering the clashes of the carryover courses taken by the students with other courses. This study will take into consideration the clashes of the carry-over courses taken by the students and it will be simulated in an examination timetable.

1.3. Aim and objectives of the study

The aim of this study is to apply genetic algorithm to the optimization of the timetable scheduling problem for the examination timetable.

The specific objectives are to;

a) identify the requirements of the system
b) formulate the genetic algorithm model
c) simulate the genetic algorithm model

1.4. Proposed methodology

In order for the aforementioned objectives to be actualized, the various methods will be adopted.

a) conduct an informal interview with the examination officer in charge of the timetables.

b) formulate hard and soft constraints of the genetic algorithm needed for the timetable scheduling problem.
c) the genetic algorithm will be simulated using the python programming language.

1.5. Scope and limitations of the study

This study is limited to the development of an examination timetable schedule for the computer science and mathematics department in Mountain Top university. This study will also be considering the carry-over courses across different levels in the department.

1.6. Significance of the study

<u>FOR MOUNTAIN TOP UNIVERSITY</u>

The system will be used to create not only an examination timetable but also a normal lecture/departmental timetable that efficiently allocates courses to lecture rooms when the lecturer will be available and at the proper time at that. It would not have to be adjusted frequently and it will save time and little effort will also be utilized. And this will be a major breakthrough for the university.

<u>FOR THE EDUCATIONAL SYSTEM AT LARGE</u>

Other educational institutions can also utilize this system and improve their time table's quality and save time.

1.7. Definition of terms

- **Heuristics:** is any strategy to problem solving or self-discovery that involves a practical method that is not guaranteed to be ideal, perfect, or rational, but is sufficient for obtaining an immediate, short-term objective or approximation.

- **Prototype:** is an early sample, model, or release of a product designed to test a concept or method. It is a phrase that is used in a variety of areas, including semantics, design, electronics, and software engineering.
- **Tabu search:** is a metaheuristic search method employing local search methods used for mathematical optimization.
- **Natural selection:** is the differential survival and reproduction of individuals due to differences in phenotype. It is a key mechanism of evolution, the change in a population's heritable traits through generations.
- **Timetable scheduling:** is selecting how to prioritize work and allocate resources among a multitude of options. (Hojjat Adeli, 2003)
- **Mutation:** is a modification that happens in the DNA sequence, either owing to mistakes when the DNA is duplicated or as the result of environmental influences such as UV light and cigarette smoke.
- **Crossover:** when two chromosomes, usually homologous instances of the same chromosome, split and subsequently reconnect but to different ends, this occurs.
- **Genetic algorithm:** modelled after the mechanism of natural selection, in which the fittest individuals are chosen for growth and reproduction in order to create offspring for the next generation.
- **Optimization:** making the finest or most efficient use of a situation or resource
- **Evolutionary algorithms:** is a subset of evolutionary computation, which is a type of population-based metaheuristic optimization method. An EA employs biological evolution-inspired techniques such as reproduction, mutation, recombination, and selection.
- **Search heuristic:** refers to a method of searching that tries to solve a problem by iteratively refining the solution using a heuristic function or cost measure.

- **Feasible solution:** is a set of decision variable values in an optimization problem that meet all of the constraints.
- **Selection:** Individual genomes from a population are selected for later breeding at this stage of a genetic algorithm.
- **Carryover courses:** refer to courses that are not passed and have to be retaken by the student in the consequent session.
- **Soft constraints:** refer to restrictions with certain variable values that are punished in the objective function if and to the degree that the variables' requirements are not met.
- **Hard constraints:** any successful model solution must satisfy this restriction.
- **Timeslots:** a period of time that has been given to someone or something
- **Metaheuristic:** is a higher-level process or heuristic for locating, developing, or selecting a heuristic (partial search algorithm) that can sufficiently solve an optimization problem.

CHAPTER TWO

LITERATURE REVIEW

2.1. Information system

An information system (IS) is formally structured to provide, process, store and disseminate information in a sociotechnical organizational structure (Piccolo & Pigni, 2018). Information systems comprise four elements: task, personnel, structure (or roles) and technology, from a socio-technological perspective. The academic analysis of the data-collecting, filtering, analyzing, producing and distributing data systems and its associated hardware and software nets are called information systems. Users, processors, storage, inputs, outputs, and the previously mentioned communications networks all constitute part of the information system (O'Hara, Watson, & Cavan, 1999). Information systems are defined as a set of components that work together to gather, store, and process data, with the data being utilized to give information, contribute to knowledge, and create digital products that help people make better decisions (Jessup & Valacich, 2008).

2.1.1. History of information systems

Information systems (IS) have only been around for five decades. Despite this, IS has done more than any other convention in history to expand business and industry into global marketplaces. The backbone of IS is currently known as the World Wide Web, Internet, or in the case of a business, a Local Area Network, as well as a slew of acronym buzz words like EDI, EIS, ERP, SCM, and a slew of others to explain new ways in which IS may be used to expand a business. Contrary to today's communication speed, just over four decades ago, the business climate in the United States was seeing post-war expansion like it had never seen before.(JMJ, 2000) Much of the knowledge that helped the economy grow was gained during World War II when the nation's industries were geared up to produce an effective war machine. The field of Operations Research arose as a result of this endeavor to win the war (OR). When the war ended, those involved with OR were freed from government service, releasing an experienced and highly trained field unlike any other in history into business and industry, ushering the United States into a period of wealth and expansion that lasted more than two decades. During World War II, the first functional computers, known as Turing Machines, were created, which were responsible for deciphering German codes and providing the allies with advanced warning of enemy operations (JMJ, 2000). These earliest practical computers were not particularly practical by today's standards, costing half a million dollars and being substantially less powerful than a pocket calculator, which can currently be purchased for less than ten dollars. These first computers, on the other hand, provided Operations Researchers with the ability to begin simulating larger and more complex systems, which in business and industry substantially aids in transforming capital expenditures into successful endeavors. This context from the early days of simulation, OR, and new technology inspired study into what became known as Information Systems (JMJ,2000).

2.1.2. Types of information system

In the 1980s, the "traditional" image of information systems in textbooks was a pyramid of systems that matched the organization's structure, with transaction processing systems at the bottom, management information systems, decision support systems, and executive information systems at the top. Although the pyramid model has remained helpful since its inception, a number of new technologies and kinds of information systems have arisen, some of which do not easily fit into the original pyramid model.

There are different types of information systems. Some of them are;

a) **Transaction processing systems**

A transaction includes all product and service purchases and sales, as well as any daily business transactions or operations required to run a business. Depending on the business and the size/scope of the organization, the quantity and types of transactions executed varies. Typical transactions include billing clients, bank deposits, new hire data, inventory counts, and a record of client-customer relationship management data. All contractual, transactional, and customer relationship data is maintained secure and accessible to all parties who require it, thanks to a transaction processing system. It also helps with sales order entry, payroll, shipping, sales administration, and other routine transactions that are necessary to keep companies running efficiently. Organizations can increase the dependability and quality of their user/customer data while reducing the risk of human mistake by implementing a TPS (Christiansen, 2021).

b) **Office automation systems**

Office Automation Systems An office automation system is a collection of tools, technologies, and people that enable clerical and managerial tasks to be completed. Printing documents, shipping paperwork, mailing, maintaining a company calendar, and providing reports are all common services handled by an OAS. An office automation system helps to improve communication between departments so that everyone can work together to finish a task. To ensure that all communication data is easily available and in one centralized area, an OAS can integrate with e-mail or word processing apps. Businesses can increase employee communication, expedite managerial processes, and maximize knowledge management by implementing an office automation system (Christiansen, 2021).

c) **Knowledge Management Systems**

A knowledge management system collects and organizes data to help users improve their knowledge and collaborate more effectively to perform tasks. Employee training materials, company policies and procedures, and replies to client questions are all examples of documents found in a knowledge management system. Employees, customers, management, and other stakeholders engaged with the firm use a KMS. It guarantees that technical skills are distributed throughout the organization while also giving graphics to assist employees in making sense of the data they are presented with. Workers who require outside knowledge to accomplish their duties can also use this information system to gain intuitive access to external information. A KMS, for example, may contain competitor data that aids a sales team member in optimizing his or her pitch to a customer. Using a KMS can improve communication among team members and aid everyone in meeting performance goals by sharing expertise and providing answers to key issues (Christiansen, 2021).

d) **Decision Support Systems**

A decision support system analyzes data to aid managers in making decisions. It collects and stores the data necessary for management to take the appropriate decisions at the appropriate time. A bank manager, for example, can use a DSS to examine changing loan trends and determine which annual loan targets to reach. The IS is built with decision models that evaluate and synthesize enormous amounts of data and provide it in a visual way that is easy to understand. Management may simply add or delete data and ask relevant questions because a DSS is interactive. This gives mid-management the evidence they need to make the best decisions possible to ensure the company fulfills its goals (Christiansen, 2021).

e) Executive Support System

Executive support systems are similar to decision support systems, except they are primarily used by executives and owners to help them make better decisions (Jackson, 1998). Enterprise leaders can use an expert system to obtain answers to non-routine issues, allowing them to make decisions that improve the company's outlook and performance. Unlike a DSS, an executive support system has superior telecommunication capabilities and more processing power. Data on tax regulations, new competitor startups, internal compliance issues, and other essential executive information is displayed using graphics software embedded into an ESS. This enables leaders to keep track of internal performance, keep tabs on the competition, and identify growth opportunities (Christiansen, 2021).

2.1.3 Student information systems

A student information system (SIS) is a management information system that is used in educational institutions to manage student data. It is also known as a student management system, school administration software, or student administration system. Student information systems allow educational institutions to register students for classes, document grading, transcripts of academic achievement and co-curricular activities, and the results of student assessment scores, create student schedules, track student attendance, and manage other student-related data requirements. Universities contain a variety of sensitive personal information, making them potentially attractive targets for security breaches similar to those faced by retail firms or healthcare organizations. (Gagliordi, 2014).

2.2. Scheduling

Scheduling is a method that is used to distribute valuable computing resources, usually processor time, bandwidth and memory, to the various processes, threads, data flows and applications that need them. Scheduling is performed to balance the load on the system, maintain equal allocation of resources, and provide some prioritization based on predefined rules. This assures that a computer system can service all requests while maintaining a particular level of service quality. In a production or industrial process, scheduling is the process of organizing, managing, and optimizing work and workloads. Plant and machinery resources are allocated, human resources are planned, production processes are planned, and supplies are purchased using scheduling. It's a crucial tool in manufacturing and engineering, where it can make a big difference in a process' productivity. The goal of scheduling in manufacturing is to reduce production time and costs by instructing a manufacturing facility when to manufacture what, with whom, and on what equipment. The goal of production scheduling is to increase the efficiency of the operation while lowering expenses.

2.2.1 Classification of scheduling

Scheduling can be broadly grouped into the following categories:

a) Semi-Active Scheduling

A schedule is considered semi-active if no job or operation can be completed sooner without affecting the processing order on any of the machines. By sequencing processes, these workable schedules are completed as soon as possible. There is no way to start an operation without first changing the processing sequences in a semi-active schedule (Rohini, 2016).

b) Active Scheduling

If it is not possible to design another schedule by changing the order of processing on the machines and having at least one job/operation finish earlier and no job/operation finish later, the schedule is called active (Nieberg, n.d). This viable schedules are those in which no process begins earlier than necessary without interruption or exceeding a precedence cap. Semi-active timetables remain in effect. In order to safely limit search space to the collection of active programs, an optimum technique is frequently used (Rohini, 2016).

c) Non-delay Scheduling

This viable schedules are those that have no interruptions in the machine's operation until it begins to function. Non-delay schedules must be active, so they're only semi-active. (Rohini, 2016). Job scheduling systems, parallel machine scheduling, group job scheduling, resource constraint scheduling, timetable scheduling, and dynamic task scheduling are all examples of scheduling systems (Rohini, 2016).

2.3. Types of scheduling systems

Scheduling has been applied to different areas and it has proven itself as effective. Below are some of the areas in which scheduling is utilized.

a) Project scheduling

Project scheduling in the service industries includes consulting projects, system installation projects, maintenance and repair projects, and so on. Annual auditing processes, which are required by every public corporation and must be done by independent accounting (CPA) companies, may also be incorporated in consulting assignments. A systems installation project could include the installation of a major computer system for a firm or the adoption of a large ERP system; these projects could take years to complete. Project scheduling has a wide range of applications in management consulting, accounting and auditing, and system deployment.

b) Workforce scheduling

Because schedules must be established in such a way that they can deal with unpredictable and random demand, workforce scheduling is a vital aspect of many service businesses. Nurse scheduling in hospitals, operator scheduling in call centers, and other application areas are examples. Workforce scheduling can be divided into two categories. The first is about shift scheduling, which is important in call centers, and the second is about crew scheduling, which is important in the transportation industry.

c) Timetabling, Reservations, and Appointments

In the hotel, education, and health-care industries, there are several timetabling, reservation, and appointment scheduling challenges. These problems are frequently mathematically linked, and similar solutions, such as integer programming formulations and graph theoretic techniques, may be required. In the hospitality industry, such as hotels and car rentals, interval scheduling challenges are common, although appointment scheduling is popular in many service industries, mostly to maximize resource use and eliminate queueing And timetabling is a general term for a set of scheduling issues that can be found in a variety of fields such as education, transportation, health care, and other service industries.

d) **Transportation Scheduling**

Transportation is a fundamental service that can take several forms depending on the mode of travel. Buses, trains, airplanes, and ships are among the different types of transportation available. Various modes of transportation have different planning horizons, restrictions, and objectives.

2.4. Timetable scheduling

According to Collins English Dictionary, "Timetabling can be defined as the act of scheduling something to happen or do something at a particular time". Timetabling is a well-known NP-Hard combinatorial optimization issue that has yet to be solved using a deterministic solution in polynomial time. To handle the timetabling problem, several strategies are utilized, including manual building, search heuristics (tabu search, simulated annealing, and evolutionary algorithms), neural networks, and graph colouring algorithms. Because most scheduling problems have application-specific properties, it is not uncommon to apply domain-specific patterns in conjunction with the majority of the aforementioned strategies to improve computing performance. (Walusungu 2014).

A school schedule is a combinatorial optimization problem that is structured as follows: Given a set of resources (lecture rooms, laboratories, etc.), a set of student groups, and a set of teachers, how can these three entities be organized in time such that given constraints are met while still satisfying optimality conditions. The most complicated timetables are found in universities, where the number of students and lecturers is high and enrolment into courses is guided by route maps. In such cases, allocating courses and lecturers to time slots and rooms necessitates the fulfilment of a number of potentially conflicting constraints. (Walusungu, 2014).

There are two types of constraints to consider: hard constraints and soft constraints. The former must be met in order for the timetable to be realistic (applicable), whereas the latter can be met to improve the timetable's consistency. Conflicts or collisions (an examination cannot take place in more than one venue, students can only attend one examination at a time), and capacity are examples of hard constraints (an examination must be allocated a venue with enough capacity).

Administrative requirements or individual/departmental desires are examples of soft constraints. Examination position and timing preferences, departmental room allocation preferences, and venue spacing are a few examples. (Walusungu 2014).

2.5. Timetable scheduling algorithms

There are different algorithms that are used to solve timetabling problems, they are:

a) **Ant colony algorithm**

A probabilistic method for solving computational problems is the ant colony optimization algorithm (ACO) which can be reduced in order to find good graphical paths. Artificial ants stand for the actions of real ants based on the multi-agent methods. The prevalent model of pheromone-based contact of organic aunts is also used. Combinations of artificial ants and local search algorithms have become a tool for choosing various optimizing activities, such as vehicle routing and internet routing. (Monmarché Nicolas, 2010)

procedure ACO_Metaheuristic is

While not terminated **do**

generateSolutions()

daemonSolutions()

pheromoneUpdate()

repeat

end procedure

Ant algorithm pseudocode (Awan-Ur-Rahman, 2020)

b) Bee algorithm

In 2005 the algorithm of the bee created Pham, Ghanbarzadeh and collaborators as a population-based research technique (Pham DT, 2005). It mimics the forging behaviour of honey bee colonies. The method performs a type of neighbourhood search, combined with global search, for combinatorial and continuous optimisation in the most basic version. The only criteria for the application of the bee approach is that some distance between the solutions can be specified. A variety of experiments have shown the efficacy and basic abilities of the bee's algorithm.

```
Initialise population with random solutions.
Evaluate fitness of the population.
While (stopping criteria not met) //Forming new population.
   Select elite bees.
   Select sites for neighbourhood search.
   Recruit bees around selected sites and evaluate fitness.
   Select the fittest bee from each site.
   Assign remaining bees to search randomly and evaluate their fitness.
End While
```

Figure2.5(a) Pseudocode of the bee algorithm (Pham DT, 2005)

c) Memetic algorithm

A memetic algorithm (MA) is a genetic algorithm that has been extended. It employs a local search technique to reduce the possibility of premature convergence (Garg, 2009). One of the most current and rapidly increasing branches of evolutionary calculation research are memetic algorithms. MA is now often applied to indicate a combination of evolutionary or other population-based approaches and independent human learning and local issue solving improvement techniques. MAs are known in the literature as Baldwinian evolutionary algorithms, Lamarckian EAs, cultural algorithms or local genetic searches (Moscato & Mathieson, 2019).

```
Memetic Algorithm Template
Begin
        INITIALIZE population;
        EVALUATE each candidate;
        Repeat Until (TERMINIATION CONDITION) Do
                SELECT parents;
                RECOMBINE to produce offspring;
                MUTATE offspring;
                IMPROVE offspring via Local Search;
                EVALUATE offspring;
                SELECT individuals for next generation;
        endDo
End
```

Figure 2.5(b) Pseudocode for the memetic algorithm (Majdi,2015)

d) Tabu search

A meta heuristics technique was developed to deal with large and complex combinatorial optimization problems (Ferland et al. 2000, Gendreau et al. 1994). This approach has been widely utilized to solve and build schedules due to the difficulties of timetabling as a combinatorial optimization problem. Regardless of its advantages and disadvantages (Brucker,1995), "Tabu search is an intelligent search method that uses a memory function to avoid becoming stuck at a local minimum and hierarchically canalizes one or more local search methods to swiftly identify the local optimality." Some previous information on the evolution of the search is retained to improve the effectiveness of the exploration process.

e) **Constant logic programming approach (CLP)**

For finite domains and huge combinatorial problems, CLP is widely and successfully employed. A schedule is a problem of the same kind with numerous resources (rooms and teachers) with particular schedule restrictions to achieve the optimum or close solution by allowing a maximum use of resources. The essential brilliance of CLP is its declarative handling of restrictions (both hard and soft). Because timetabling has been proven to be NP complete, despite the fact that a variety of software is available in the market, it is inflexible due to a variety of constraints in some situations, making it a tough topic of research (Murugan, 2009).

f) Genetic algorithm

Genetic algorithms were introduced as a ciphering analogy of adaptive systems. It is used for problem solving and for modelling (Murugan, 2009). A genetic algorithm is a search heuristic based on Charles Darwin's theory of natural selection. This algorithm is modelled after the mechanism of natural selection, in which the fittest individuals are chosen for growth and reproduction in order to create offspring for the next generation.

```
Create a population of objects (creatures) //Initialization
the fitness of each object //analysing
While the population is not fit enough //Fitness check
{
Delete all unfit objects //Removing unfit objects
While population size <max: //size check
```

{

Select two best populations

Create new objects

Random mutations

Evaluate and place in population //breeding

Evolutionary growth. (Murugan, 2009)

Genetic Algorithm is a particular order of evolutionary algorithms that uses the methodology of evolutionary biology such as mutation, crossover, selection and inheritance (Murugan, 2009).

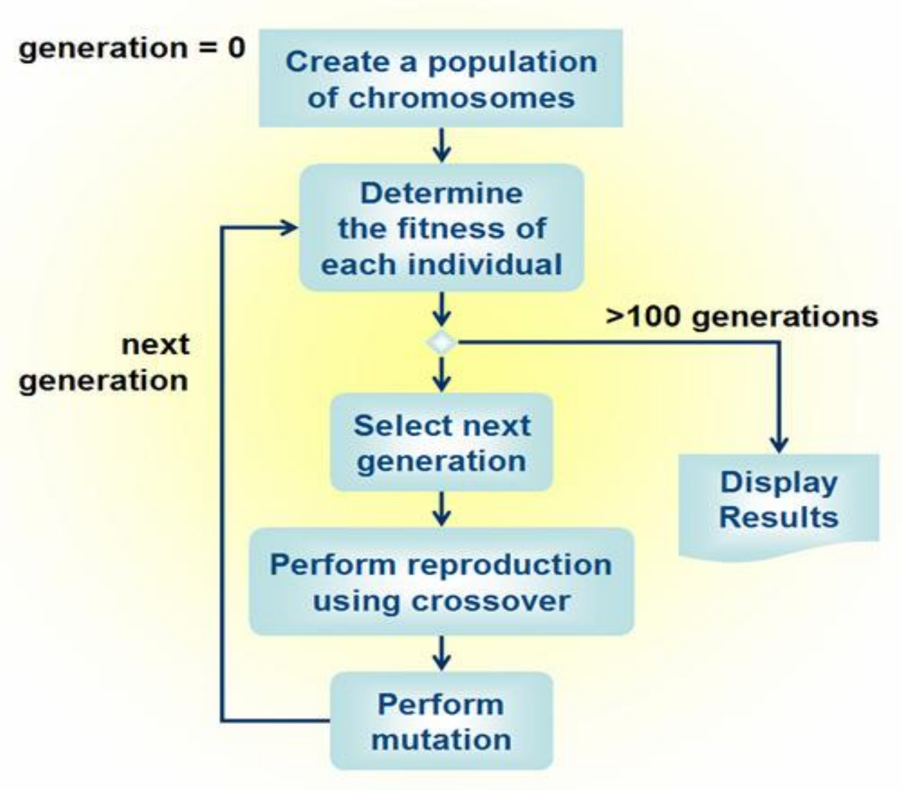


Figure 2.5(c) the basic genetic algorithm (Murugan, 2009)

2.5.1. The theory of natural selection

Natural selection begins with the selection of the fittest individuals from a population. They have children that inherit the traits of their parents and are passed on to the next generation. If parents are more fit, their children would be fitter than their parents and have a greater chance of survival. This method is repeated indefinitely until a generation of the fittest individuals is discovered. There are five cases to be considered in genetic algorithm.

They are:

a) **Initial population**

The phase starts with a group of individuals known as a Population. Each individual is a solution to the problem you wish to solve. A person is defined by a set of parameters (variables) known as Genes. To form a Chromosome, genes are linked together in a string (solution). A genetic algorithm represents an individual's collection of genes as a string in terms of an alphabet. Binary values are commonly used (string of 1s and 0s). We call this encoding the genes on a chromosome.

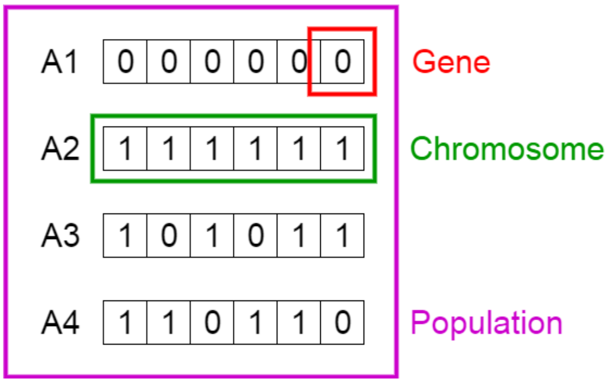


Figure2.5.1(a) representation of the initial population of a GA (Mallawaarachchi, 2017)

b) **Fitness function**

The fitness function defines an individual's level of fitness (the ability of an individual to compete with other individuals). It assigns a fitness score to each individual. The fitness score of an organism determines the likelihood that it will be chosen for reproduction. (Mallawaarachchi, 2017).

c) Selection

The concept behind the selection process is to choose the fittest individuals and allow them to pass on their genes to the next generation. Two pairs of people (parents) are chosen based on their fitness levels. Individuals with high fitness have a better chance of being chosen for reproduction. (Mallawaarachchi, 2017)

d) Crossover

The most important step of a genetic algorithm is crossover. A crossover point is selected at random from within the genes for each pair of parents to be mated.

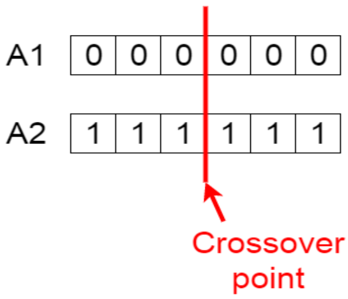


Figure 2.5.1(b) representation of the crossover point in the GA

Offspring are created by exchanging the genes of parents among themselves until the crossover point is reached.

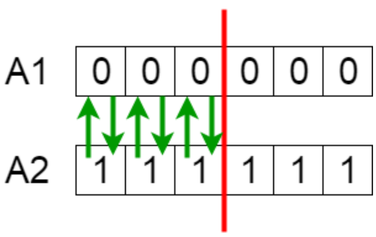


Figure 2.5.1(c) representation of the exchanging genes among parents

The new offspring are incorporated into the population.

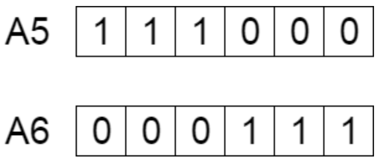
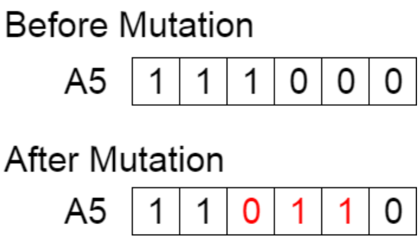


Figure 2.5.1(d) representation of the new offspring (Mallawaarachchi, 2017)

e) Mutation

Any of the genes of certain newly developed offspring may be exposed to a mutation with a low random probability. This means that some of the bits in the bit string are able to be flipped.



2.5.1(e) representation of the mutation: Before and After

(Mallawaarachchi, 2017)

2.6. Fitness function

The fitness function is a function that, when given a solution, determines whether it is good or bad. The answer improves as the fitness function's return value drops. The fitness function might simply check for any constraints that have been violated and return infinity if this is the case. It returns 0 if no limitations are violated. The fitness function provides no information about how excellent or awful the solution is, which is a flaw in that method. To gain a feel of how excellent or poor a solution is, the fitness function should return a value proportionate to the number of constraints violated. There are two sorts of limitations: hard and soft constraints. Hard constraint violations will not be permitted. As a result, violating a single hard requirement while also violating a large number of soft constraints is worse than meeting all soft constraints while also violating a single hard constraint. The fact that there are two types of constraints does not imply that there are only two penalty values. Some of the soft limitations turn out to be more important than others. Staff members' seniority, for example, suggests that those with higher seniority should be comfortable first, followed by those with lower seniority. To account for the severity of the breach, soft constraints must have varied punishments. Hard constraints, on the other hand, all bear the same penalty: breaking any one of them indicates the solution is infeasible and unacceptable. As a result, the fitness function still indicates whether a solution is good or bad (Ahmed F. AbouElhamayed, 2016).

$$fitness\ value = P_h * N_h + \sum_{n=1}^{N_s} P_s(n)\ ;\ where\ P_h > \sum_{i=1}^{N_s} P_s(i)$$

P_h *is penalty of violating a hard constraint.*

$P_s(i)$ *is the penalty of violating soft constraint number* i.

N_h *is the number of hard constraints.*

N_s *is the number of soft constraints.*

Figure2.6 Mathematical model of the fitness function

2.7. Review of related works

(Hong Siaw Theng, Abu Bakar Bin Md Sultan and Norhayati Mohd Ali, 2015) operated on a hybrid case-based reasoning approach to solving the university timetabling problem They developed the case-based approach as well as the classical and model methods to case-based reasoning. They used real-life test cases, which were past timetables obtained from the department of computer science at the University of Putra in Malaysia, for their study methodology. The timetables were then restructured into Database Management System (DBMS) format with suitable connections. They used the PHP web-based scheduler for their paper test. The DBMS of their experiment was MySQL, and the webhosting service that was used was APACHE. Their hardcoded text was converted into a database using Regular Expression Replacement (RegEx). Notepad++ served as the Integrated Development Environment (IDE). The entire experiment was based on actual past databases, and the results were planned to be compared for current accuracy, making the experiment a real-time test bed. The experiment was carried out on a high-performance desktop computer. The desktop's performance specifications were 3.4 Ghz i7-Quad Core (Hyper-thread) Intel processor and 8GB RAM. They conducted experiments with datasets obtained from the department of computer science at the University of Putra in Malaysia. The Human Preference Adaptable Retrieval Approach was used (HPARA). They demonstrated accuracy while generating a new schedule by obtaining lower counts of soft constraint violation. The proposed method's results demonstrated that there will be no soft constraint violation with a lower number of components. They also discovered that time taken results were significantly improving for schedule plotting with an average rating of 200ms. Their experiments demonstrated that the HPARA approach was successful in terms of both accuracy and reduction in solving scheduling in University timetabling. However, the component-by-component approach was used instead of the retrieval method used in their paper. This posed a problem in terms of reducing retrieval processing time.

(R Raghavjee, N Pillay, 2014), utilized a selection perturbative hyper-heuristic in solving the school scheduling problem was investigated. On various types of school timetabling problems, a genetic algorithm selection perturbative hyper-heuristic was implemented and tested. The hyper-heuristic generated feasible timetables for all problem instances and provided a generalized solution to the school scheduling problem, outperforming other methods applied to the same set of problems. They also discovered that the hyper-heuristic genetic algorithm outperformed the genetic algorithm applied directly to the solution space.

Another body of work done by (Rakesh P. Badoni and D.K. Gupta, 2015) combined the use of Genetic Algorithm (GA) and the Iterative Local Search (ILS) which resulted in Genetic Algorithm Iterative Local Search (GAILS) was described for solving the University Course Timetabling Problem (UCTP) It was based on ILS using three types of neighbourhood moves and four types of perturbations. This allowed them to develop each generation produced by GA. It was demonstrated that GAILS provided optimal solutions with zero fitness function values in all small problem instances within nanoseconds. Comparisons were made in order to demonstrate the efficacy of their proposed algorithm. When used differently, they suggested that GAILS is a better algorithm than GA and ILS.

In contrast to (Nashwan Ahmed Al-Majmar, Talal Hamid Al-Shfaq, 2016), which demonstrated that the use of Genetic algorithm (GA) was a powerful method for solving timetabling problems, especially with some suggested improvements. By combining multiple binary variables into one gene value on the chromosome, the initial timetabling problem with a large number of binary variables was greatly reduced to an appropriate scale. They created their software application in C# and used a SQL SERVER database to store and archive timetable data for future use. The model tested the method's efficacy and functionality using real-world datasets. The software model was very useful because it generated various types of timetables and contained a strong mix of artificial intelligence and software engineering. Their only drawback was that real-world teaching scheduling issues were not addressed.

Research by (H. M. Sani, 2016) also applied GA on ETP. They used a sample exam's data of college of education located in Sokoto and grouped them into three scenarios; 40, 100 and 200 exams, respectively. They scheduled these exams into 36 timeslots (2 weeks of exams from Monday to Saturday with 3 time slots in 1 day). In Sokoto, students are allowed to take elective courses. The elective courses will increase the possibility of clashes since students are free to choose many other courses. They ran the GA by using the ECJ toolkit. The ECJ toolkit is a software system that is specially designed for GAs and provides most of the standard components needed. Their aimed to avoid clashes (hard constraint) and also avoid students having two consecutive exams on the same day (soft constraint), but the result shows that the timetable produced had two exams scheduled on the same day. This means that the ECJ toolkit is only suitable to schedule courses or subjects which are not elective courses.

(Farah Adibah Adnan, 2018), unlike (H. M. Sani, 2016) specified the hard and soft constraints for the examination timetable. A weighted penalty value was attached to each violation of the soft constraint and the objective was to minimize the total penalty value of those violations. The paper was focused on applying the combination of the use of heuristic methods and the filtering of overlapping courses. There was no implementation of a timetable but they created mathematical models to satisfy the constraints.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

When it comes to scheduling timetables, it is always referred to be a difficult optimization issue that has been proved to be related to the clique of minimization problem, which is also referred to as NP complete. When faced with a problem for which there is no efficient algorithm available, it is desirable to apply a genetic algorithm to the problem, which is used to search for a solution space in the first place. It is important to recognise that this type of scheduling is a global problem that has immediate relevance in a variety of timetabling situations, including typical course timetables, examination timetables, and public transportation schedules.

3.1.1. Method of identification of user and system requirements

When it comes to identifying user and system needs, there are a variety of approaches that can be applied. The following are the sources of identification that were used in this project:

a) **Primary source:** It refers to the method of data collection, which in this case was an empirical approach, which was a personal interview with the timetabling officer, which was utilised to gather the information.

b) **Secondary source:** refers to information obtained from publications such as journals, conferences, websites, and books.

Functional, service, and operational restrictions of the software system are described in further detail in system requirements documents (also known as system specifications). The computer system is composed of various components that work together to accomplish a specific aim.

The following are the requirements that must be met in order for the system to be implemented.

a) **Non-functional requirements**

These are limitations on the services that the system is able to provide. They include time limits as well as restrictions imposed by industry standards. The following are the non-functional requirements that must be met by this system.

i. When it comes to preparing error-free timetables, the system must be up to speed.

ii. A flexible mechanism is required

b) **Functional requirements**

It is a list of services that the system must deliver, as well as how the system should react to specific inputs and how the system should behave in specific circumstances. The following are the functional requirements that must be met by the system:

i. The system must be able to create an initial population when at initial stage for creating a schedule
ii. The system must be able to assign a fitness for resolving clashes when two or more exams allocated to the same time

c) Hardware requirements

For reliable and productive project efficiency, certain hardware requirements must be met which are as follows:

i. 250 GB hard drive
ii. an Intel i5 processor
iii. 4 GB ram size

d) Software requirements

For efficiency of use and to have system performance in developing of software the following various software requirement must be met.

i. Operating system: Mac OS, Windows, Linux etc.
ii. Python interpreter
iii. Anaconda Navigator

3.1.2 Requirements gathering

During the system design phase, the quantitative technique of collecting and analyzing data for the purpose of testing the system was chosen, and the data to be collected consisted of sets of school configurations that were collected. Which are as follows:

i. The number of halls,
ii. The capacity of each hall,
iii. The total number of exams to be administered,
iv. The amount of time-slots per day,

v. Population of the students for each course,

vi. The number of days for examination.

3.1.3 Constraints

It is necessary to deal with constraints since they represent barriers to progress. It is not a case of literal disagreement. For example, because to time constraints, some events (such as the project's conclusion) must take place on specific dates in order to be completed. Resources are nearly always a limitation, because they are not available in a limitless supply in a free market. There are two sorts of limitations that will be addressed in this project: structural and operational restrictions.

They are;

a) Hard constraints

Hard constraints play a major role in arriving at an optimized timetable. They are;

i. Exam Population (registered students for the exam) must be less than the venue capacity i.e All events are to be assigned to rooms having adequate seating capacity and all the required features

ii. Students in the same class cannot take exams in different venues at the same time slot.

iii. No two courses taken by the same student group or registered as a carryover course can take place at the same venue and time slot.

iv. Each course should be assigned to a timeslot.

v. Final period on Wednesdays is not permitted.

b) Soft constraints

Soft constraints do not adversely affect the quality of the timetable but they have considerable influence on the output of the system. The considered soft constraint are

i. Students can have exams in consecutive time slots in a day.
ii. No gaps between examinations.

3.2. Genetic Algorithm based approach

Genetic algorithms (also known as adaptable evolutionary algorithms) are evolutionary algorithms that are inspired by nature and can be used to solve complex problems as well as search vast problem spaces. According to GA, every possible solution is considered a "person," and a significant number of such individuals or a collection of solutions compose the "population" at the end of every generation. Random selection can be used to generate the first set of solutions (referred to as the "initial population"); individuals are then randomly mated, allowing for the recombination of genetic material. Individuals resulting from this method can then be mutated with a particular mutation probability assigned to each of the individuals. Natural selection is then applied to the new population, favouring the survival of better solutions and serving as the starting point for a new evolutionary cycle to begin. A database of feasible timetables is maintained and updated on a regular basis. The most effective timelines are selected as the foundation for the next iteration or generation of the product. Basic operators such as selection, mutation, and crossover are employed in order to produce the best possible results.

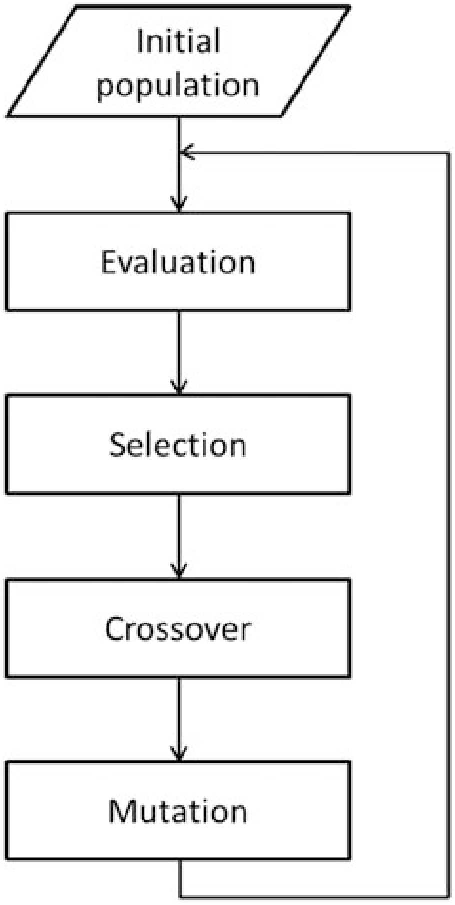


Figure 3.2 Flowchart of a simple of genetic algorithm

The first genetic operator, selection, aims to increase the number of copies of people who have better fitness values than those who have lower fitness values. It is also referred to as "roulette wheel" selection because it produces more copies of people who have better fitness values than those who have lower fitness values. Due to the fact that the mechanism is similar to that of a roulette wheel, it has gained popularity. The roulette wheel is spun in order to generate the future generation, with the large chunk indicating high fitness and the small piece representing low fitness reflecting the two extremes of fitness. So, the segment with the highest surface area stands a better chance of being selected as the next generation. The second operator is referred to as a crossover. As a result of the selection process, it generates two new individuals (parents). Among the different types of crossover operations are the one-point crossover, two-point crossover, cycle crossover, and uniform crossover. Mutation is regarded as the final operator in GA, and it occurs frequently after a crossover event. It makes a random change to a single bit of the bit string. This is accomplished by selecting a random value from the bit string and then transforming that value into another value (Mitchell 1997). The binary scenario, for example, converts the selected value from 0 to 1 or vice versa if the selected value is 0.

3.3. Model formulation

Variables

a) A set of students, $S=\{s_1,s_2\ldots,s_i\}$

The students in the department of computer science and mathematics.

b) A set of examination venues, $E=\{e_1,e_2,e_3\}$

The examination halls for the examinations to be held.

c) A set of courses , $C=\{c_1,c_2,c_3\ldots,c_k\}$.

Courses offered by students in the department for the semester

d) A set of students taking the same exams, $z=\{z_1,z_2\ldots,z_l\}$

$$Z_l = \sum s_i \longrightarrow C_k. \quad \text{defined as } Z(s) = C$$

e) A set of examination taking place, $A=\{a_1,a_2...a_m\}$

$\alpha: Z_k \longrightarrow A_m$. A group of students taking the same exam.

f) A set of time slots, $T=\{t_1,t_2\ldots t_n\}$ is the number of timeslots available daily except on Wednesday which has two slots. Examination will hold for two weeks leads to a total timeslot of 28 timeslots due to the exclusion of the final period Wednesdays.

Where T_1 = Monday first period, T_2 = Monday second period, T_3 = Monday third period etc.

3.4. Problem definition

To allocate an examination to a specific time-slot of a particular day in a particular venue.

3.4.1. Model

Each examination is A_m

Time slot is T_n

Venue E_j

$$\phi : A_{mn} \longrightarrow E_j$$

Such that:

$|Z_k| \leq |E_j|$ fulfils the constraint which says size of students taking a course must be less than capacity of venue

$\alpha_{jn}(Z_1) \neq \alpha_{jn}(Z_2)$ fulfils the constraint which says no two courses can take place at the same venue and time.

α_{1n} $(Z_1) \neq \alpha_{2n}$. fulfils the constraint which says students cannot take exams at different venues at the same time slot, n.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1. Introduction

This section presents the result and the discussion of this study which involved the development of an examination timetabling system using genetic algorithm. The chapter presents the Comma separated values (CSV) files where the list of courses, venues and their capacities are stored as well has the various modules used for the simulation of the GA.

4.2. Representation of the data used in the simulation of the Genetic Algorithm

The figure 4.2.1 shows the test values that are passed into the genetic algorithm for a simulation test of the implemented genetic algorithm. As a result of this, a CSV file was implemented called the Courses.csv file which were required for managing the various courses to be displayed in the timetable.

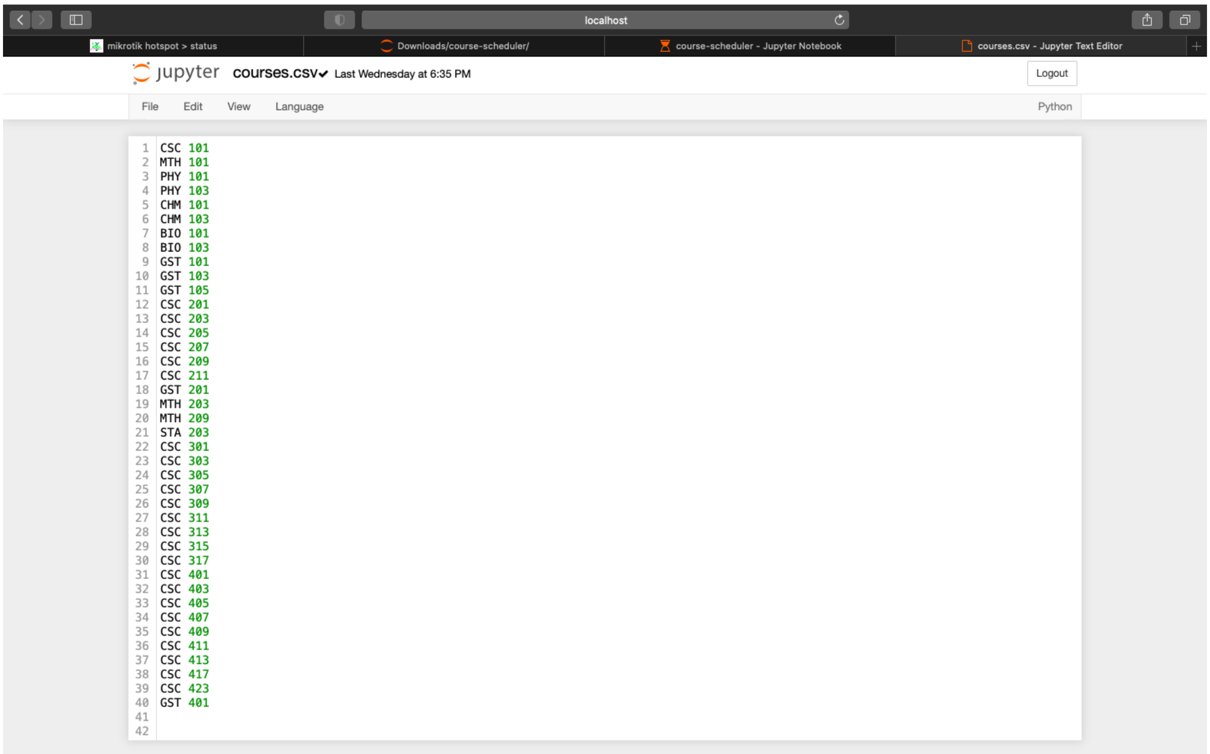


Figure 4.2.1 displays the courses passed into the GA

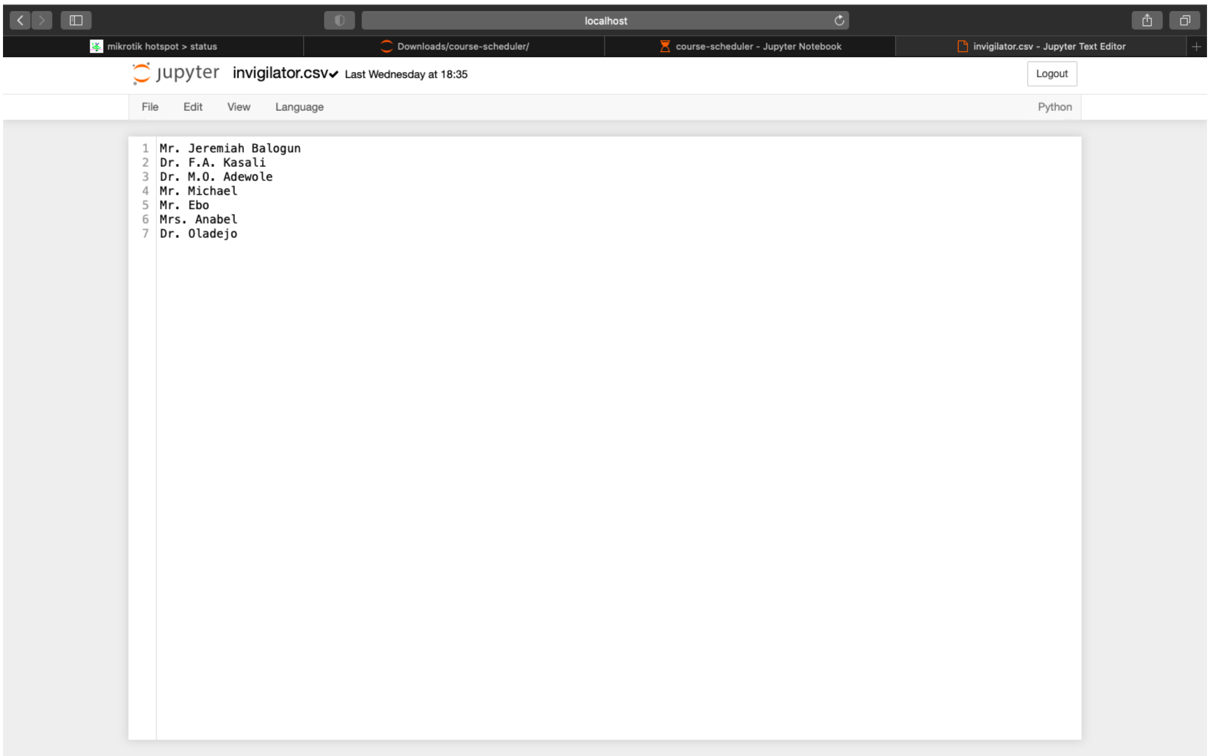


Figure 4.2.2 shows the name for the invigilators to be used in the GA

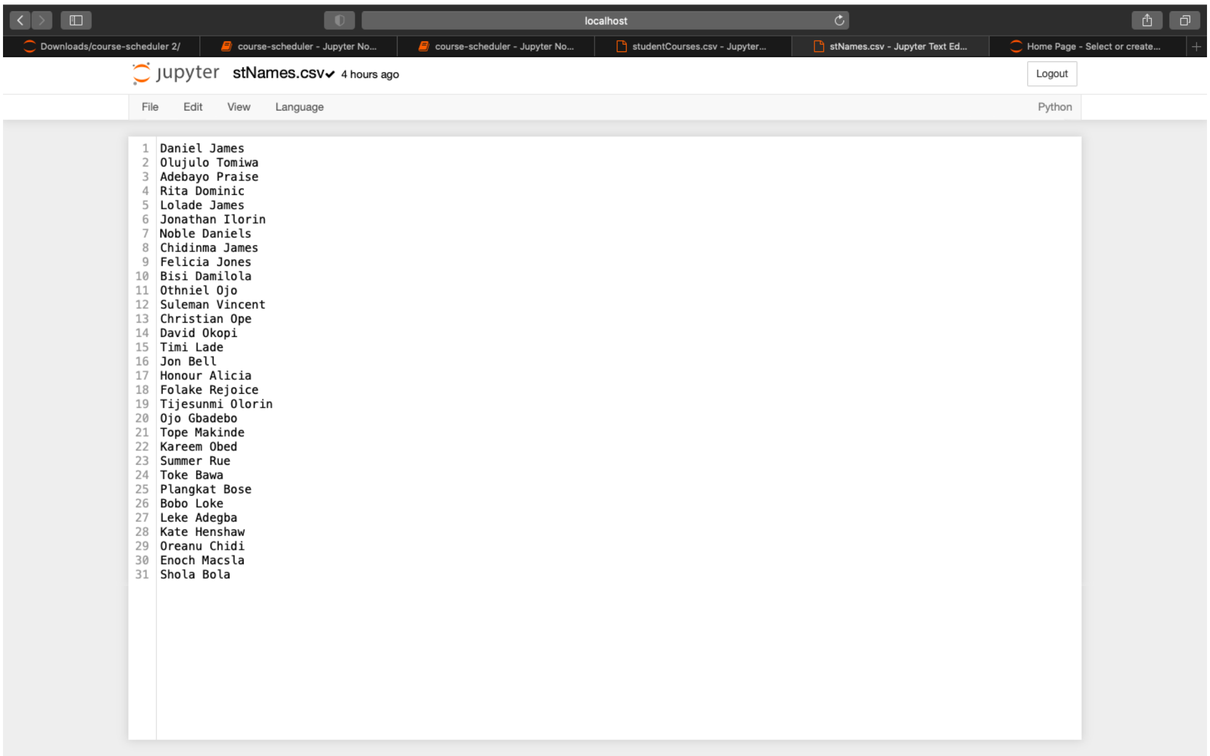


Figure 4.2.3 shows the student names to be used in the timetable

In the figure 4.2.4 below, students and the various courses that they registered for are displayed. Many names occur more than once and some students have courses that are seen as foreign. These courses are recognized as carry over courses.

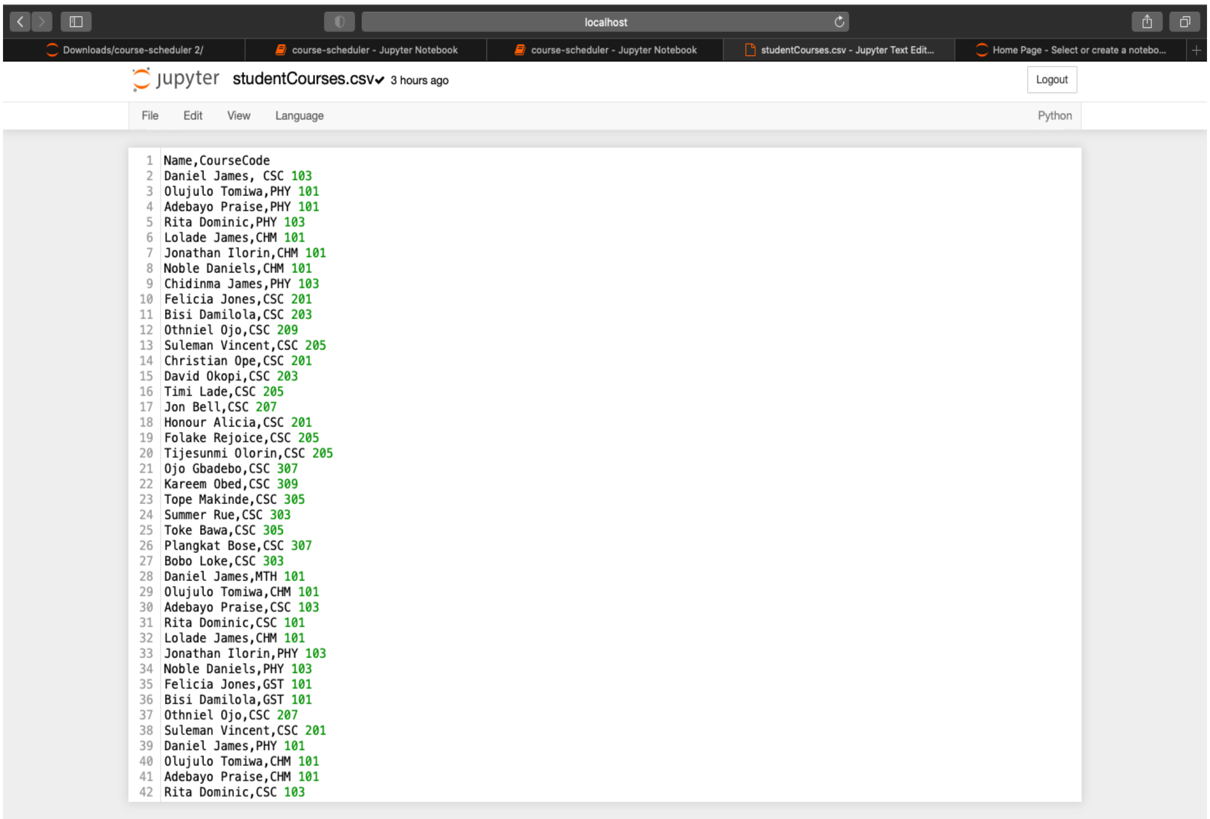


Figure 4.2.4 shows the students and the courses they registered.

4.3 Requirements for the simulation of the Genetic Algorithm

The figure 4.3.1 below shows the various functions that make the simulation of the genetic algorithm feasible. The panda function or pandas dataframe consists of rows and columns so, in order to iterate over dataframe, we can iterate a dataframe like a dictionary, the NumPy function contains a vast number of diverse mathematical operations. NumPy includes standard trigonometric functions, functions for arithmetic operations, managing complex numbers, etc. Standard trigonometric functions in NumPy return trigonometric ratios for a given angle in radians.

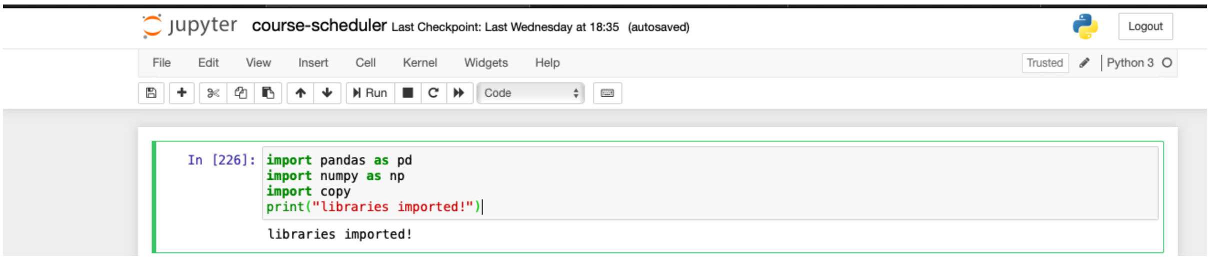


Figure 4.3.1 displays the methods used

4.4 Loading of data into the genetic algorithm program

For the timetable to actually work, all the data stored in the csv file has to be loaded into the GA program. Figure 4.4.1 displays the first five student names present in the CSV file. Figure 4.4.2 displays the first five courses present in the CSV file and this was called using the . head function. Figure 4.4.3 displays the first five invigilator names that will be used in the GA.

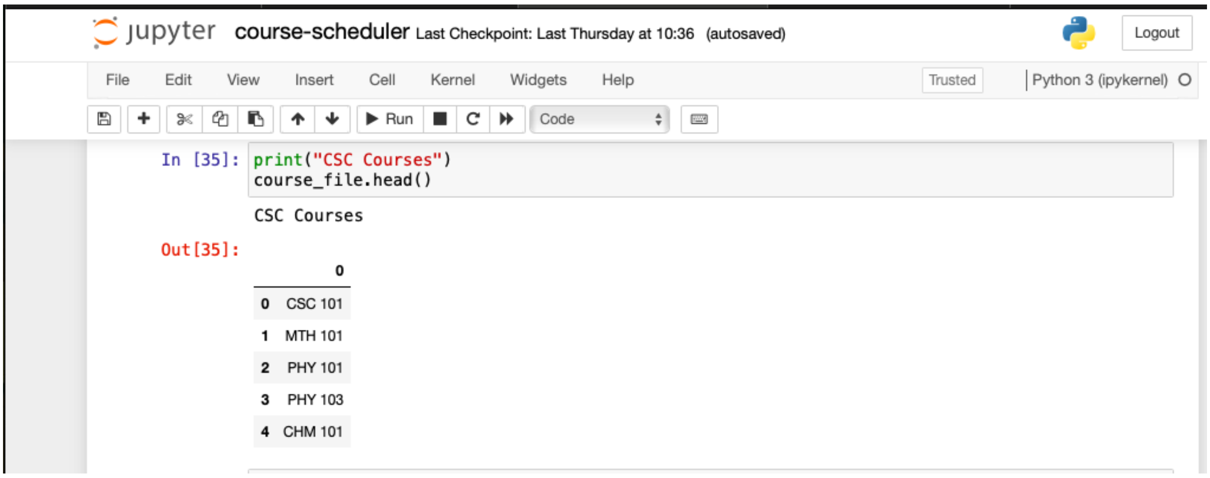


Figure 4.4.1 displays the first five student names

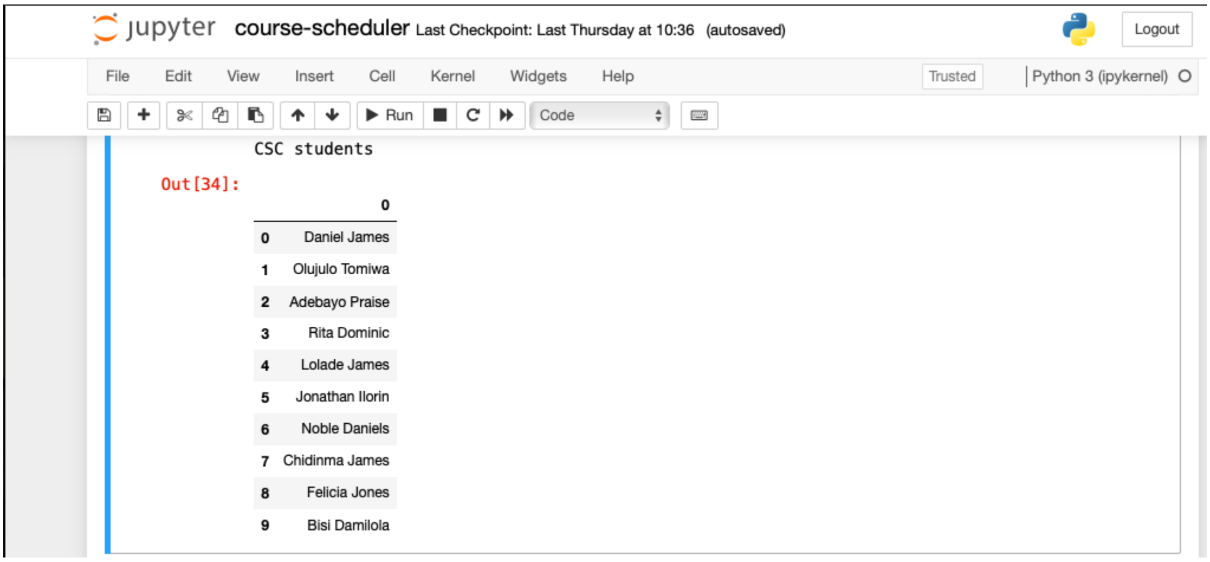


Figure 4.4.2 displays a list containing 10 student names

4.5 Population model

Genes - Chromosome - Population. Each course, instructor, students, day, time, classroom name were stored as genes. Random.randint was utilized for the generation of random data from each array. Gene is a full schedule entry. It contains:

a) Day of exam,

b) Start time of exam,

c) End time of exam,

d) Invigilator (a lecturer),

e) A list of students taking the exam,

f) A classroom

Chromosome is made up of a range of genes. A range of chosen chromosomes are present in the population. The schedule is established for each chromosome.

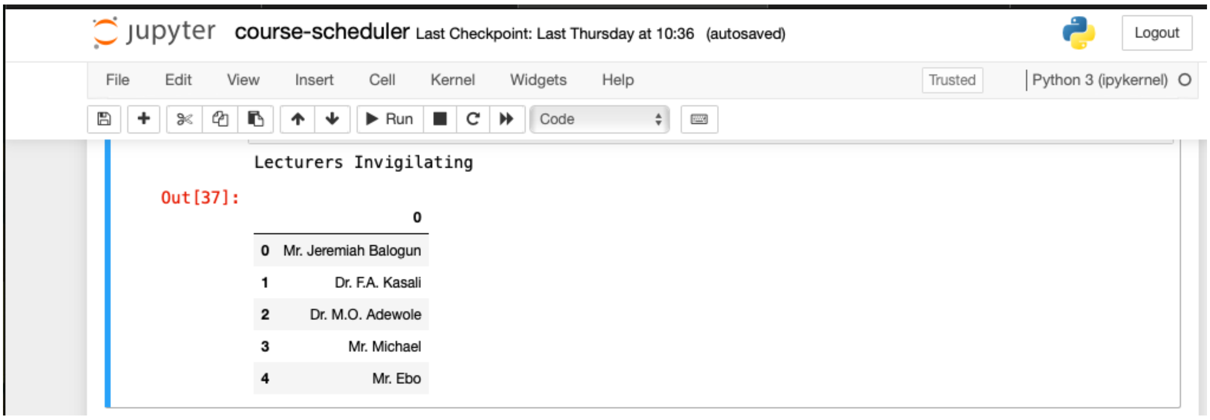


Figure 4.4.3 displays the first five invigilators

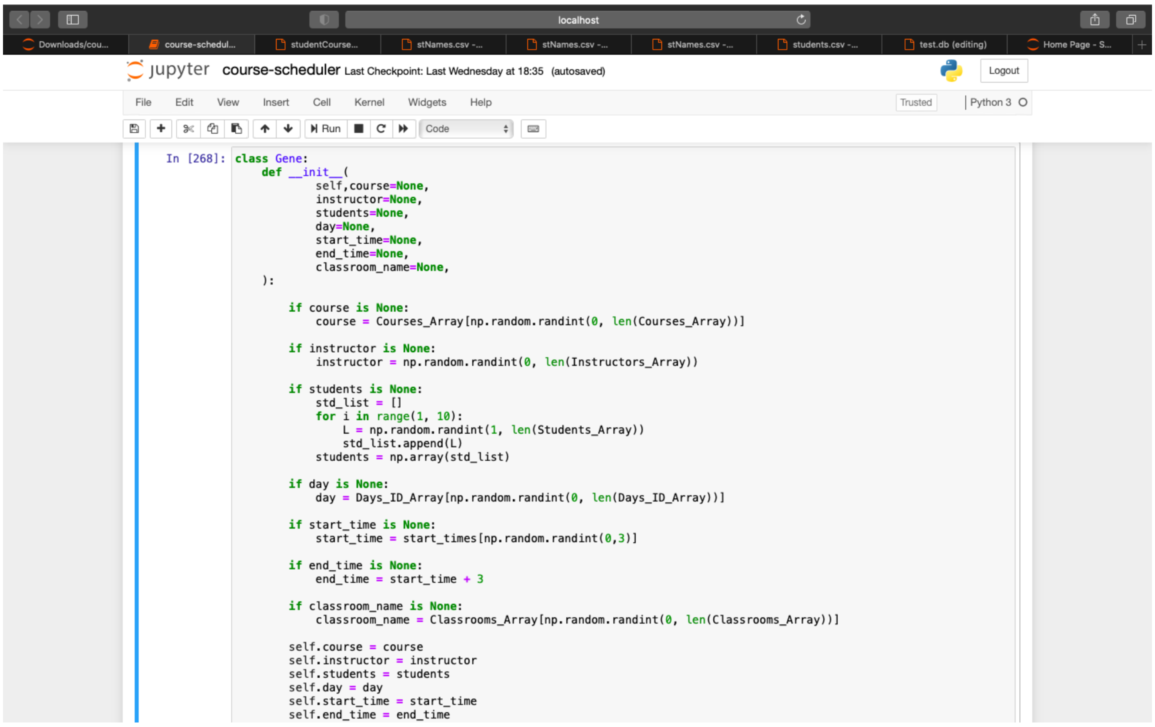


Figure 4.5.1(a) displays the representation of the genes.

Figure 4.5.1(b) shows the representation of the chromosome.

Based on the Darwin theory of natural evolution, the genetic algorithm produces an initial population, shown in Figure 4.5.2. The roulette wheel selection is employed in Figure 4.5.3(a) when parent chromosomes have been selected randomly. The function elitism is then specified by parent chromosomes in figure 4.5.3(b) The best chromosomes are represented. I utilized the randomized fixed point crossover technique shown in Figure 4.5.4 for crossover.

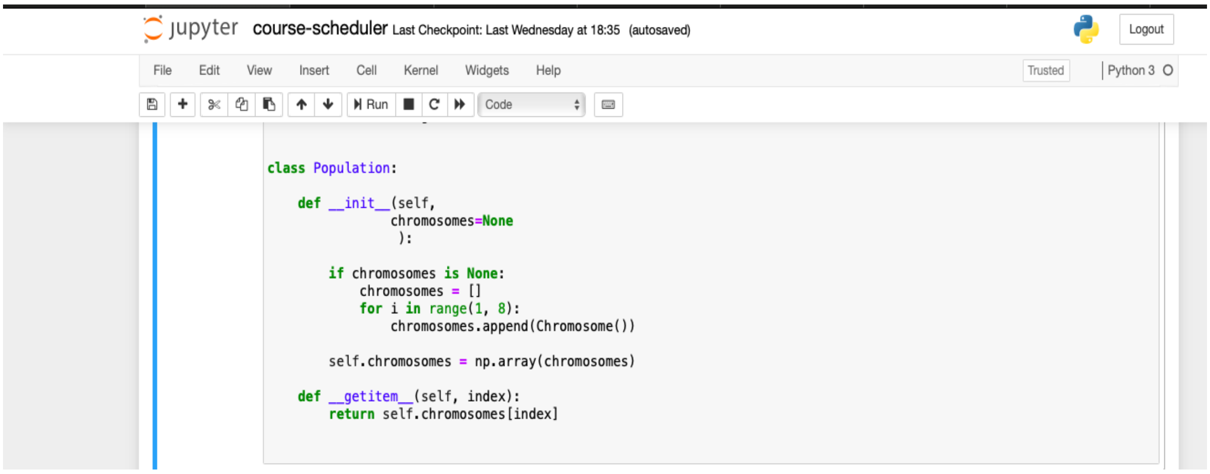


Figure 4.5.1(c) shows the representation of the population.

Initialization

In [309]:
```
def initialize_population():
    return Population()
```

Figure 4.5.2 displays the initialization of the first population

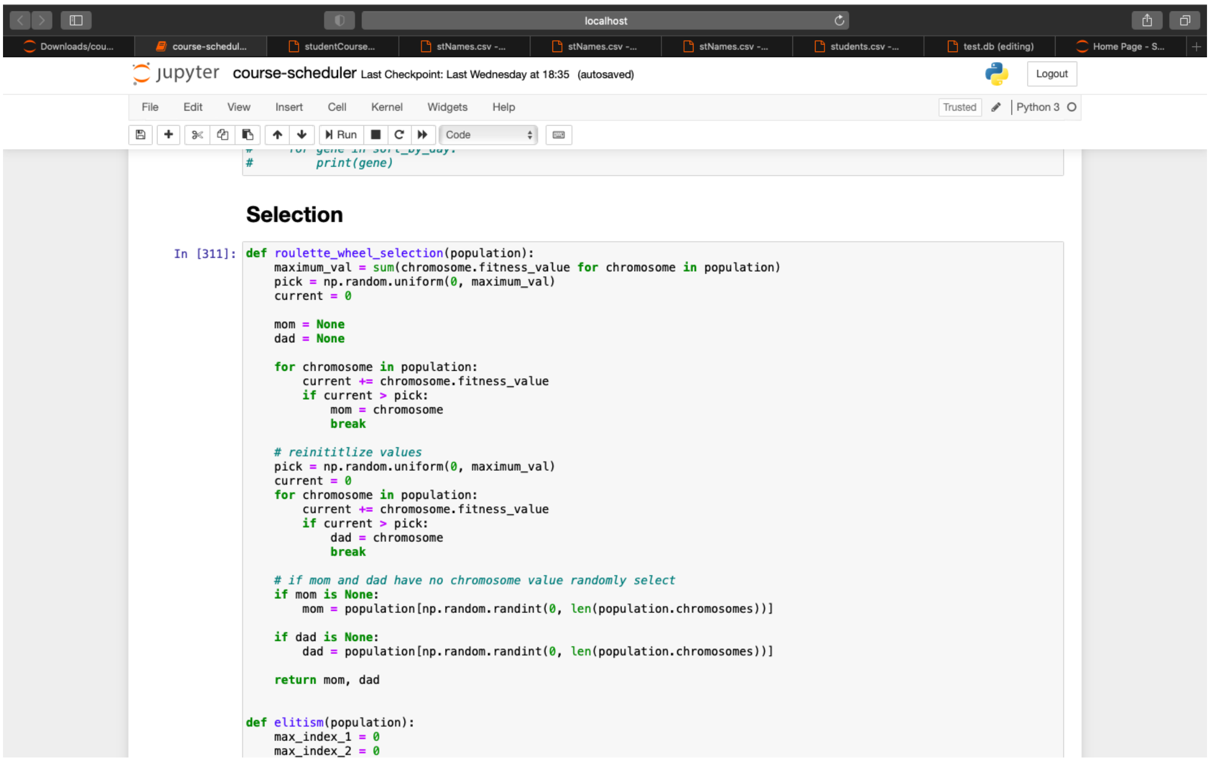


Figure 4.5.3(a) displays the roulette wheel selection method

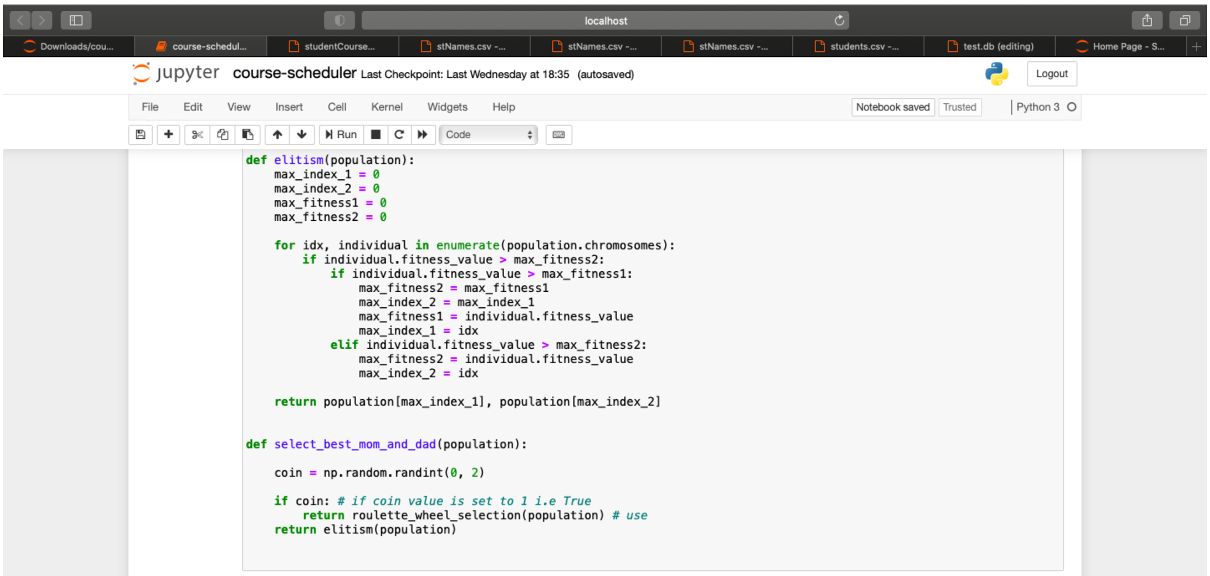


Figure 4.5.3(b) displays elitism

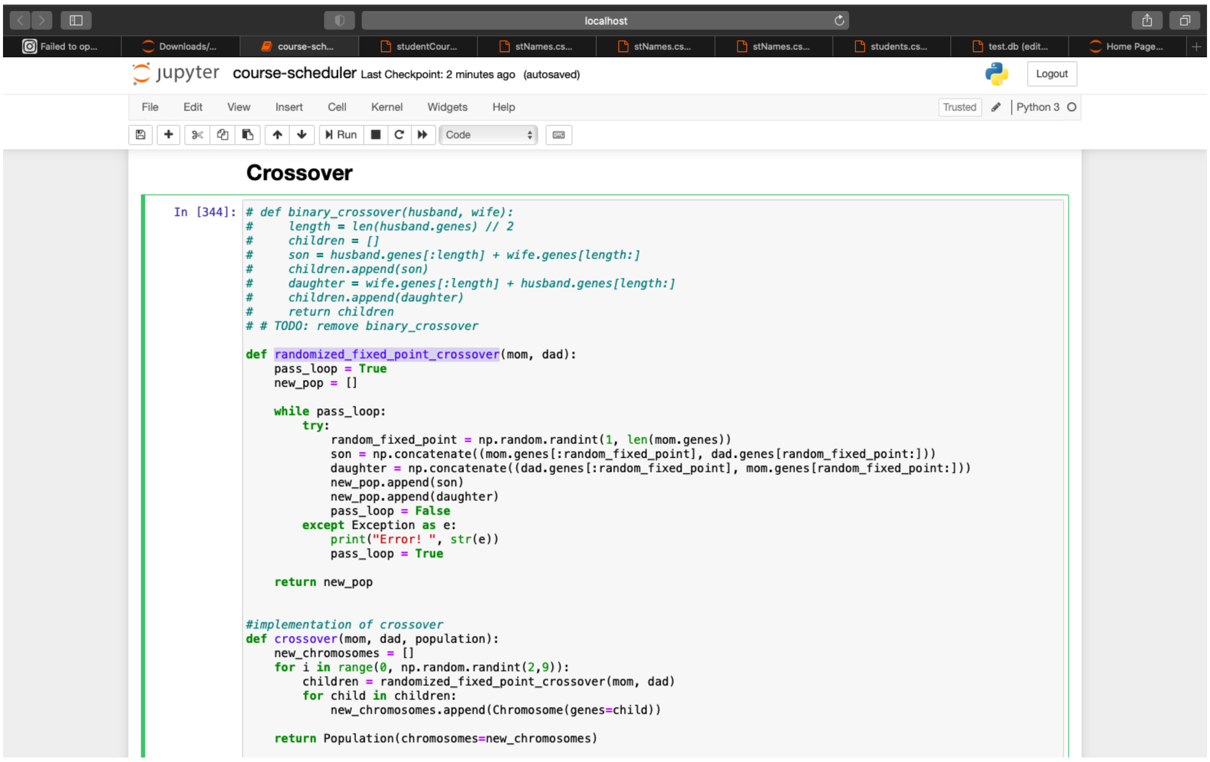


Figure 4.5.4 displays the randomized fixed point crossover technique.

Random probability is dependent on mutation. The chromosome level or gene level may be affected by mutations. It is either used heuristically at the gene level or a totally new gene is produced. At the level of a chromosome, a gene can be removed or an entirely new chromosome can be produced. Figure 4.5.5 displays start time, end time, instructor and day as the variables that underwent mutation.

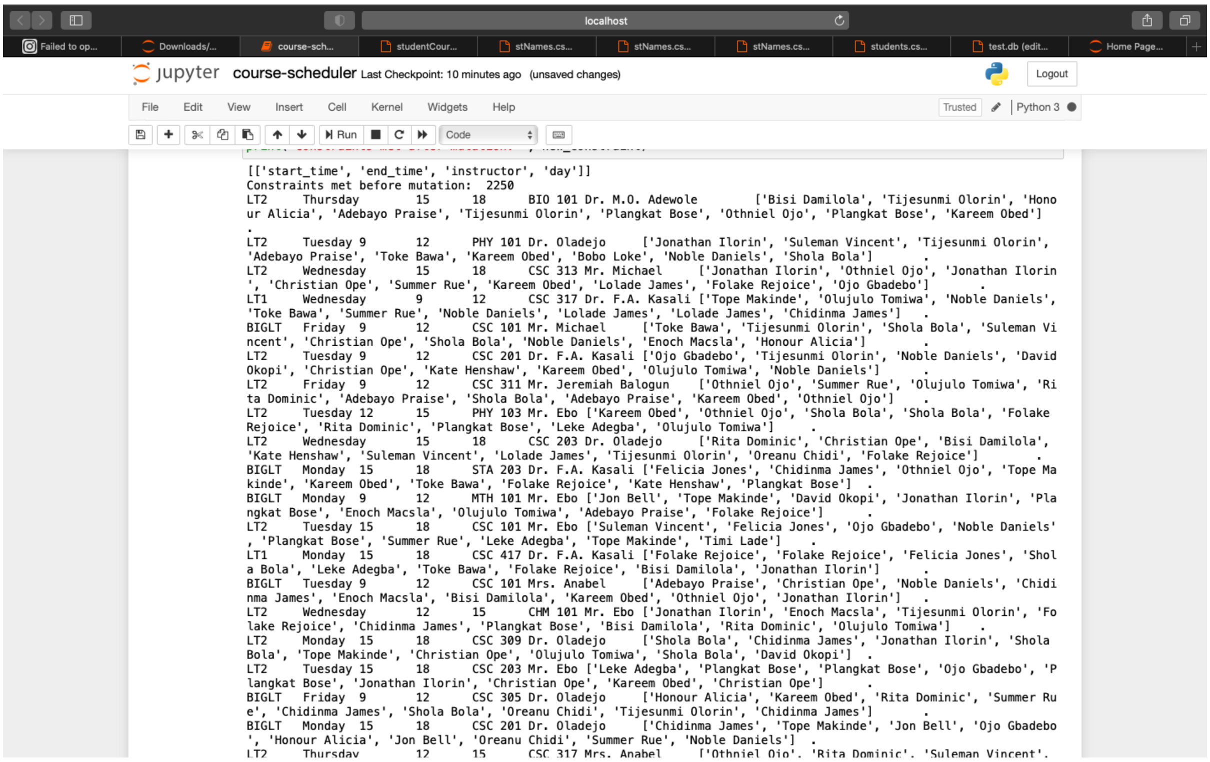


Figure 4.5.5 displays the mutated variables.

4.6 Fitness Calculation and heuristic function

To maximise the quality of the chromosome, I assigned points upon every constraint that was passed into the Genetic algorithm. For the hard constraints , the points assigned were fixed. However, for the soft constraints, I assigned variable points that was less than the hard constraints based on their priority. Whenever a constraint's passing points are less than a desired amount for a specific gene, I added only the relevant fields to a list. This list is provided to the mutation function, which uses it to only modify the fields which actually need to be modified.

4.7 Constraints

The constraints that were to be addressed like , students in the same class cannot take exams in different venues at the same time slot, no two courses taken by the same student group or registered as a carryover course can take place at the same venue and time slot and each course should be assigned to a timeslot were addressed.

4.8 Timetable

After generating this timetable 131 times, it was selected as that which best fits the addressed constraints. Although not generated in a table format, the timetable contained the venue, start and end time, the course, students and the invigilator. In figure 4.8.1(a), The fitness value for each parent is displayed for each generation of the timetable. The timetable is displayed in figure 4.8.1 (b,c,d,e) respectively.

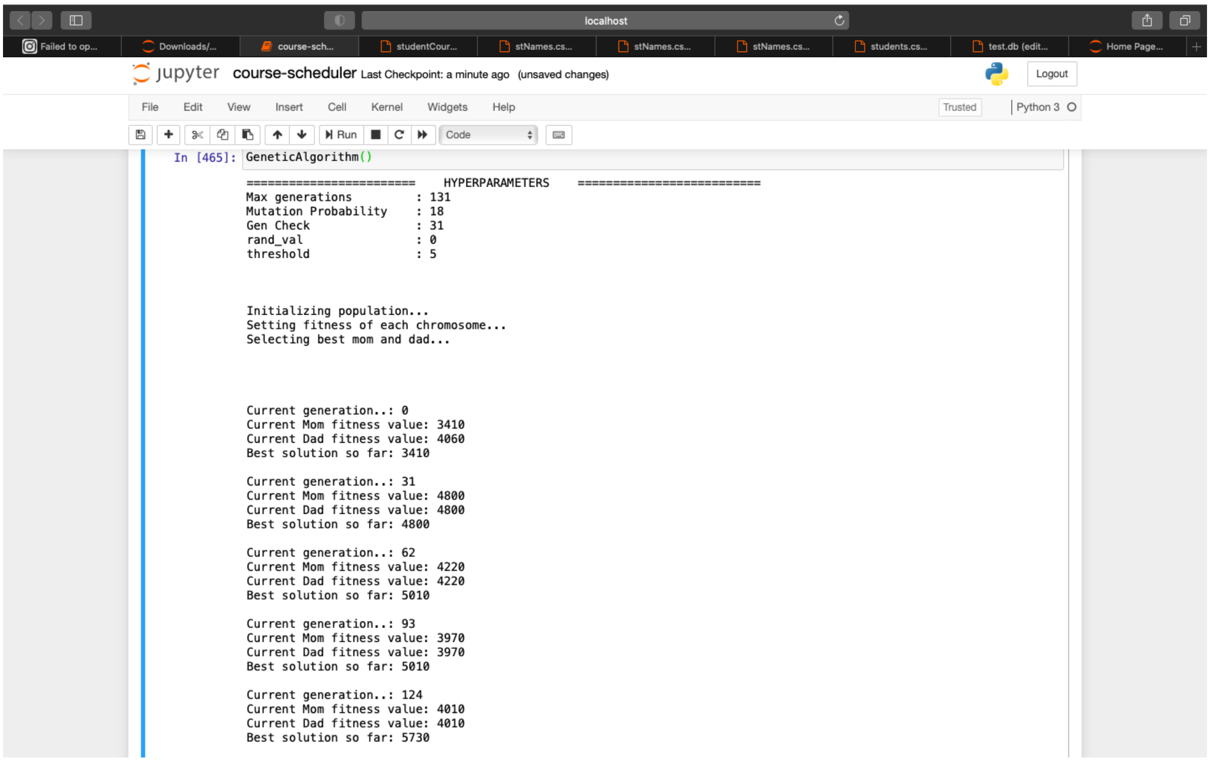


Figure 4.8.1(a) The fitness value for each parent is displayed.

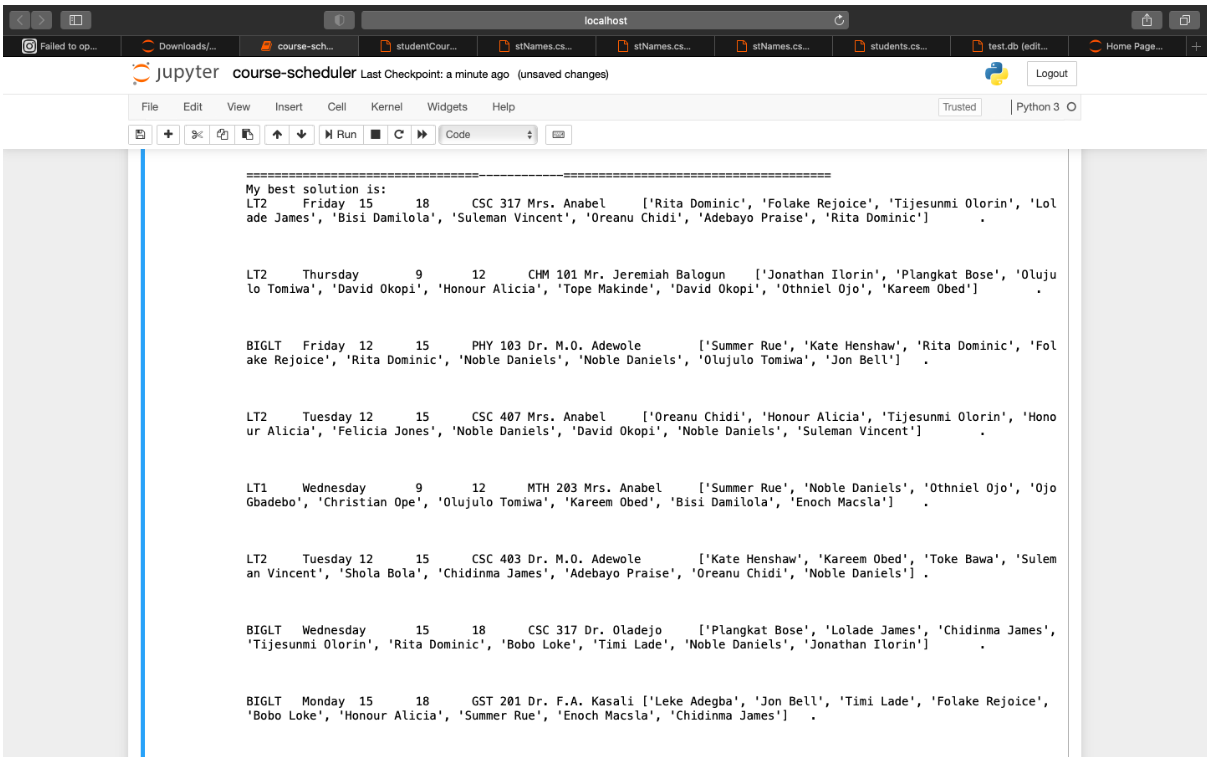


Figure 4.8.1 (b) displays the examination timetable.

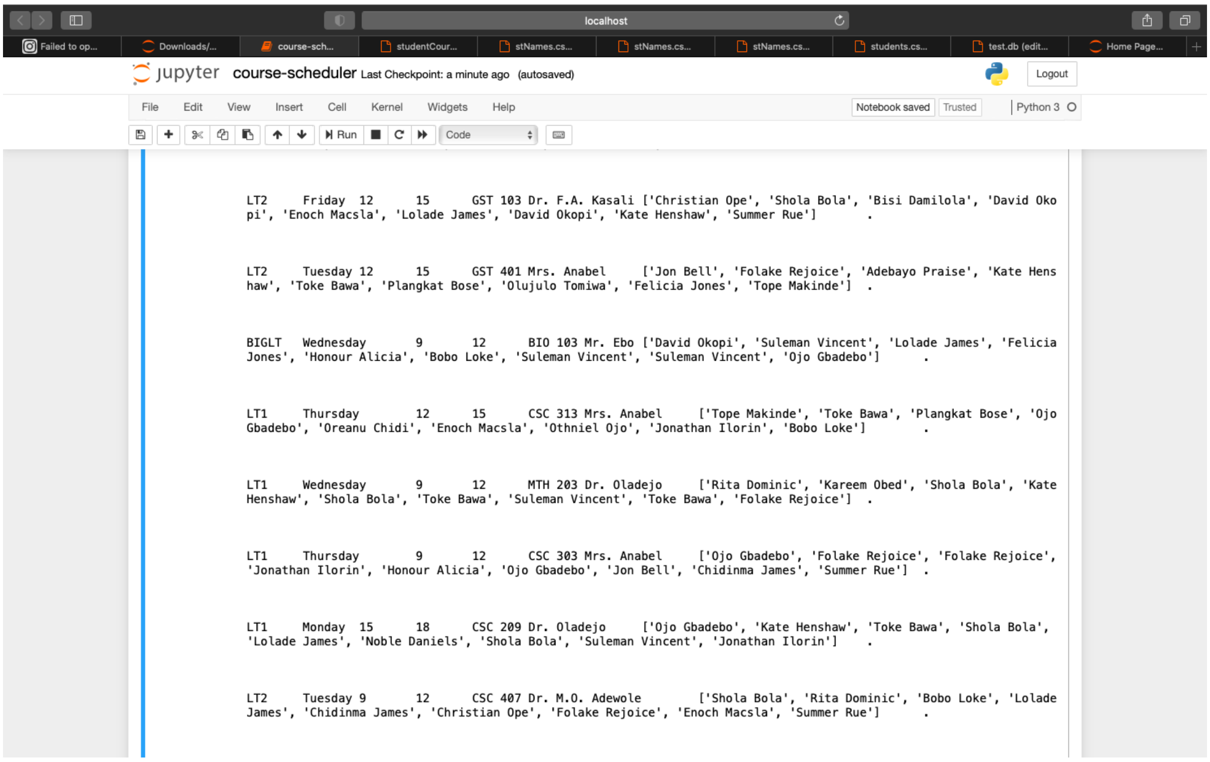


Figure 4.8.1(c) displays the examination timetable.

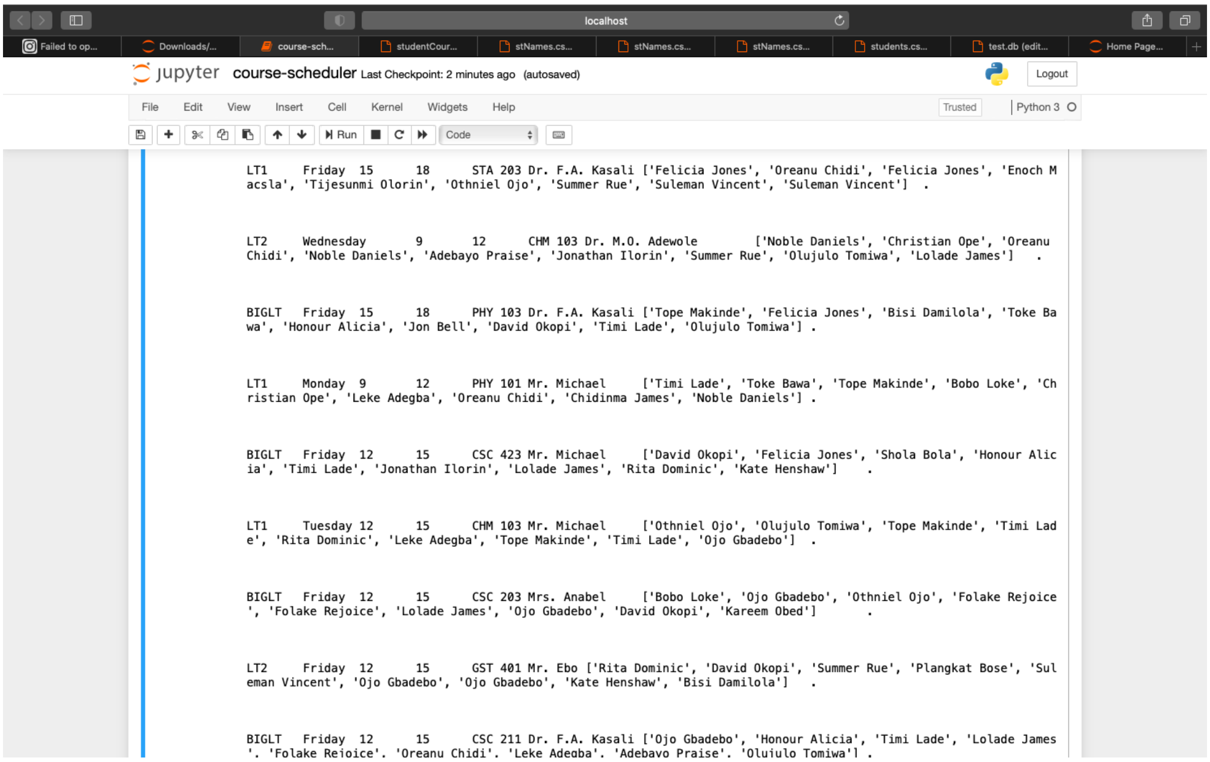


Figure 4.8.1 (d) displays the examination timetable.

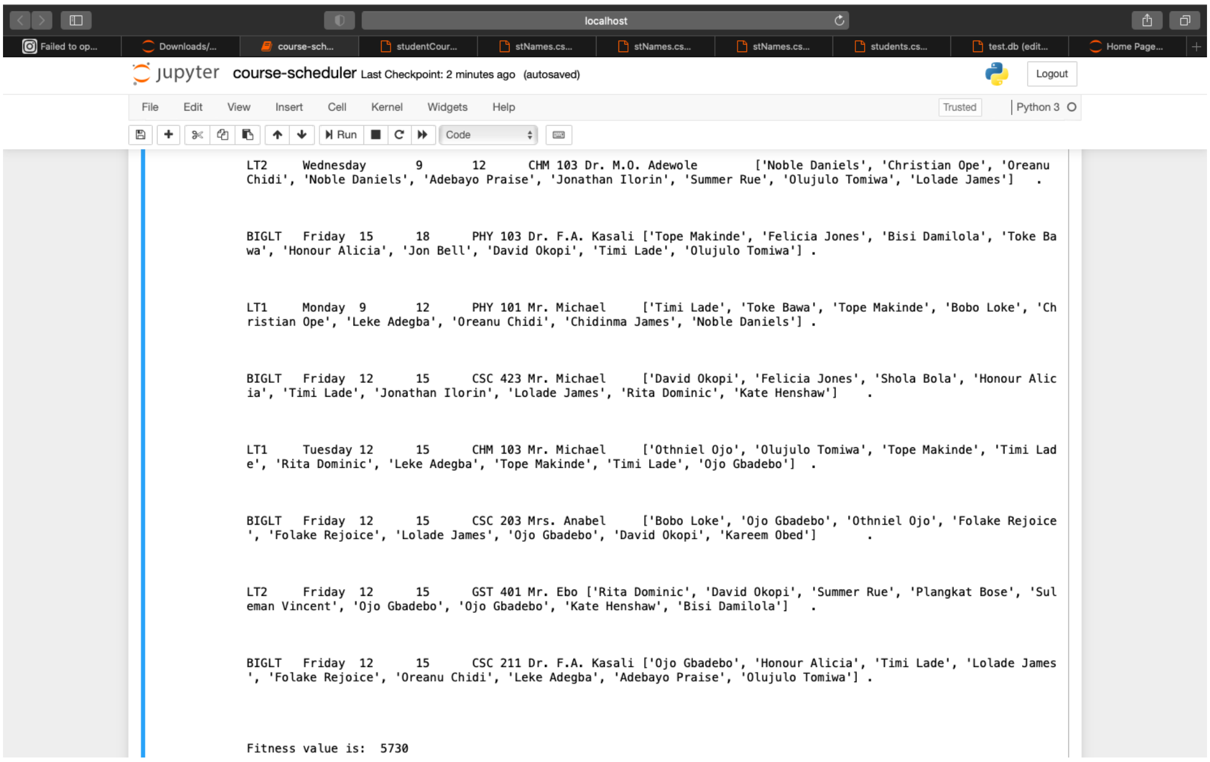


Figure 4.8.1 (e) displays the examination timetable.

CHAPTER FIVE

SUMMARY AND CONCLUSION

The system was developed to improve timetabling scheduling in general, the primary aim of applying genetic algorithm to the optimization of the timetable scheduling problem for the examination timetable. Examination timetables, especially if they are manually generated, are in effect a very demanding chore in any learning institution. The preparation can take days and weeks. This study suggested that the approach/technique to tackling the problem of genetic algorithms should be used. Although the experimental results show that a more efficient and dependable schedule can be reached with a properly constructed genetic algorithm. This offers good review schedules without conflicting examinations and in a much quicker time.

5.1 Limitations

One of the limitations of this project was that not all the hard constraints were addressed. I was able to populate each of the venues with students but due to the limited time, I was not able to declare the capacity of each of the venue and populate the venue based on that constraint. Due to the same reason, I could not fulfil the final period on Wednesday should not have any exams constraint.

5.2 Conclusion

The result has shown that the system is capable of providing useful solutions. It does not, however, fully automate the process. There are still some circumstances when the operator will need to make changes to some of the entries in order to achieve a flawless result. The enormous number of possible combinations for testing in order to arrive at an appropriate assessment for the application has proven to be impossible. However, considering the number of constraints set on the system, it can be inferred that the system was able to provide findings that, despite being imperfect, are valid and acceptable.

5.3 Recommendation

This study recommends that further work on this study can be focused on increasing the number of constraints and creating a neat table using the Hypertext Markup Language (HTML).

References

Ahmed F. AbouElhamayed, A. S. (2016). *An Enhanced Genetic Algorithm-Based Timetabling System with Incremental Changes.*

Awan-Ur-Rahman. (2020, April 26). *Introduction to ant colony optimization*. Retrieved from Towards data science: https://towardsdatascience.com/the-inspiration-of-an-ant-colony-optimization-f377568ea03f

Burke, E. A. (1996). Lecture Notes in Computer Science . *Practice and Theory of Automated Timetabling First International Conference.* Edinburgh, U.K.: Springer-Verlag Berlin Heidelberg.

Colorni, A. M. (1991). Genetic Algorithms and highly constrained problems:The time-table case,. *Parallel Problem Solving from Journal of Enhanced Research in Management & Nature, 496,* 55-59.

Christiansen, L. (2021, January 6). *The 6 Main Types of Information Systems*. Retrieved from Altametrics: https://altametrics.com/en/information-systems/information-system-types.html

Farah Adibah Adnan, S. A. (2018). Genetic Algorithm Method in Examination Timetabling Problem: A Survey. *Regional Conference on Science, Technology and Social Sciences* (pp. 901-907). Perlis, Malaysia: Springer Nature Singapore Pte Ltd.

Gagliordi, N. (2014, august 21). Retrieved from "US universities at greater risk for security breaches than retail and healthcare: BitSight | ZDNet".

H. M. Sani, M. M. (2016). Solving Timetabling problems using Genetic Algorithm Technique. *International Journal of Computer Applications*, volume 134.

Hamed Babaei, J. K. (2015). A survey of approaches for university course timetabling problem. In J. K. Hamed Babaei, *Computers & Industrial Engineering,* (pp. 43-59).

Hojjat Adeli, A. K. (2003). *Construction Scheduling, Cost Optimization and Management.*

Jackson, P. (1998). *Introduction To Expert Systems (3 ed.).* Addison Wesley.

Jessup, L. M., & Valacich, J. S. (2008). *Information Systems Today.* Pearson Publishing.

Khan Arman, K. S. (2015). Review of Generation of timetable using genetic algorithm implemented in java. 137-141.

Monmarché Nicolas, G. F. (2010). *Artificial ants.* Wiley ISTE.

O'Hara, M., Watson, R., & Cavan, B. (1999). *"Managing the three levels of change". Information Systems Management.*

Pham, D. A. (2015). A comparative study of the bees algorithm as a tool for function optimisation. *Cogent Engineering*, 2(1), 1091540.

Pham DT, G. A. (2005). The Bees Algorithm. *Technical note*.

Appendix

```
import pandas as pd

import numpy as np

import copy

print("libraries imported!")

 TO LOAD DATA

student_file = pd.read_csv('stNames.csv', header=None)

student_file.reset_index(drop=True, inplace=True)

course_file = pd.read_csv('courses.csv', header=None)

student_course_file = pd.read_csv('studentCourses.csv')

teacher_file = pd.read_csv('invigilator.csv', header=None)

Classrooms_Array = ["BIGLT", "LT1", "LT2"]

Courses_Array = list(course_file[0])

Instructors_Array = list(teacher_file[0])

Students_Array = list(student_file[0])

student_courses = []

for idx, student in enumerate(Students_Array):

    for index, row in student_course_file.iterrows():

        if row["Name"] == student:

            student_course_file.at[index, "Name"]=idx
```

```
Days_Array = [
    'Monday',
    'Tuesday',
    'Wednesday',
    'Thursday',
    'Friday',
]
num_of_days_in_week = 5
start_times = [9, 12, 15] # i.e 9am - 12pm, 12pm - 3pm, 3pm - 6pm in 24hrs format
Days_ID_Array = [0,1,2,3,4,]
def change_fundamentals():
    next_day = Days_Array[(Days_ID_Array[-1] + 1)%num_of_days_in_week]
    Days_ID_Array.append(Days_ID_Array[-1] + 1)
    Days_Array.append(next_day)
```

POPULATION MODEL

```
class Gene:
    def __init__(
            self,course=None,
            instructor=None,
```

```
        students=None,
        day=None,
        start_time=None,
        end_time=None,
        classroom_name=None,
):

    if course is None:
        course = Courses_Array[np.random.randint(0, len(Courses_Array))]

    if instructor is None:
        instructor = np.random.randint(0, len(Instructors_Array))

    if students is None:
        std_list = []
        for i in range(1, 10):
            L = np.random.randint(1, len(Students_Array))
            std_list.append(L)
        students = np.array(std_list)
```

```
if day is None:
    day = Days_ID_Array[np.random.randint(0, len(Days_ID_Array))]

if start_time is None:
    start_time = start_times[np.random.randint(0,3)]

if end_time is None:
    end_time = start_time + 3

if classroom_name is None:
    classroom_name = Classrooms_Array[np.random.randint(0, len(Classrooms_Array))]

self.course = course
self.instructor = instructor
self.students = students
self.day = day
self.start_time = start_time
self.end_time = end_time
self.classroom_name = classroom_name
```

```
    def __str__(self):
        student_names = []
        for student in self.students:
            student_names.append(Students_Array[int(student)])
        temp_day_name = Days_Array[self.day]


        instructor_name = Instructors_Array[self.instructor]


        return ("{}\t{}\t{}\t{}\t{}\t{}\t{}\t.".format(
            self.classroom_name,
            temp_day_name,
            self.start_time,
            self.end_time,
            self.course,
            instructor_name,
            student_names
            )
        )


class Chromosome:
```

```
def __init__(self,
            genes=None,
            fitness_value=None,
            ):

    if genes is None:
        genes = []
        for i in range(0, np.random.randint(10, 31)):
            genes.append(Gene())

    if fitness_value is None:
        fitness_value = 0

    self.genes = np.array(genes)

    self.fitness_value = fitness_value

def __getitem__(self, index):
    return self.genes[index]
```

```
class Population:

    def __init__(self,
                 chromosomes=None
                 ):

        if chromosomes is None:
            chromosomes = []
            for i in range(1, 8):
                chromosomes.append(Chromosome())

        self.chromosomes = np.array(chromosomes)

    def __getitem__(self, index):
        return self.chromosomes[index]
```

SELECTION

```
def roulette_wheel_selection(population):

    maximum_val = sum(chromosome.fitness_value for chromosome in population)

    pick = np.random.uniform(0, maximum_val)

    current = 0

    mom = None

    dad = None

    for chromosome in population:

        current += chromosome.fitness_value

        if current > pick:

            mom = chromosome

            break

    # reinititlize values

    pick = np.random.uniform(0, maximum_val)

    current = 0

    for chromosome in population:

        current += chromosome.fitness_value

        if current > pick:

            dad = chromosome

            break
```

```
    # if mom and dad have no chromosome value randomly select
    if mom is None:
        mom = population[np.random.randint(0, len(population.chromosomes))]
    if dad is None:
        dad = population[np.random.randint(0, len(population.chromosomes))]
    return mom, dad
def elitism(population):
    max_index_1 = 0
    max_index_2 = 0
    max_fitness1 = 0
    max_fitness2 = 0

    for idx, individual in enumerate(population.chromosomes):
        if individual.fitness_value > max_fitness2:
            if individual.fitness_value > max_fitness1:
                max_fitness2 = max_fitness1
                max_index_2 = max_index_1
                max_fitness1 = individual.fitness_value
                max_index_1 = idx
            elif individual.fitness_value > max_fitness2:
```

```
            max_fitness2 = individual.fitness_value
            max_index_2 = idx
    return population[max_index_1], population[max_index_2]
def select_best_mom_and_dad(population):
    coin = np.random.randint(0, 2)
    if coin: # if coin value is set to 1 i.e True
        return roulette_wheel_selection(population) # use
    return elitism(population)
```

CROSSOVER

```
def binary_crossover(husband, wife):
    length = len(husband.genes) // 2
    children = []
    son = husband.genes[:length] + wife.genes[length:]
    children.append(son)
    daughter = wife.genes[:length] + husband.genes[length:]
    children.append(daughter)
    return children
# TODO: remove binary_crossover
def randomized_fixed_point_crossover(mom, dad):
    pass_loop = True
```

```
    new_pop = []

    while pass_loop:

        try:

            random_fixed_point = np.random.randint(1, len(mom.genes))

            son = np.concatenate((mom.genes[:random_fixed_point],
dad.genes[random_fixed_point:]))

            daughter = np.concatenate((dad.genes[:random_fixed_point],
mom.genes[random_fixed_point:]))

            new_pop.append(son)

            new_pop.append(daughter)

            pass_loop = False

        except Exception as e:

            print("Error! ", str(e))

            pass_loop = True



    return new_pop

#implementation of crossover

def crossover(mom, dad, population):

    new_chromosomes = []

    for i in range(0, np.random.randint(2,9)):
```

```
        children = randomized_fixed_point_crossover(mom, dad)
        for child in children:
            new_chromosomes.append(Chromosome(genes=child))

    return Population(chromosomes=new_chromosomes)
```

CONSTRAINTS

```
def hard_test_invigilator(chromosome):
    wrong_value = 0
    wrong_genes = []

    points = 0
    for gene_1 in chromosome:
        for gene_2 in chromosome:
            if gene_2 == gene_1:
                continue
            if gene_2.day == gene_1.day:
                if abs(gene_2.start_time - gene_1.start_time) < 3:
                    if gene_2.classroom_name == gene_1.classroom_name:
                        if gene_2.instructor == gene_1.instructor:
```

```
                continue
            else:
                points += 10
    return points



def hard_test_valid_paper_duration(chromosome):
    points = 0
    for gene_p in chromosome:
        if (gene_p.end_time - gene_p.start_time) != 3 or gene_p.start_time == 14:
            continue
        else:
            points += 10

    return points



def hard_test_no_exam_on_weekends(chromosome):
    #     for chromosome in chromosomes:
    points = 0
```

```
    for gene_p in chromosome:
        if (gene_p.day in ["Saturday", "Sunday"]):
            continue
        else:
            points += 10
    return points




## Possible generic constraints
def hard_test_student_one_exam_at_a_time(chromosome):
    """At a given time, student can only give one exam"""
    # Note: Also test the current gene from which the student is selected, because the same student might be repeated in a single class
    points = 0
    i = j = 0

    while i < len(chromosome.genes):
        for student in chromosome.genes[i].students:
            for genes in chromosome:
```

```
            if genes == chromosome.genes[i]:
                continue
            if genes.start_time == chromosome.genes[i].start_time and genes.day == chromosome.genes[i].day:
                if student in genes.students:
                    continue
                else:
                    points += 10
        i += 1
    return points


def hard_test_one_exam_per_course(chromosome):
    """First hard constraint: Every course must have an exam"""
    courses_marked = []
    points = 0

    for genes in chromosome:
        if genes.course not in courses_marked:
            courses_marked.append(genes.course)
```

```
        points += 10

    return points



def hard_test_one_exam_per_classroom(chromosome):
    """At a given time, a classroom can only have one exam ongoing"""
    for genes_1 in chromosome:
        for genes_2 in chromosome:
            if genes_1 == genes_2:
                continue
            if genes_2.classroom_name == genes_1.classroom_name:
                if genes_1.start_time < genes_2.start_time or genes_2.start_time < genes_1.start_time:
                    return False
    return True


def hard_test_students_taking_correct_exam(chromosome):
    """Students must be taking the exam of the course they're registered for"""
    points = 0
```

```
    for genes in chromosome:
        for student in genes.students:
            mask = student_course_file["Name"] == student
            student_courses = student_course_file[mask]["CourseCode"]
            if genes.course in student_courses:
                points += 10
    return points
def soft_test_consecutive_exams(chromosome):
    points = 0
    for genes in chromosome:
        for student in genes.students:
            first_paper_time = genes.start_time
            for gene_1 in chromosome:   # for the rest of the genes in the chromosome
                if gene_1 == genes:
                    continue
                if student in gene_1.students:  # if the student is present in other gene
                    if gene_1.start_time != first_paper_time + 3: # if the gene start_time != first paper +3 hrs
                        points += 5
    return points
```

```
def soft_test_manna_water(chromosome):
    points = 0
    for genes in chromosome:
        if genes.day == 'Wednesday':
            if 10 <= genes.start_time and genes.start_time <= 15:
                continue
            else:
                points += 8
    return points
```

FITNESS OF CHROMOSOME

```
def set_fitness(chromosome_p):
    constraints_passed = 0
    fields_to_mutate = []
    # Right now, checking only hard constraints
    for constraint in HARD_CONSTRAINTS:
        constraints_passed += constraint["func"](chromosome_p)
        if constraints_passed < 1000:
            if constraint["fields"] not in fields_to_mutate: fields_to_mutate += constraint["fields"]
    # Set fitness value of individual chromosome
    chromosome_p.fitness_value = constraints_passed
```

```
    return constraints_passed, fields_to_mutate
```

MUTATION

```
def modify_gene(gene, fields):

    # Two different approaches:

    # 1. Only mutate those values which were found to violate the constraints e.g time was not
right so mutate time only

    #    -> will require a list of all constraints not met by the chromosome

    # 2. Mutate the entire gene completely



    # For now, let's try a combination of both.

    # Our model is partially self-aware.

    # So what we can do is that flip a coin.

    # If heads, then return a completely new gene.

    # If tails, apply some self-awareness.



    coin = np.random.randint(0, 2)



    # First make a completely random new Gene

    new_gene = Gene()
```

```
    # If heads...
    if coin:
        return new_gene

    # Else if tails...

    # Modify the existing fields of the selected gene
    else:
        for field in fields:
            setattr(gene, field, getattr(new_gene, field))
        return gene


def modify_chromosome(chromosome, random_index):
    # if heads, delete a gene
    # if tails, regenerate the entire chromosome
    coin = np.random.randint(0, 2)

    if coin:
        np.delete(chromosome.genes, random_index)
```

```
    else:
        chromosome = Chromosome()

    return chromosome


def mutate(chromosome, fields_to_mutate):
    random_index = np.random.randint(0, len(chromosome.genes))

    coin = np.random.randint(0, 2)

    # if heads, modify a gene
    if coin:
        chromosome.genes[random_index]  =  modify_gene(chromosome.genes[random_index],
fields_to_mutate)
    # if tails, modify the chromosome itself
    else:
        chromosome = modify_chromosome(chromosome, random_index)
    return chromosome
    new_pop = initialize_population()
```

```
for chromosome in new_pop:
    chromosome.fitness_value = np.random.randint(0, 21)
mom, dad = select_best_mom_and_dad(new_pop)
fields_to_mutate = []
fitness_value, temp_fields = set_fitness(mom)
if temp_fields not in fields_to_mutate:
    fields_to_mutate.append(temp_fields)
else:
    fields_to_mutate.append([])


print(fields_to_mutate)


random_index = np.random.randint(0, len(mom.genes))
previous_constraint, fields_to_mutate = set_fitness(mom)
print('Constraints met before mutation: ', previous_constraint)
for i in range(0, 200):
    mom = mutate(mom, fields_to_mutate)
    new_constraint, fields_to_mutate = set_fitness(mom)
for gene in mom:
    print(gene)
```

```
    print('Constraints met after mutation: ', new_constraint)
```

EVALUATION

```
min_soft_constraints = 3

def evaluate(candidate, useless):

    if useless is None:

        useless = copy.deepcopy(candidate)

    elif candidate.fitness_value > useless.fitness_value:

        useless = copy.deepcopy(candidate)

    return useless

def print_mom_and_dad(mom, dad):

    print("Mom genes:")

    for gene in mom:

        print(gene)

    print("Mom Fitness Value: ", mom.fitness_value)

    print("\n")

    print("Dad genes:")

    for gene in dad:

        print(gene)

    print("Dad Fitness Value: ", dad.fitness_value)
```

```
data = {}

monday_courses = []

tuesday_courses = []

wednesday_courses = []

thursday_courses = []

friday_courses = []

def GeneticAlgorithm():

    max_generations = np.random.randint(40, 201)

    mutation_probability = np.random.randint(0, 21)

    gen_check = np.random.randint(10, max_generations // 4)

    rand_val = np.random.randint(0, 6)

    threshold = np.random.randint(rand_val, gen_check // 2)



    print("=========================                    HYPERPARAMETERS
==========================")

    print("Max generations\t\t:", max_generations)

    print("Mutation Probability\t:", mutation_probability)

    print("Gen Check\t\t:", gen_check)

    print("rand_val\t\t:", rand_val)

    print("threshold\t\t:", threshold)
```

```
print("\n\n")

print("Initializing population...")
new_generation = initialize_population()

solution_counter = 0
prev_best_solution = None
best_solution = None

print("Setting fitness of each chromosome...")
for chromosome in new_generation:
    set_fitness(chromosome)

print("Selecting best mom and dad...")
mom, dad = select_best_mom_and_dad(new_generation)

print("\n\n")

for i in range(0, max_generations):
    children = crossover(mom, dad, new_generation)
```

```
new_chromosomes = []

fields_to_mutate = []

for idx, chromosome in enumerate(children):

    fitness_value, temp_fields = set_fitness(chromosome)

    fields_to_mutate.append(temp_fields)



    if np.random.randint(0, 101) < mutation_probability:

        mutate(chromosome, fields_to_mutate[idx])

        fitness_value_changed, fields_to_mutate[idx] = set_fitness(chromosome)

        fitness_value = fitness_value_changed



    new_chromosomes.append(chromosome)



next_gen = Population(chromosomes=new_chromosomes)



mom, dad = select_best_mom_and_dad(next_gen)



best_solution = evaluate(mom, best_solution)



### print Every 100th generation results
```

```
if i % gen_check == 0:

    print('\nCurrent generation..: {}'.format(i))
    print('Current Mom fitness value:', mom.fitness_value)
    print('Current Dad fitness value:', dad.fitness_value)
    print('Best solution so far: {}'.format(best_solution.fitness_value))
    if prev_best_solution is None:
        prev_best_solution = best_solution
    if best_solution.fitness_value > prev_best_solution.fitness_value:
        solution_counter = 0
    elif best_solution.fitness_value == prev_best_solution.fitness_value:
        solution_counter += 1

    if solution_counter > threshold:
        # change_fundamentals()

        new_generation = initialize_population()
        np.append(new_generation.chromosomes, mom)
        np.append(new_generation.chromosomes, prev_best_solution)
```

```
            solution_counter = 0


        elif solution_counter > rand_val:
            dad = best_solution
        else:
            prev_best_solution = best_solution
    # -------------------------------- END GA LOOP
    print("\n\n")
    print("\n\n==================================------------
=======================================")


    print("Our best solution is:")
    for gene in best_solution:
        print(gene)
        if Days_Array[gene.day] == "Monday":
            monday_courses.append(gene.course)
        elif Days_Array[gene.day] == "Tuesday":
            tuesday_courses.append(gene.course)
        elif Days_Array[gene.day] == "Wednesday":
            wednesday_courses.append(gene.course)
```

```
    elif Days_Array[gene.day] == "Thursday":

        thursday_courses.append(gene.course)

    else:

        friday_courses.append(gene.course)

    print("\n\n")

print("\nFitness value is: ", best_solution.fitness_value)
```