Time Table Generation using Genetic Algorithms

Design Capstone Report

by

Ayush Mishra (UCSE20054)
Preetibarna Panda (UCSE20038)
Ashutosh Pradhan (UCSE20058)

Under the supervision of

Prof. Rudra Mohan Tripathy

DESIGN CAPSTONE I: CSE 692

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

XIM UNIVERSITY

March, 2023

Contents

1 Abstract **1**

2 Introduction **1**

3 System Analysis **2**

4 System Design **2**

5 System Implementation **2**

5.1 Implementation 4

5.2 Encoding 4

5.3 Population Initialization 4

5.4 Fitness Evaluation 5

5.5 Selection 6

5.6 Crossover 8

5.7 Crossover Algorithms Implemented 9

5.8 Repair Function 10

5.9 Mutation 10

5.10 Mutation Algorithms Implemented 11

6 Acknowledgement **12**

7 Appendix **13**

1 Abstract

A customized heuristic solution to the famous Time-Table problem involving the concepts of Combinatorial Optimization [1] using Genetic Algorithms and Fuzzy Logic methodologies in the form of an application. This would be made with the intention of actually reducing the time taken by our school for the timetable-making process. We will also try to improve the implementation by using Parallel processing paradigms.

2 Introduction

- **Project Objectives**
  The following are the objectives of this project:-
  - 1. Minimizing the conflicts between constraints in Time-Table scheduling.
  - 2. Automating and Optimizing the Time-Table generation processes customized for the School of Computer Science and engineering, XIM University with the purpose of reducing the time taken in the manual process.
  - 3. Hard Constraints or any other critical constraints should not be violated and soft constraints should be applied as much as possible.
  - 4. Creating a usable GUI to provide input and view the time-tables.

- **Background of the Project**
  Although most of the Educational Institution's administrative work has been computerized, [2] due to human errors or difficulties involved, some work is still done manually in these institutions.
  A Time-Table consists of assigning certain resources(Students, Faculty, Courses, etc.) to objects(Classrooms, laboratory, etc.) placed in time and space so that they fulfill a set of objectives.
  Time-Table problem is a Combinatorial Optimization problem, in which the time increases exponentially with the increase in the number of variables.

- **Operation Environment**
  Educational Institutions like Schools, Colleges, Universities, etc. will be the Operation Environment for Time-Table generation project.

3 System Analysis

- **Software Tools used**
  The following Software tools are currently in used-
  - 1. Python Language
  - 2. At the end of this project, the project will be deployed using micro web framework **Flask**.

4 System Design

The application will consist of 2 interfaces. The first interface is for taking 2 .csv files as input. The input will be extracted and the details will be structured using classes and objects.
The .csv files taken as input will be ”faculty.csv” which contains data regarding faculties, and ”course.csv” which contains data regarding all courses of that semester. One class will be ”Faculty” which contains the faculty name, faculty id, a custom faculty id, and Faculty type (either visiting faculty or not). Another class will be ”Course” which contains Course id, Course name, Faculty concerned, Course type (lab, workshop, lecture), Elective, etc.
The functions involved in the genetic algorithm are initial population, fitness, crossover, mutation, etc. There are some utility functions included in the initialization file. After executing the genetic algorithm. The final Timetable would be displayed in another interface. There can also be an option to generate a different timetable.

5 System Implementation

1. Problem Formulation

- **Input :- Courses.csv** For every batch the ”Courses.csv” file should contain -
  - **Course ID -** The respective ID of the courses.
  - **Course Name -** The respective name of the course
  - **Course Code -** The respective code of the course
  - **Semester -** The Semester to which the course is taught to
  - **Faculty ID -** The respective ID’s given to the faculties.
  - **NOCW -** Number of credit hours of the course
  - **Type -** Lab or Lectures.
    - ∗ Workshops are considered as lectures as they are conducted in classrooms itself.

- **Input :- Faculty.csv** For every faculty the ”Faculty.csv” file should contain -
  - **ID -** The respective ID of the faculty.
  - **Faculty Name -** The respective name of the Faculty
  - **Visiting -** If the corresponding Faculty is a visiting faculty or not

  The input would be taken with the help of a form format.

- **Output :** Four tables, One for each batch, satisfying some predefined constraints in the form of a structured timetable.

- **Constraints :** The constraints for the timetable problem are as follows.
  - A time slot cannot have more than one course of the same batch.
  - Courses taught by a single faculty cannot exist simultaneously in the same time slot for different batches as a faculty member cannot take 2 courses at the same time.
  - Lab classes should not clash within batches.
  - If one course has 2 time-slots in a day then the slots have to be consecutive.
  - Number of occupied time slots in a day cannot be more than 5.
  - Lab classes are held in the second half.
  - Consecutive days cannot have the same courses.

  The constraints are ordered in the decreasing order of priority

2. Problem Description
Before going to the actual problem, it is better to understand what and how genetic algorithm works.

- **What are Genetic Algorithms?**
  Genetic algorithms are meta-heuristic methods used to solve computational problems which require large search areas for possible solutions. In Time-Table scheduling, it is a self-adaptive method that is desired to increase the level of generality.

- **How Genetic Algorithm Work?**
  In a genetic algorithm, a population of chromosomes consisting of a given random collection of genes is initiated according to following steps:-
  - 1. Initial random population generation
  - 2. Evaluating the fitness function
  - 3. Selection
  - 4. Crossover
  - 5. Mutation

5.1 Implementation

Every Genetic Algorithm starts with some populace of chromosomes which are generated [3] using user defined methodologies. In most cases we generated the chromosomes by randomly placing the genes wherever possible. The population then subjected to continuous generational evolution changes to a populace with much more fit individuals than initial ones.

5.2 Encoding

In our Time-Table Generation problem we initialize the skeleton of chromosome as a list containing the smaller lists for each day :

$$[Monday, Tuesday, Wednesday, Thursday, Friday]$$

Each day consists of 6 different slots from 9:30AM to 5PM , for example :

$$Monday = [Slot1, Slot2, Slot3, Slot4, Slot5, Slot6]$$

Each slot is defined as a list of 4 string values , representing classes of 4 different batches :

$$Slot1 = ['','','','']$$

Index 0 : First Year Index 1 : Second Year Index 2 : Third Year Index 3 : Fourth Year Thus this structure creates a 3-Dimensional chromosome of total gene size 120.

$$Numberofgenes = NumberofClasses * Numberofslots * 4$$

$$Number.ofgenes = 5 * 6 * 4 = 120$$

5.3 Population Initialization

We extract all useful data such as Courses , their corresponding credit hours , to which batch a particular course is associated with , teacher - course correspondence and many other elements ; from the two CSV files provided by the user.

Once we have a chromosome's skeleton ready , using the extracted data elements we can create an initial chromosome. For this we randomly choose a day , a slot in that day and a batch of that slot. Then for the corresponding batch , we choose a random subject that is taught to the corresponding batch. The subject is placed in the selected gene place.

$$...]['EMII','','','PDC'][...$$

Let's consider a snippet of chromosome as shown above. The red colored gene is the randomly selected gene to be filled next. As this index corresponds to 3rd year , we will randomly choose any valid subject and put it there.

$$...][{'}EMII{'},{'}{'},{'}AI{'},{'}PDC{'}][...$$

We continue to do the same until every subject is placed anywhere in the chromosome , fulfilling it's credit hours as well.

5.4 Fitness Evaluation

Evaluation is the process of calculating the fitness of a chromosome. Fitness is the measure of the quality of the chromosome relative to the desired solution. Our Time - Table generation problem involves a lot of constraints that should be full-filled in order to create a useful and good Time-table. In order to satisfy those constraints we evaluate the chromosome on many factors.

A conflict is an occurrence of unwanted positioning of class or classes, which does not satisfy our constraints. for eg.

- **A time slot cannot have more than one course of the same batch.**
- **Courses taught by a single faculty cannot exist simultaneously in the same time slot for different batches as a faculty member cannot take 2 courses at the same time**
- **No two batches should have the same lab allotted to them in the same slot of time**

We check these constraints by iterating through the chromosome and for each combination of two classes in the same slot, if they both are of the same faculty, we consider it as a conflict.

For the same, if two such subjects exist of different batches that are lab classes, and are allotted the same lab, we consider it as a conflict.

- **Lab classes are held in the second half.**

In this, if we ever find any lab class positioned in the second half slot of the day, we consider it as a conflict.

- **If one course has 2-time slots in a day then the slots have to be consecutive.**

We check the whole chromosome of the week for any discontinuity of classes in a particular batch. Any discontinuity found results in a conflict.

- **Class after lunch and before lunch should not be the same.**

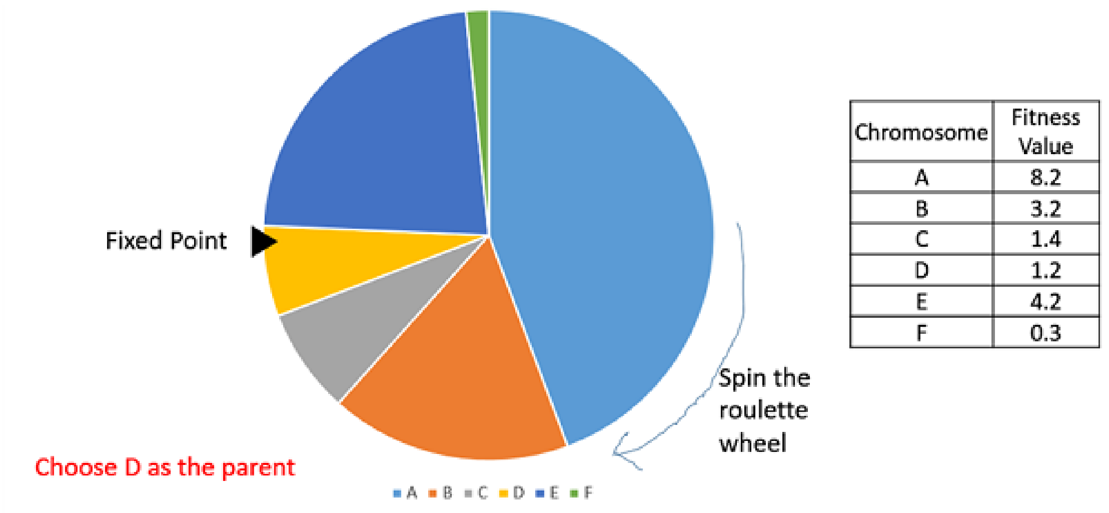
For any particular batch, if the classes are positioned in slots just before the lunch break at 12:30 PM and slots just after lunch; are the same, will result in a conflict.

- **Try to have at-least one 2 hours continuous class of a subject with credit greater than or equal to 3.**
- **Subjects held today should not be held tomorrow.**
- **There should be only one , two hour class in a day for every batch**
- **Instructor should get a break after a 2 hours consecutive classes.**

5.5 Selection

Selection can be defined as a stage of a genetic algorithm in which individual genomes are chosen from a population for later bleeding. It is very crucial to the convergence rate of the GA as good parents drive individuals to better and fitter solutions [4] . Maintaining good diversity in the population is very crucial for the success of a GA.
A condition where the entire population is taken over by an extremely fit solution is called **Premature Convergence**. It is a very undesirable condition in genetic algorithms.
There are many different types of Selection used in Genetic Algorithms-

- **1. Fitness proportion Selection**
  a) Roulette Wheel Selection
  In the roulette wheel section, the circular wheel is used. The region of the wheel which comes in front of a fixed point is chosen as a parent. For the second parent, the same process is repeated. A fitter individual will have a greater pie on the wheel and therefore there's a greater chance of landing in front of the fixed point when the wheel is rotated.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Figure 1: Roulette Wheel Selection

Source: Google Images

- **2. Tournament Selection**
  Tournament Selection selects individuals for the mating pool depending on their fitness function. It is a frequently used method in GA due to its simplicity and efficiency. This method involves a random selection of individuals from a bigger population.

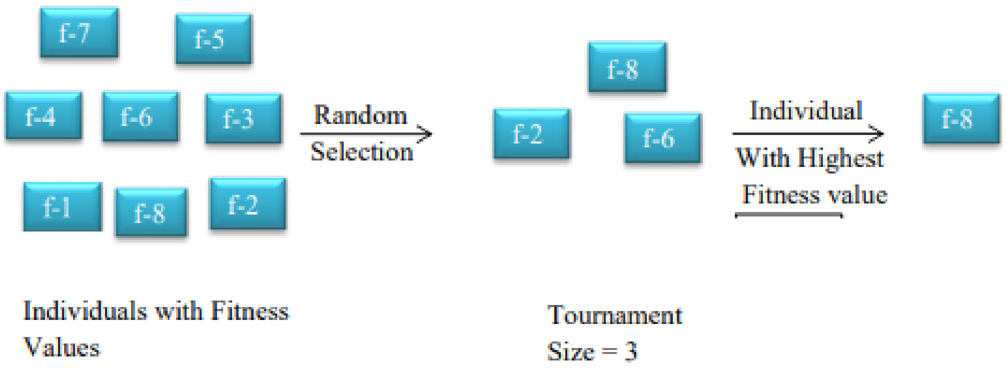
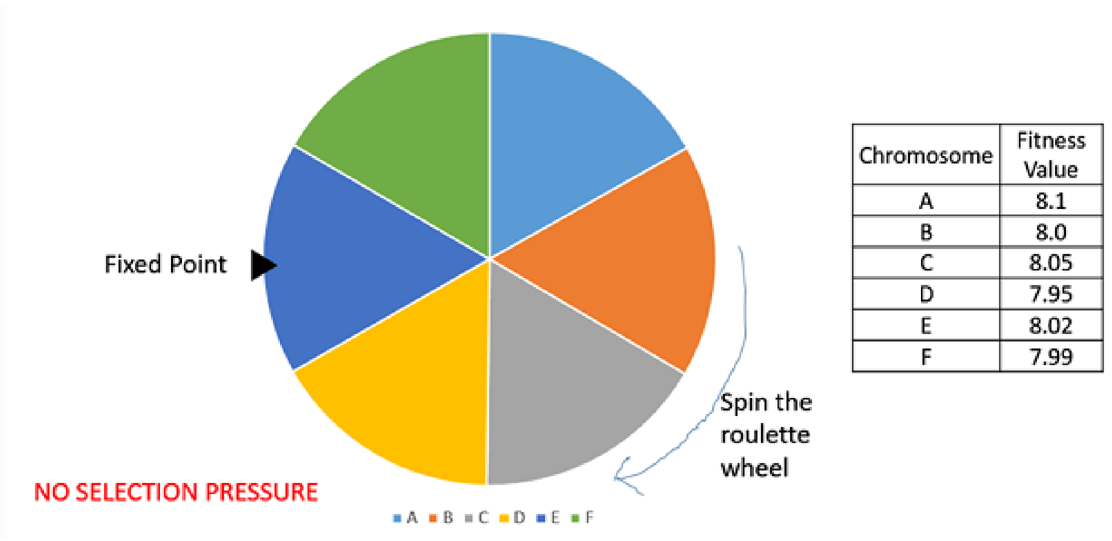


Figure 2: Tournament Selection

Source: Google Images

- **3. Rank Selection**
  Rank Selection works with the negative fitness values and is mostly used when the individuals in the population have very close fitness values. This leads to each individual having an almost equal share of the wheel or pie chart. No matter how fit the individuals are relative to each other, every individual has the approximate same probability of getting selected as a parent.



Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

Figure 3: Rank Selection

Source: Google Images

– **4. Random Selection**
Random Selection selects parents randomly from the existing population. There is no selection pressure toward fitter individuals and therefore this strategy is not very often used.

5.6 Crossover

The crossover is analogous to reproduction and biological crossover. In this, one parent is selected [5] and one or more off-springs are produced using the genetic material of the parents. Crossover is usually applied in a GA with a high probability – Pc .
There are three types of Crossovers Operators in Genetic Algorithms -

– **1. One Point Crossover**
In One-Point Crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.

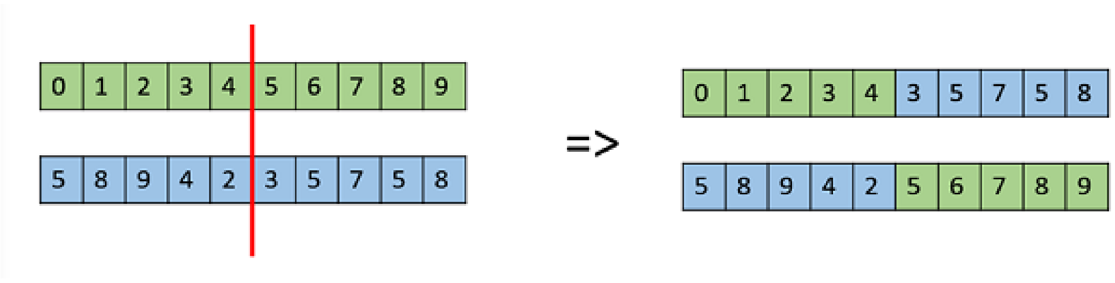


Figure 4: One Point Crossover

Source: Google Images

– **2. Multi Point Crossover**
Multi-point crossover is a generalization of the one-point crossover where alternating segments are swapped to get new off-springs.

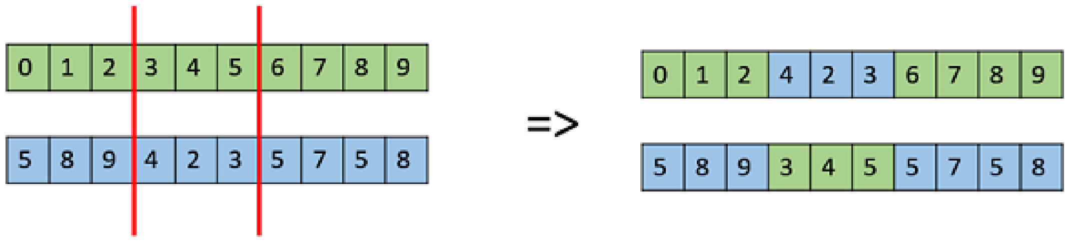


Figure 5: Multi Point Crossover

Source: Google Images

– **3. Uniform Crossover**
In Uniform Crossover, chromosomes are not divided into segments, rather we treat each individual gene separately. In this crossover, we decide

whether each chromosome will be or will not be included in the off-springs or not.

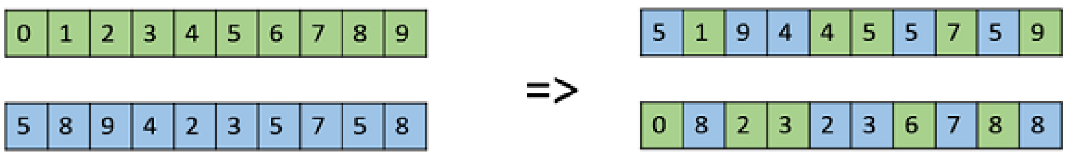


Figure 6: Uniform Crossover

Source: Google Images

5.7 Crossover Algorithms Implemented

– 1. Index - Wise Crossover

We take two parents as inputs and using the function 'weektosubs', we convert each of the previously encoded multidimensional chromosomes of parents to a 1-D list. This new list consists of all the subject placements opened day-wise and then slot-wise. Following this, the parents are crossed over using Multi-point Crossover. As the crossover will eventually destroy some of the genes, we will have to repair the lost elements before passing them to a new generation.

After opening the chromosome , let's consider there are two points of crossover denoted by red genes

$$Parent1 = ...AI, PDC,'', DBMSLab, SNA, PDC, EMII,''...$$

$$Parent2 = ...SC,'', OOPJ, WT, SC,'','', DM...$$

Then a point-oriented crossover will create two different offsprings :

$$Offspring1 = ...AI, PDC,'', DBMSLab, SC,'','', DM...$$

$$Offspring2 = ...SC,'', OOPJ, WT, SNA, PDC, EMII,''...$$

- **2. Slot - Wise Crossover**

  We take two parents as inputs and using the function 'weektoslots', we convert each of the previously encoded multidimensional chromosomes [6] of parents to a 2-D list. This new list consists of all the subject placements opened slot-wise. Following this, the parents are crossed over using Multi-point Crossover. As the crossover will eventually destroy some of the genes, we will have to repair the lost elements before passing them to a new generation.

- **3. Uniform Crossover**
  We take two parents as inputs and using the function 'weektosubs', we convert each of the previously encoded multidimensional chromosomes of parents to a 1-D list. This new list consists of all the subject placements opened day-wise and then slot-wise. Following this, the parents are crossed over using a uniform-crossover algorithm; which states that each gene of corresponding parents gets a 50 percent chance to be part of either of the offspring.

5.8 Repair Function

The repair function takes account of the following things :

- 1. Some genetic values are destroyed during crossover
- 2. Genetic material lost should be derived from the parents
- 3. Still missing genetic materials can be brought back according to our likeness

Using these postulates we repair any chromosome by creating a note of every missing class from it. Firstly we repair it according to the parent's gene present. That is; if a parent's gene value exists at the position **i**, whereas the offspring chromosome misses that value, we simply put that value of the parent back to the offspring chromosome at position **i**.

Finally, we selectively repair chromosomes such that if we find any occurrence of class that is also missing part from the chromosome. We check for the possibility of putting another class consecutive to it, if possible so we put the consecutive classes together. Otherwise, we continue to the section where we randomly put the classes missing to any random black space we encounter or find.

5.9 Mutation

Mutation in Genetic Algorithms can be defined as a small random twist in the chromosome, which gives a new solution. The mutation is part of the GA which is related to the "exploration" of the search space [7] . It is used

to maintain and introduce diversity in the genetic population and is usually applied with a low probability - **Pm**. If the probability is very high, the GA gets reduced to a random search.
The following are the Mutation Operators used in Genetic Algorithms-

- **1. Swap Mutation**
  In Swap Mutation, we select two positions on the chromosome at random and interchange the values. This is common in permutation-based encoding.

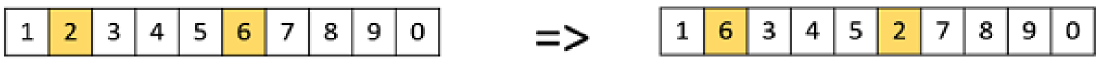


Figure 7: Swap Mutation

Source: Google Images

- **2. Scramble Mutation**
  Scramble Mutation is used for permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.

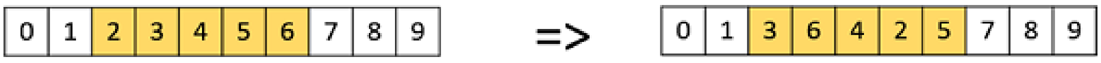


Figure 8: Scramble Mutation

Source: Google Images

- **4. Inversion Mutation**
  In Inversion Mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.

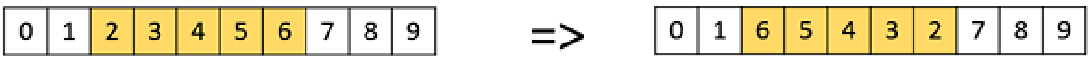


Figure 9: Inversion Mutation

Source: Google Images

5.10 Mutation Algorithms Implemented

Mutation in a timetable problem helps in various aspects. One of which is load distribution over a whole week [8] , as it can randomly swap a very packed schedule to a sparsely packed one. Thus creating a sense of balance between the class load. Also it helps in creating and utilizing unique solutions. We implement different types of mutation algorithms. Those are :

* **1. Slot-Wise mutation** A chromosome is opened slot-wise using the function 'weektoslots'. Then we randomly choose two slot elements that are further swapped with each other.
  Let's consider a case of 3 slots where the slots colored in red and blue are chosen to be swapped.

  $$['MB','DBMS','SC','DL'],['EMII','','','DL'],['EMII','WT','AI','']$$

  The resulting configuration after swap will be :

  $$['EMII','WT','AI',''],['EMII','','','DL'],['MB','DBMS','SC','DL']$$

* **2. Day-Wise mutation** We randomly choose two-day elements from the original chromosome who are further swapped with each other.

6 Acknowledgement

We want to give special thanks to **Prof. Bijan Bihari Misra** who is guiding us in the field of Genetic Algorithms along with **Prof. Rudra Mohan Tripathy**.

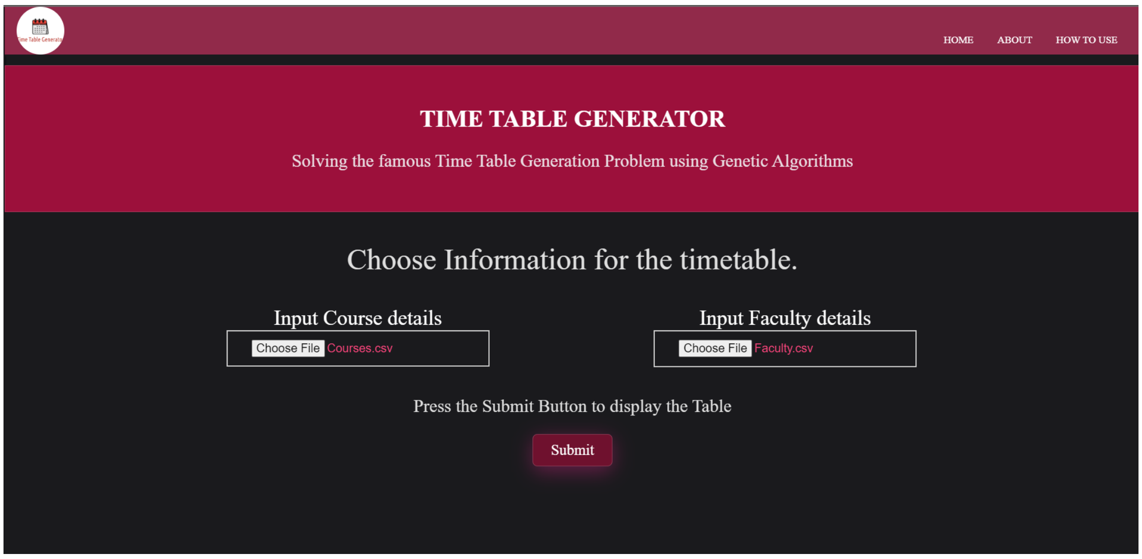
7 Appendix

1. Codes

- **Flask Main Code**

```python
def call_genetic():
call(['python','GeneticAlgo.py'])

app = Flask(__name__)
@app.route('/', methods = ['GET','POST'])
def index():
    return render_template('display.html')

@app.route('/data', methods = ['GET','POST'])
def data():
    if request.method == 'POST':
        f1 = request.files['courses']
        f1.save(f1.filename)
        f2 = request.files['faculty']
        f2.save(f2.filename)
    call_genetic()
    with open('Year1.csv', newline='') as f:
        reader = csv.DictReader(f)
        data1 = [row for row in reader]
    with open('Year2.csv', newline='') as f:
        reader = csv.DictReader(f)
        data2 = [row for row in reader]
    with open('Year3.csv', newline='') as f:
        reader = csv.DictReader(f)
        data3 = [row for row in reader]
    with open('Year4.csv', newline='') as f:
        reader = csv.DictReader(f)
        data4 = [row for row in reader]

# Create a list of headers

    header = list(data1[0].keys())
```

- **Front End Page**



TIME TABLE GENERATOR

Solving the famous Time Table Generation Problem using Genetic Algorithms

Choose Information for the timetable.

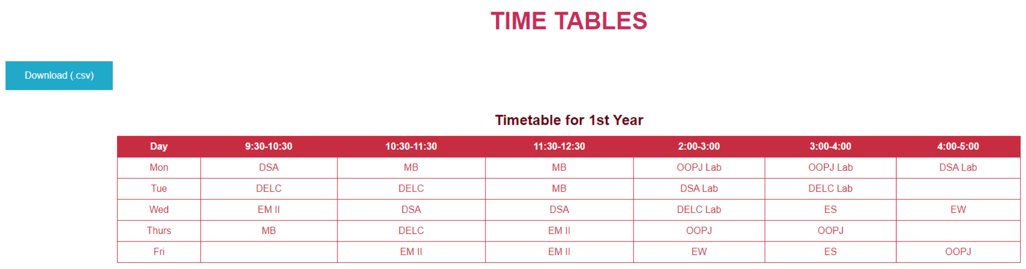
Input Course details

Input Faculty details

Press the Submit Button to display the Table

Figure 10: Main Flask Page

- **Result**



TIME TABLES

Timetable for 1st Year

Day	9:30-10:30	10:30-11:30	11:30-12:30	2:00-3:00	3:00-4:00	4:00-5:00
Mon	DSA	MB	MB	OOPJ Lab	OOPJ Lab	DSA Lab
Tue	DELC	DELC	MB	DSA Lab	DELC Lab	
Wed	EM II	DSA	DSA	DELC Lab	ES	EW
Thurs	MB	DELC	EM II	OOPJ	OOPJ	
Fri		EM II	EM II	EW	ES	OOPJ

Figure 11: Final Result

2. Flow Chart

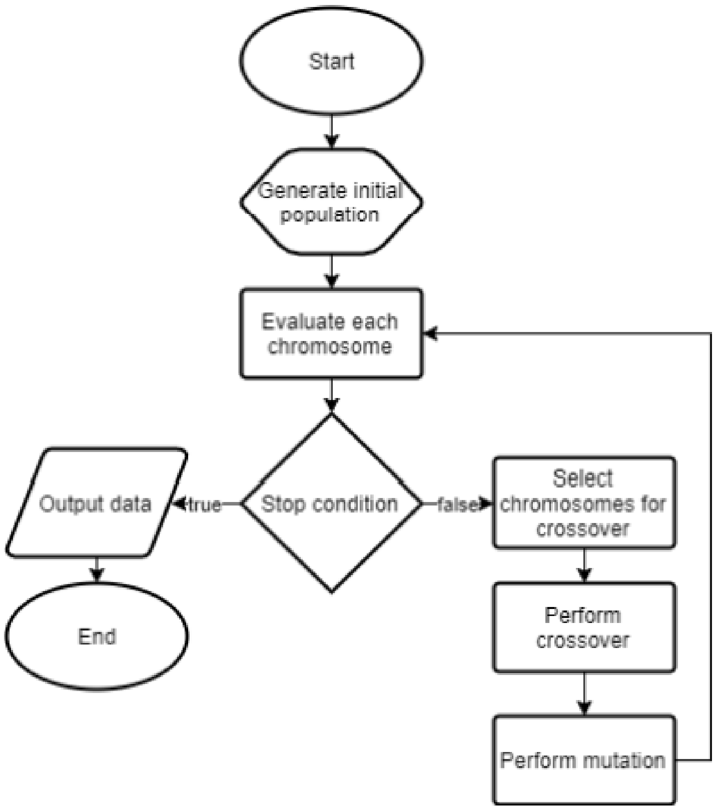


Figure 12: Basic Genetic Algorithm Process

Source: Google Images

References

[1] S. Deris, S. Omatu, H. Ohta, and P. Saad, "Incorporating constraint propagation in genetic algorithm for university timetable planning," *Engineering applications of artificial intelligence*, vol. 12, no. 3, pp. 241–253, 1999.

[2] A. Jain, S. Jain, and P. Chande, "Formulation of genetic algorithm to generate good quality course timetable," *International Journal of Innovation, Management and Technology*, vol. 1, no. 3, p. 248, 2010.

[3] A. Colorni, M. Dorigo, and V. Maniezzo, "Genetic algorithms and highly constrained problems: The time-table case," in *Parallel Problem Solving from Nature: 1st Workshop, PPSN I Dortmund, FRG, October 1–3, 1990 Proceedings 1*, pp. 55–59, Springer, 1991.

[4] G. Tai and D. Young, "Early generation selection for important agronomic characteristics in a potato breeding population," *American Potato Journal*, vol. 61, pp. 419–434, 1984.

[5] O. I. Obaid, M. Ahmad, S. A. Mostafa, and M. A. Mohammed, "Comparing performance of genetic algorithm with varying crossover in solving examination timetabling problem," *J. Emerg. Trends Comput. Inf. Sci*, vol. 3, no. 10, pp. 1427–1434, 2012.

[6] P. Chootinan and A. Chen, "Constraint handling in genetic algorithms using a gradient-based repair method," *Computers & operations research*, vol. 33, no. 8, pp. 2263–2281, 2006.

[7] V. Sapru, K. Reddy, and B. Sivaselvan, "Time table scheduling using genetic algorithms employing guided mutation," in *2010 IEEE International Conference on Computational Intelligence and Computing Research*, pp. 1–4, IEEE, 2010.

[8] I. De Falco, A. Della Cioppa, and E. Tarantino, "Mutation-based genetic algorithm: performance evaluation," *Applied Soft Computing*, vol. 1, no. 4, pp. 285–299, 2002.