VIETNAM NATIONAL UNIVERSITY HCMC

UNVERSITY OF INFORMATION TECHNOLOGY

FACULTY OF INFORMATION SYSTEMS

NGUYỄN HOÀNG LONG - NGUYỄN SƠN LÂM

GRADUATE THESIS

STUDY ON EXAM INVIGILATOR ASSIGNMENT ALGORITHMS

BACHELOR OF ENGINEERING IN INFORMATION SYSTEMS

HO CHI MINH CITY, 2020

VIETNAM NATIONAL UNIVERSITY HCMC

UNVERSITY OF INFORMATION TECHNOLOGY

FACULTY OF INFORMATION SYSTEMS

NGUYỄN HOÀNG LONG - 16520688

NGUYỄN SƠN LÂM - 16521708

GRADUATE THESIS

STUDY ON EXAM INVIGILATOR ASSIGNMENT ALGORITHMS

BACHELOR OF ENGINEERING IN INFORMATION SYSTEMS

ADVISOR

DR. ĐỖ TRỌNG HỢP

HO CHI MINH CITY, 2020

ASSESSMENT COMMITTEE

The Assessment Committee is established under the Decision . ..., date ....by Rector of the University of Information Technology.

1 .... - Chairman.

2 .... - Secretary.

3 .... - Member.

4 .... - Member

ACKNOWLEDGMENT

Throughout the completion of this project we have received a great deal of support and assistance.

First, we would like to express our dearest gratitude to our advisor Dr. Đỗ Trọng Hợp, and our mentor Dr. Nguyễn Thanh Bình for providing guidance and feedback throughout this project. They continuously provided encouragement and were always willing and enthusiastic to assist in any way they could throughout the research project.

Second, we want to give our special thanks to Bsc. Huỳnh Thiện Ý, whose support as part of his science paper allowed our studies to go the extra mile.

We are also thankful to the Faculty of Information System and all its staff for all the considerate guidance.

To conclude, we cannot forget to thank our family and friends for all the unconditional support, as well as providing a happy diversion to rest our mind outside of our research.

Once again, we sincerely thank you!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

GA	Genetic algorithm
LP	Linear programing
UI	User Interface
DB	Database

ABSTRACT

We propose a model for a decision support problem: exam invigilator assignment problem. For this problem, a wide range of schedules will be assigned for our invigilators; each person only performs a single task at a certain point in time. Meanwhile, the distance between exam days, assigned to each invigilator, is as little as possible. Regarding the solution, we first try to derive the problem with 2 sets of constraints: hard constraint and soft constraint. Based on the constraints, we design a genetic algorithm for finding solutions for this problem. The results are compared with findings of linear programing. Finally, we build an application that can run the algorithm and display the acceptable results from the algorithm.

Chapter 1: INTRODUCTION

1.1 Motivation

In life we often encounter problems related to scheduling such as scheduling machine operation, work (staff) schedules, sports competitions, and scheduling for the implementation of a project … With this kind of problems, we need to find a scheduling scheme that satisfies all constraints as well as efficiently exploits available resources, reducing implementation time and cost.

The scheduling problem belongs to the NP-complete problems, so it may not be possible to find the optimal solution. This is not a new problem and many algorithms have been introduced to solve such as hill climbing algorithm, graph coloring algorithm, approximation algorithm, ... However, these algorithms are often not generalized and only apply effectively on a small scale, with little data constraints.

The problem of assignment or scheduling work, especially in a university, is one such problem. There are many constraints to consider in this problem such as constraints on participants (invigilators, students), time constraints (number of shifts, number of shifts each day). Assigning invigilators is possible, but the results will likely make the invigilators unhappy. The schedule assigned to the invigilators can take several days during the exam, so they must be present at school only to supervise one or two subjects per day, which is time consuming, especially for those who far from school. So, it is necessary to build a timetable that satisfies all the above constraints and at the same time minimize the number of days that invigilators have to come to school and effectively exploits the resources for the exam invigilator assignment problem.

1.2 Objectives of the research

We focus on researching and applying genetic algorithm and linear programing to exam invigilator assignment problem to propose an optimal solution that satisfies the constraints and efficiently exploits human resources in a short time.

To achieve the above-mentioned objectives, we concentrate on the following specific tasks:

- Analyze the problem and then give reasonable solutions in building and implementing the system.
- Research the above algorithms and its applications to effectively solve the optimization problem.
- Apply the algorithms to the scheduling assignment problem for invigilators.
- Analyze and evaluate and compare the results obtained after applying them to sample dataset.

1.3 Target and scope of the research

Study the characteristic features of genetic algorithms, basic components of genetic algorithm such as population initiation, evaluation of fitness function, and genetic operators (selection, crossover, mutation), stop conditions and then compare the findings with the results of linear programing.

Apply genetic algorithm to the problem of assigning exam invigilators in universities with constraints and basic requirement and then build an application to display the acceptable results.

Because of the limited time and knowledge about Technologies, in this thesis, we only used some technologies:

- Programing languages: Python
- Database System: SQLite
- Software: PyCharm, Jupyter Notebook

1.4 Expected result

Fully aware about the exam invigilator assignment problem.

Fully aware of the strengths of genetic algorithms and linear programming in solving optimization problems.

Represent the problem in the form of integer programming.

Successfully implement the genetic algorithm to a working application.

1.5 Thesis structure

The thesis is organized as follows. This introduction summarized the motivation, the objectives, the target, the scope, and the expected result of this graduate thesis. Chapter 2 describes the theoretical background of the assignment problem that we need to research and discusses about the method choices. Chapter 3 proposes the solutions for our problem with its components. Chapter 4 explains the implementation detail of the application, including the descriptions, inputs, output, and the graphical user interface of the apps. Finally, Chapter 5 presents the conclusions of this graduate thesis and future developments that we will do in the future.

Chapter 2: BACKGROUND AND THEORY

2.1 Overview of assignment problem

Assignment or scheduling problems can be defined as a search for the optimal solution to perform various types of works or tasks bounded to a set of constraints. The main objective of these problems is to increase the appropriateness or satisfaction between resources and their assigned tasks under a number of constraints. Therefore, the scheduling problem is a very complex problem as the number of constraints that need to be satisfied increases.

Properties of the assignment problem:

- Resources: these are the input data of the problem.
- Tasks: assessed by performance criteria such as execution time, cost, resource consumption.
- Constraints: These are the conditions that need to be met in order for the problem to give the best solution
- Objective: to evaluate the optimal solution of the problem. When the goals are met, the constraints must also be fulfilled.

2.2 Exam invigilator assignment problem

We are dealing with the exam invigilator assignment problem, a type of assignment problem. This problem is quite complex, and its complexity varies depending on the specific requirements such as participants, time constraints … To address this problem, two types of constraints usually considered when addressing these problems are hard constraints and soft constraints. Hard constraints are those which cannot be violated by the system and strictly enforced by the process. Soft constraints, on the other hand, are those which violable and often treated as a penalty when the system violates.

We will go more into detail about **Exam invigilator assignment problem** in chapter 3.

2.3 Related methods in solving the scheduling problem

2.3.1 Linear programing (LP)

Linear programming is a simple technique where we depict complex relationships through linear functions and then find the optimum points. the solution of a linear programming problem reduces to finding the optimum value (largest or smallest, looking on the problem) of the linear expression (called the target function) subject to a group of constraints expressed as inequalities.

2.3.2 Genetic algorithm (GA)

Genetic algorithms are a subset of a larger class of evolutionary algorithms that describe a set of techniques inspired by natural selection such as inheritance, mutation, and crossover. GA require both a genetic representation of the solution domain and a fitness function to evaluate the solution domain. The technique generates a population of candidate solutions and uses the fitness function to select the optimal solution by iterating with each generation. The algorithm terminates when the satisfactory fitness level has been reached for the population or the maximum generations have been reached.

2.3 Methods summary and conclusion

Linear Programming and Genetic algorithm have many more advantages. Because of that, these methods are used a lot in solving optimization problems, including exam invigilator assignment problem.

In terms of optimization, linear programming is better but also more complex to implement than the genetic algorithm. Moreover, the genetic approach has many outstanding features such as does not require knowledge, avoids local optimization, performs well with all problems that require large solution space and can be applied to many different types of optimization problems.

Therefore, in this thesis, for simplicity and fastness, we will use genetic algorithm and linear programming to solve the exam invigilator assignment problem and apply genetic algorithm to build an application.

Chapter 3 EXAM INVIGILATOR ASSIGNMENT PROBLEM

3.1 Describe the problem

There are about 20000 candidates taking the exam in 12-15 days at each exam of University of Information Technology.

In each exam, there are about 120 invigilators. Before each exam, the school will announce the exam schedule and the required number of invigilators for each subject. Then invigilators will sign up to be able to supervise in the examination.

Each exam day will have 4 shifts.

Shift	Time span
1	07:30 AM – 09:00 AM
2	09:30 AM – 11:00 PM
3	13:30 PM – 15:00 PM
4	15:30 PM – 17:00 PM

Table 3.1 Duration of each shift per day

The task here is to assign invigilator to exam room, satisfy the constraints, which is not to assign one invigilator to 2 exam rooms at the same time, and each exam room must have the enough invigilators as required. After that, study and apply algorithms to minimize the number of days that invigilators have to come to school.

An acceptable solution of the problem must satisfy the following constraints:

Hard constraints:

- In a certain shift, invigilator is not allowed to oversee multiple rooms. (1)
- Each exam must meet the required number of invigilators. (2)

Soft constraints:

- The distance between two consecutive shifts of invigilator is as close as possible. (3)
- Each invigilator must meet the required number of shifts. (4)

3.2 Datasets:

We will use 2 datasets: invigilator dataset and exam schedule dataset and the datasets must meet the required column below:

- The exam schedule dataset must have 6 columns “STT”, “Tên MH”, “Ngày thi”, “Ca thi”, “Phòng thi”, “Số CBCT”.
- The invigilator dataset must have 2 columns “ID” and “Full_name”.

3.3 Genetic Algorithm (GA)

In this section, we will describe the design of GA to solve the exam invigilator assignment problem.

The algorithm started with generating a random population contain solutions. The population will be evolved through multiple generations by using the genetic operators (selection, crossover, mutation). After each generation, the fittest individuals, which are calculated by the fitness function, are passed on to the next generation. After the evolution process, the fittest individuals in the final population will be the best possible solution for the problem.

To apply GA to solving a certain problem, it is necessary to do some of the following important tasks:

1. Choose how to represent the model of chromosomes so that each chromosome can hold a solution to the problem.
2. Building adaptive evaluation function (fitness function) for each chromosome. This is a difficult step and affects the effectiveness of the algorithm.
3. Select suitable genetic operators, in which focus on three main operators: selection, crossover and mutation.
4. Determine parameters of GAs such as population size, probability of crossover, probability of mutation.
5. Determine the stopping conditions for evolution.

3.3.1 Individual representation

Individual representation plays a vital role in solving the scheduling problem. In this problem, we will use a binary representation method to describe the individual in the population. Instead of pairing an invigilator with a specific exam subject, we align that invigilator with the shift in which the exam subject will take place.

For binary representation, each chromosome can be thought of as a two-dimensional array: The first dimension represents the invigilators, the second dimension represents the shifts. The elements in the array will have a value of 1 or 0.

Where:

- 1 – invigilator is assigned to that shift.
- 0 – opposite.

	Shift 1	**Shift 2**	**Shift 3**	**…**	**Shift n**
Invigilator 1	0	1	0	…	0
Invigilator 2	0	0	1	…	1
Invigilator 3	1	1	0	…	0
…	…	…	…	…	…
Invigilator n	0	1	1	…	1

Table 3.2 Two-dimensional array of each chromosome

Because our objective is to minimize the number of days that the invigilators have to go to school, we will initialize each individual with a solution that satisfied hard constraints in order to shorten the running time of the algorithm. With the above binary representation method, we can help invigilator avoid viewing multiple exam subjects at the same shift, which removes the (1) constraint.

3.3.2 Individual adaptive function (Calculation)

Adaptability is the ability of each chromosome (solution) to adapt to the environment (problem). Building adaptability is an important step in the GA. To evaluate the adaptability of chromosomes, GA use a measuring function called the Fitness function.

The fitness function is the function used to evaluate the fit of a solution or chromosome. The fitness function takes a parameter, the binary array of a chromosome, and returns a real number. Depending on the value of this real number, we know the fit of that individual (for example, with the maximum finding problem, the larger the return value, the better the individual, and vice versa, and for the minimum finding problem, the smaller the return value, the better the individual). The problem is to optimize each invigilator's time at school. So this will be the problem of finding the minimum value of the total distance between shifts of each invigilator.

The evaluation of the individual's fitness is based on the number of violations of the hard constraints. To do this, we first compute the individual's fit score based on each constraint, and then add up all of the fit score to get the individual's total fit score.

To increase the effectiveness of the algorithm, depending on the type of constraint, we multiply the number of violations by an appropriate weight.

3.3.2.1 Weight function

To reduce the number of days at school, each invigilator must reduce the number of scheduled vacancies between the 2 shifts. On a day that invigilator supervises all 4 shifts, then it will not be counted as a violation of the constraint.

Specify:

- Input: the binary array of the individual, **pen_val = 10.**
- Output: total distance between invigilators shifts.

Describe:

1. Let **weight = 0**
2. Loop for each row (i = 0) of the individual.

   a. Calculates the number of values equal to 1 per row: **num_of_ones.**

   b. Calculates the expected number of days: **actual_days = num_of_ones/4.**

   c. Find the position of value 1 on each row.

   d. Find the maximum and minimum position of value 1 on each row: **max_pos, min_pos.**

   e. Calculates the number of values equal to 0 per row: **num_of_zero = (max_pos – min_pos + 1) – num_of_ones**

   f. Calculates the actual number of days that the invigilator goes to see the exam: **day_spans = ((max_pos + 1) / 4) – min_pos / 4**.

   g. Calculate penalty days: **day_pens = day_spans – actual_days**

   h. Calculate the distance: **weight = weight + (num_of_zero + day_pens * pen_val)**

3. End of the loop

4. Return weight

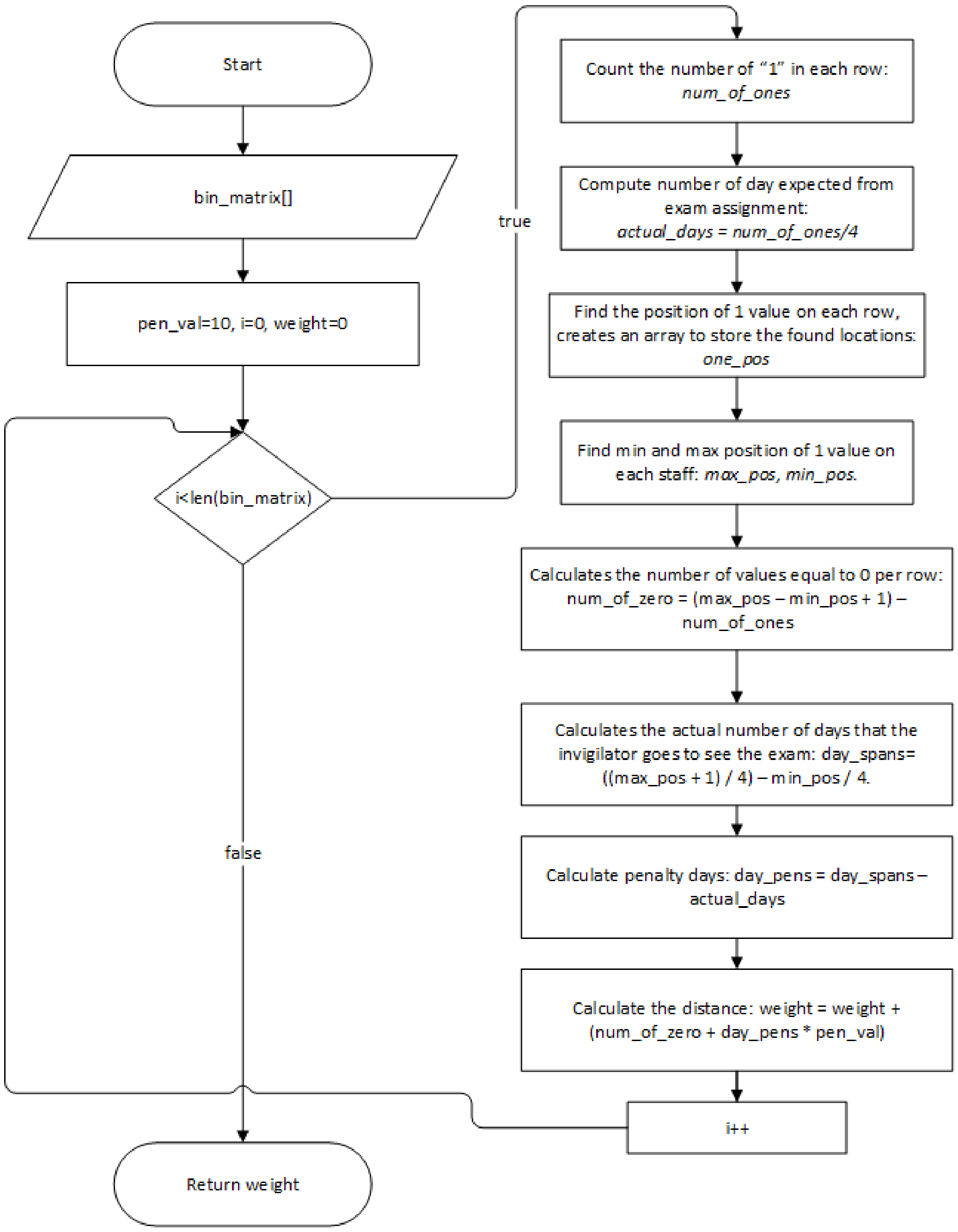

Start
bin_matrix[]
pen_val=10, i=0, weight=0
i<len(bin_matrix)
true
false
Count the number of "1" in each row: num_of_ones
Compute number of day expected from exam assignment: actual_days = num_of_ones/4
Find the position of 1 value on each row, creates an array to store the found locations: one_pos
Find min and max position of 1 value on each staff: max_pos, min_pos.
Calculates the number of values equal to 0 per row: num_of_zero = (max_pos – min_pos + 1) – num_of_ones
Calculates the actual number of days that the invigilator goes to see the exam: day_spans= ((max_pos + 1) / 4) – min_pos / 4.
Calculate penalty days: day_pens = day_spans – actual_days
Calculate the distance: weight = weight + (num_of_zero + day_pens * pen_val)
i++
Return weight

Figure 3.1 Weight compute function flowchart

3.3.2.2 Fitness function

The fitness function gives a fitness score to each individual that participates in reproducing. The more score an individual has will increase the probability to be selected to pass to next generation.

Specify:

- Input: the binary array of the individual. (**individual[ ]**)
- Output: fitness of individual.

Describe:

1. Let **fitness = 0**
2. Fitness = weight_compute(individual)
3. Loop for each row (i = 0) of the individual
    a. Calculate total value of each column: **Sum**
    b. Compare Sum with the required number of invigilators of each respective shift, if : **fitness = fitness + 20000** then break.
4. End of the loop
5. Return **fitness**

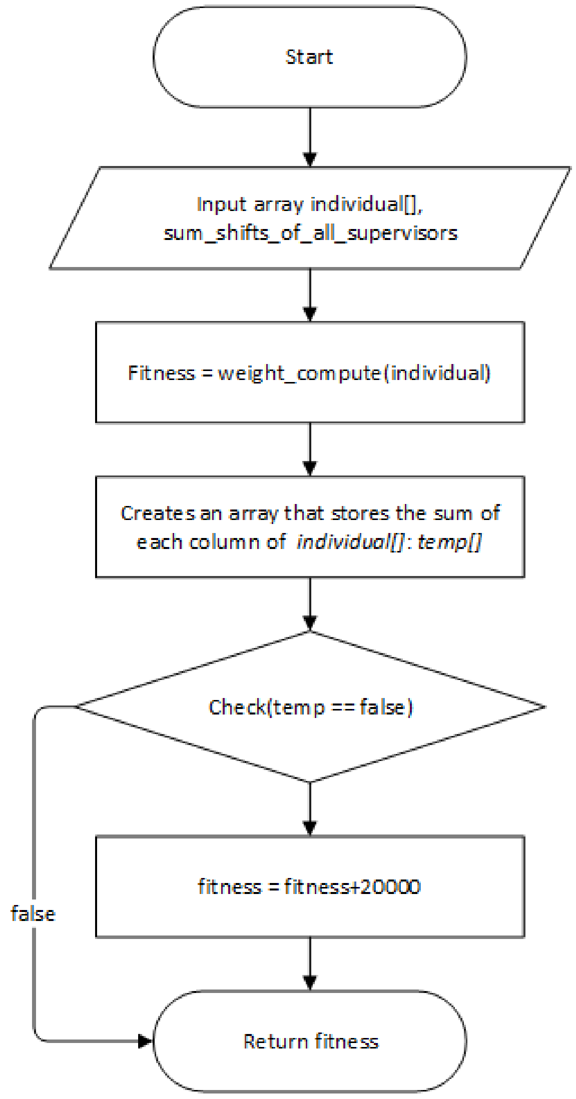

Start
Input array individual[], sum_shifts_of_all_supervisors
Fitness = weight_compute(individual)
Creates an array that stores the sum of each column of *individual[]*: *temp[]*
Check(temp == false)
false
fitness = fitness+20000
Return fitness

Figure 3.2 Compute fitness function flowchart

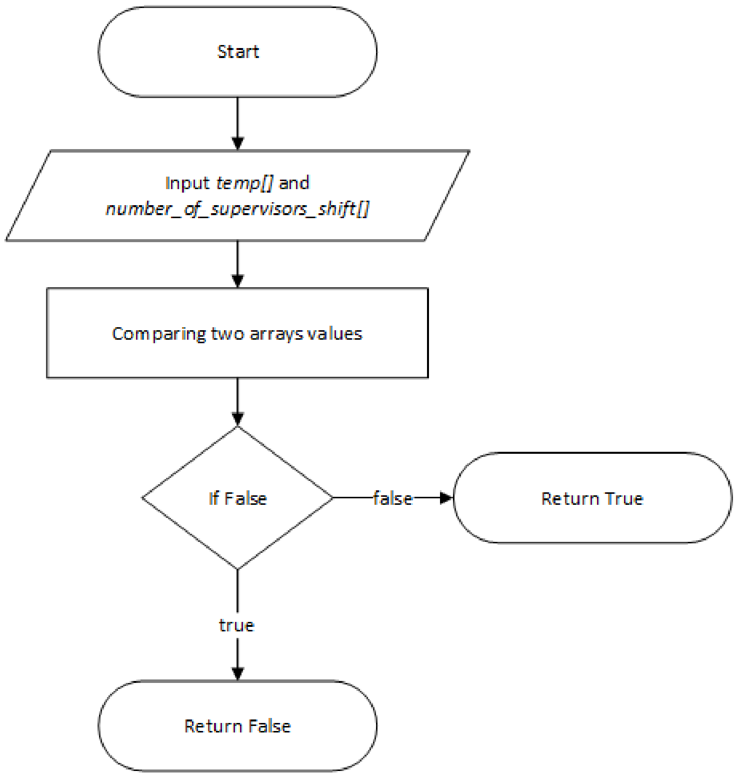

Start
Input *temp[]* and *number_of_supervisors_shift[]*
Comparing two arrays values
If False
false
Return True
true
Return False

Figure 3.3 Check function flowchart

3.3.3 Genetic operators

3.3.3.1 Selection operator

In the population, we arrange the adaptability of the individuals to increase gradually meaning their fitness value becomes smaller. The selection process is to select highly adaptable individuals from the current population to create its next generation.

- Calculate the Fitness value of each chromosome in the current population.
- Arrange the chromosomes in the population in descending order of Fitness value.
- Choose which individuals to use for breeding and mutating.
- Select 10 best individuals at the end of each generation and past to next generation, eliminating others.

Specify:

- Input: population
- Output: selected individual

Describe:

1. **Index1**=randomize value from 0 to number of individuals inside population – 1
2. **Index2**=randomize value from 0 to number of individuals inside population – 1
3. If **Index1** < **Index 2**
    a. Return the individual located at **Index2**
4. Return the individual located at **Index1**

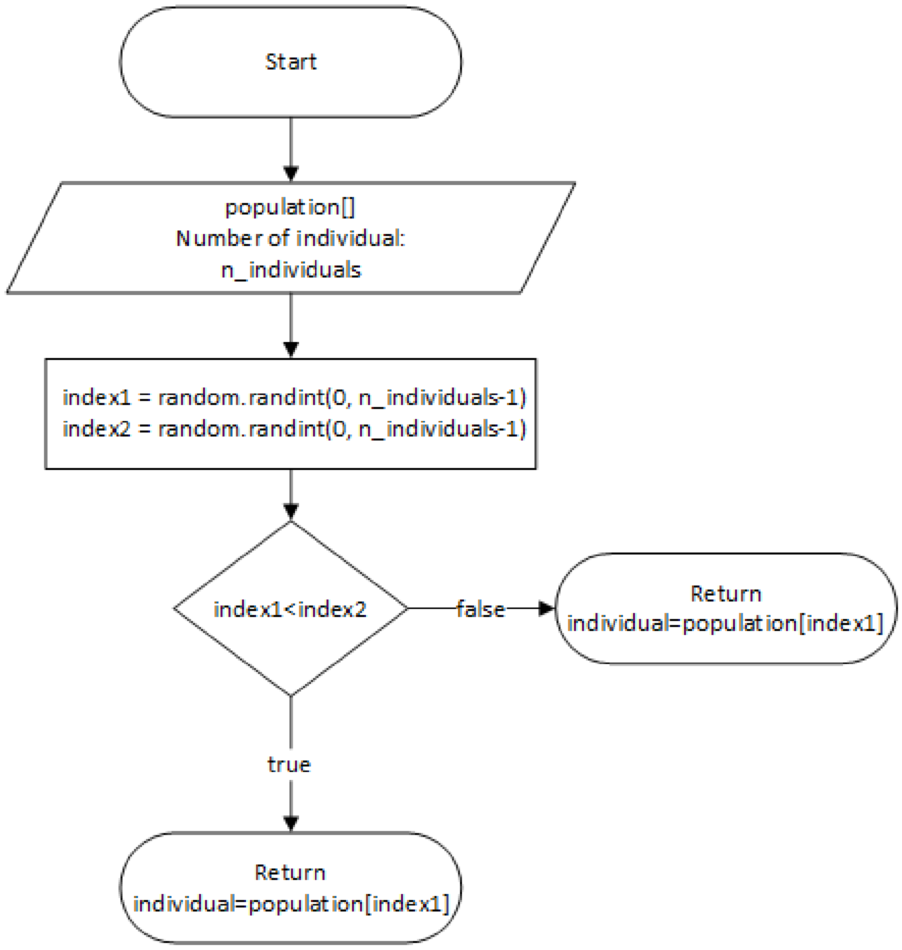

Start
population[]
Number of individual:
n_individuals
index1 = random.randint(0, n_individuals-1)
index2 = random.randint(0, n_individuals-1)
index1<index2
false
Return
individual=population[index1]
true
Return
individual=population[index1]

Figure 3.4 Selection flowchart

3.3.3.2 Crossover operator

Due to the chromosomal structure with columns created from hard constraints, we cannot choose column hybridization. Hence, we chose a hybrid multi-point crossover to apply to the problem with randomly selected points.

Specify:

- Input: **parent1[ ]**, **parent2[ ]**, **crossover_rate = 0.5**
- Output: **Child1[ ]**, **Child2[ ]**

Describe:

1. Create 2 new individuals resemble two input individuals: **Child1 = parent1**, **Child2 = parent2**
2. Loop for each row (i=0) of the parent individual
    a. Loop for each column (j=0) of the parent individual
        i. If random value from 0 to 1 smaller than **crossover_rate**
            - **Child1[i][j]=parent2[i][j]**
            - **Child2[i][j]**=**parent1[i][j]**

    b. End of the loop
3. End of the loop
4. Return **Child1[ ]**, **Child2 [ ]**

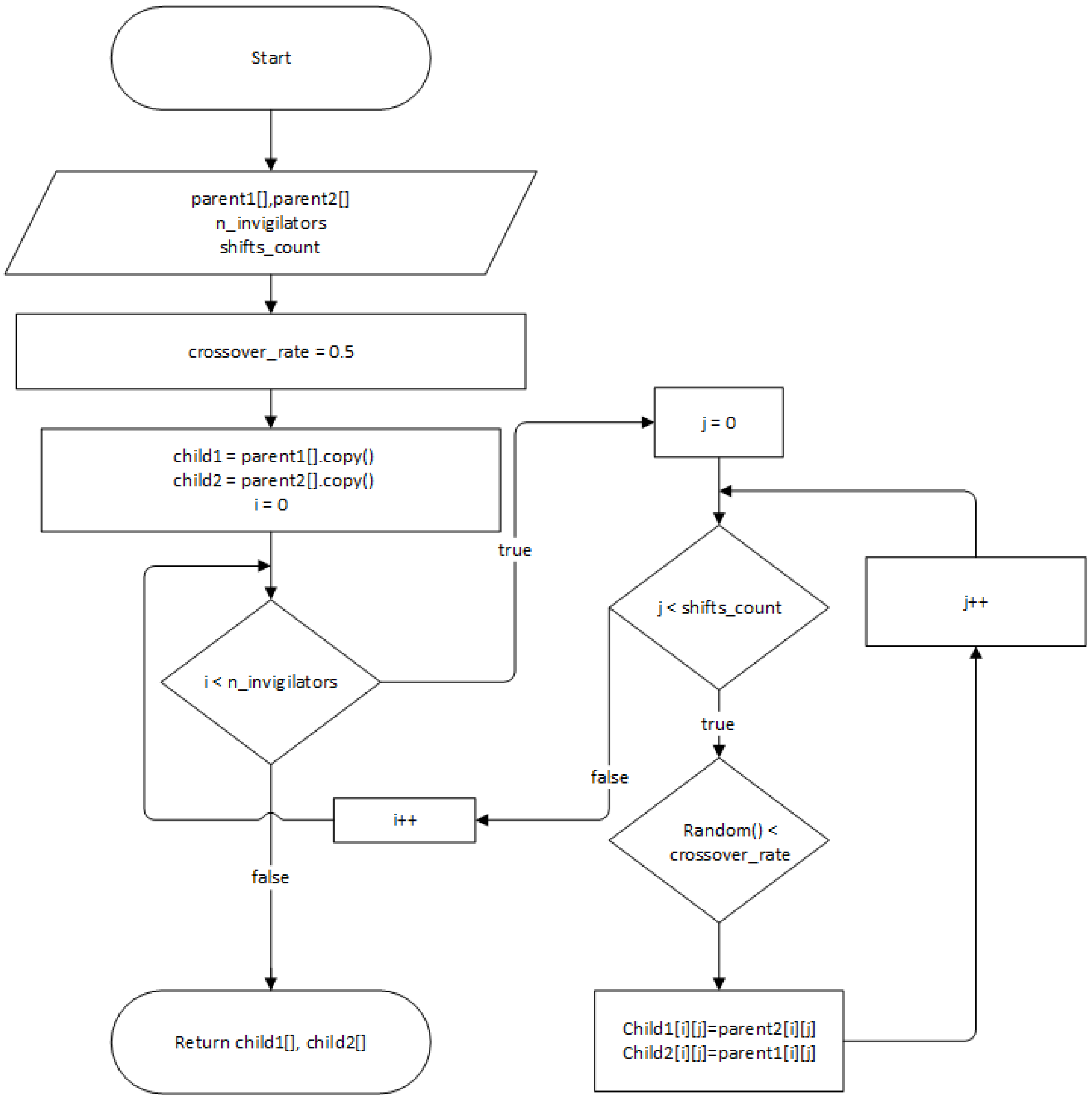

Start
parent1[],parent2[]
n_invigilators
shifts_count
crossover_rate = 0.5
child1 = parent1[].copy()
child2 = parent2[].copy()
i = 0
i < n_invigilators
true
false
i++
j = 0
j < shifts_count
true
false
Random() < crossover_rate
Child1[i][j]=parent2[i][j]
Child2[i][j]=parent1[i][j]
j++
Return child1[], child2[]

Figure 3.5 Crossover flowchart

3.3.3.3 Mutation operator

Mutation is a random alteration of one or more genetic components of an individual in a previous generation, creating a completely new individual in the next generation. But mutation can only be allowed to occur with very low frequency, as this can disturb the result and make the algorithm no longer effective.

Specify:

- Input: **individual[ ]**, **mutation_rate = 0.05**
- Output: **individual[ ]**

Describe:

```
1. Create new individual resemble input individual: individual_m[]= individual[ ]
2. Loop for each row (i=0) of parent individual
    a. Loop for each column (j=0) of parent individual
        i. If random value from 0 to 1 smaller than mutation_rate
            - Individual_m[i][j] = random from 2 value (0,1)
    b. End of the loop
3. End of the loop
4. Return individual_m[ ]
```

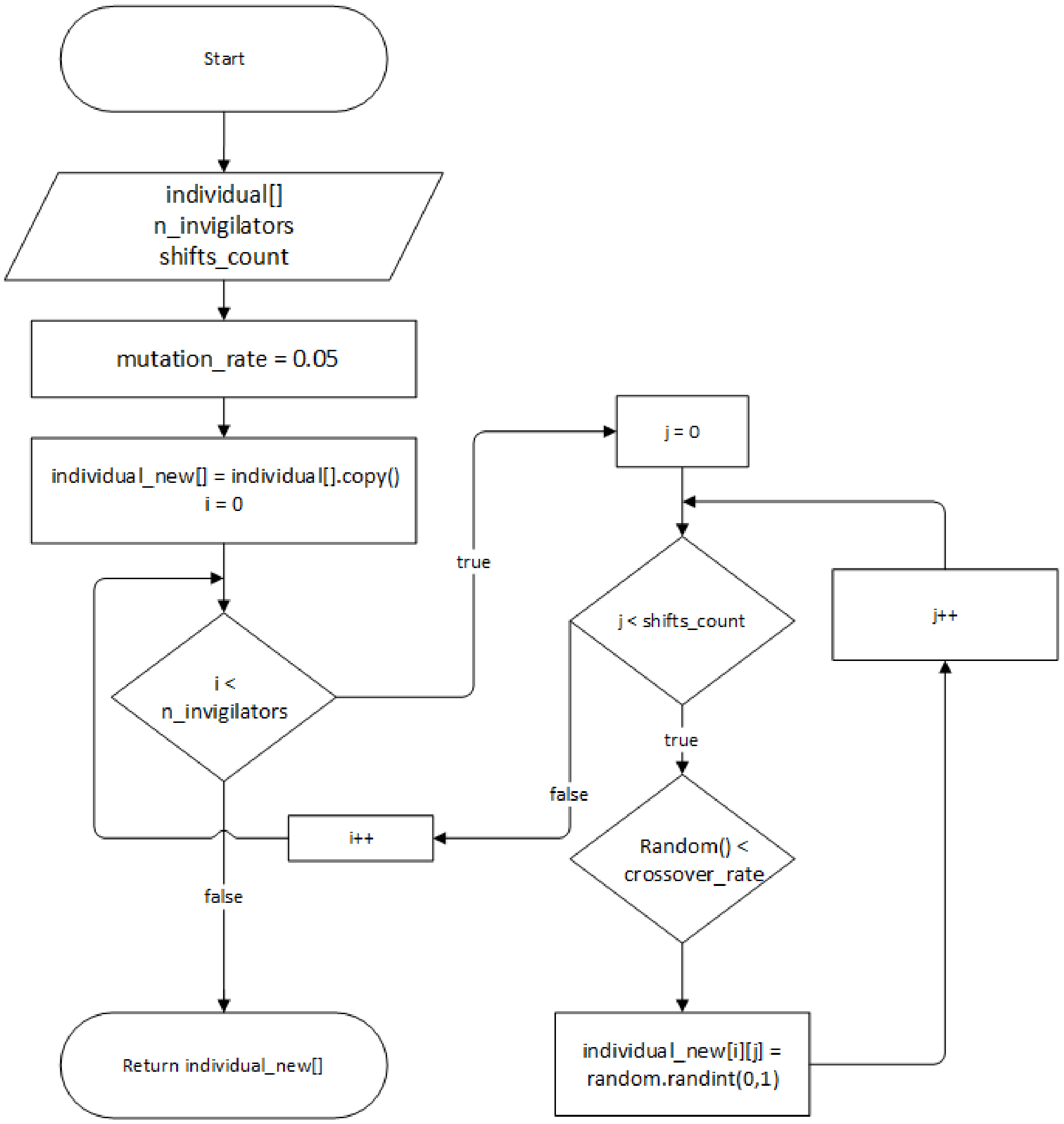


Figure 3.6 Mutation flowchart

3.3.4 Brute-force search

Specify:

Input:

- Number of invigilators: **n_invigilators**

- Number of shifts: **shifts_count**
- Total number of shift of invigilators: **total_shifts**
- Average shifts: **average_shifts = total_shifts//n_invigilators**
- Number of invigilators watch more than the average shift: **f=total_shifts%n_invigilators**
- Number of invigilators per shift: **number_of_supervisors_per_shift[]**

Output:

- Array for solutions.

Describe:

#Funtion 1: Sort by the number of shifts per invigilator

Since there are no specific data, in the problem the number of exams of each examiner is divided equally.

This function is used to sort individuals with rows that are each invigilator's shifts.

```
def arrange_supplies(arr):
   k=0
   for i in range(0, n_invigilators):
      sum=0
      for j in range(0, shifts_count):
         if k<f :
             if sum<average_shifts+1 :
                arr[0][i][j]=1
                sum=sum+arr[0][i][j]
        else :
            if sum<average_shifts :
                arr[0][i][j]=1
                sum=sum+arr[0][i][j]
    random.shuffle(arr[0][i])   # Mix random position of row i
    k=k+1
return arr
```

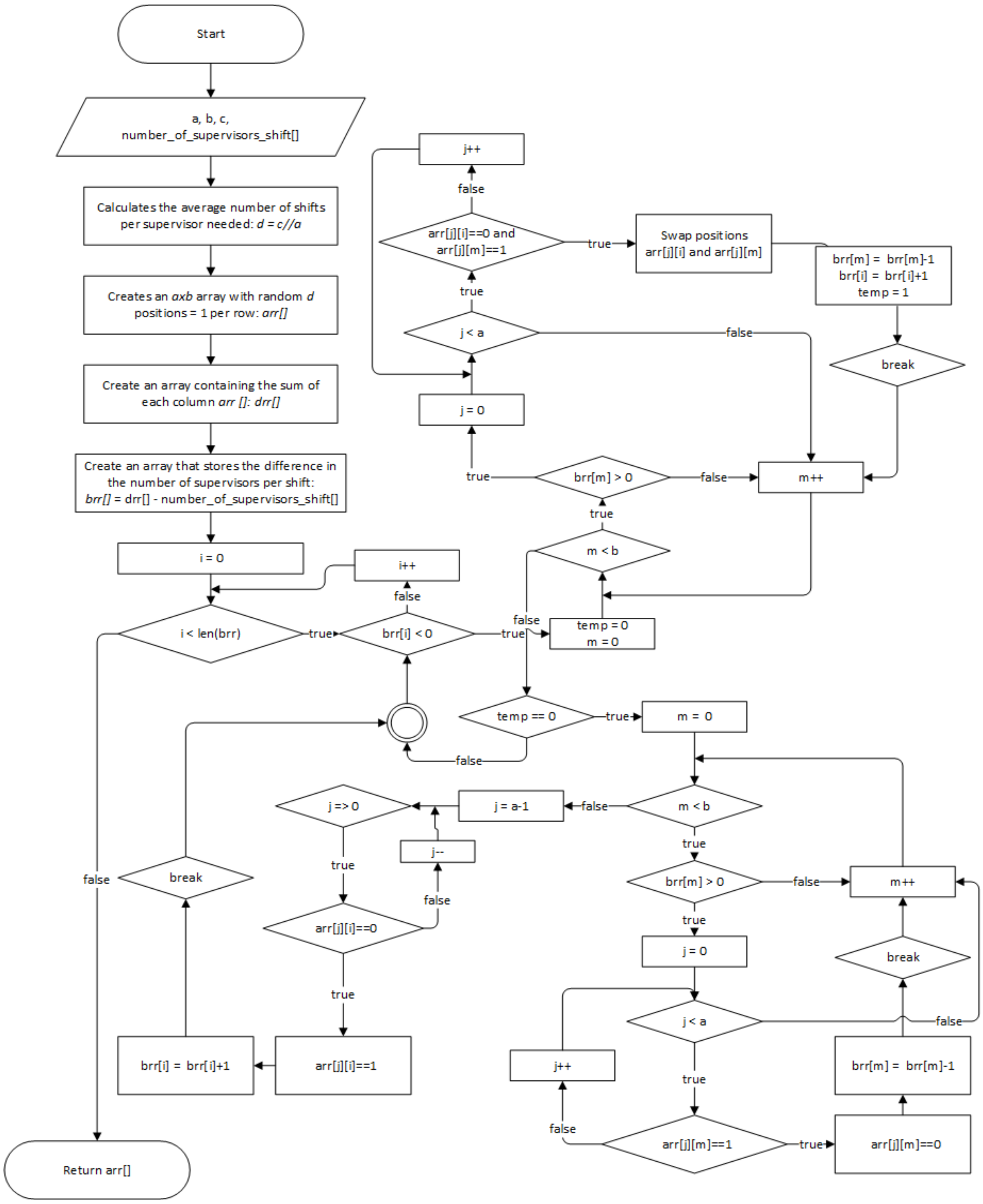

Start
a, b, c, number_of_supervisors_shift[]
Calculates the average number of shifts per supervisor needed: d = c//a
Creates an axb array with random d positions = 1 per row: arr[]
Create an array containing the sum of each column arr []: drr[]
Create an array that stores the difference in the number of supervisors per shift: brr[] = drr[] - number_of_supervisors_shift[]
i = 0
i < len(brr)
true
false
brr[i] < 0
i++
j++
arr[j][i]==0 and arr[j][m]==1
Swap positions arr[j][i] and arr[j][m]
brr[m] = brr[m]-1
brr[i] = brr[i]+1
temp = 1
break
j < a
j = 0
brr[m] > 0
m++
m < b
temp = 0
m = 0
temp == 0
m = 0
m < b
j = a-1
j => 0
j--
arr[j][i]==0
arr[j][i]==1
brr[i] = brr[i]+1
break
brr[m] > 0
m++
j = 0
j < a
j++
arr[j][m]==1
arr[j][m]==0
brr[m] = brr[m]-1
break
Return arr[]

Figure 3.7 Brute force search flowchart

3.3.5 Genetic algorithm execution

Specify:

Input:

- Number of invigilators: **n_invigilators**
- Number of shifts: **shifts_count**
- Total number of shift of invigilators: **total_shifts**
- Number of invigilators per shift: **number_of_supervisors_per_shift[ ]**
- Number of individuals: **n_individuals**
- Number of generations: **n_generations**

Output:

- invigilator's roster, fitness

Describe:

1. Randomly generate individual (**n_individuals**), in which each individual is created by brute-force search.
2. Loop start from i=0 to **n_generations**
    a. Align the individual of the current population based on descending fitness value
    b. Create a new population containing the best 10 individuals of the current population
    c. Loop to add more individual to the new population starts with j=10 to (**n_individual -** 1)

        i. Selection: randomly select two individuals from the current population
        ii. Crossover creates 2 new individuals
        iii. Mutating 2 individual
        iv. If individual does not satisfy the hard constraints, then we replace with new individual form brute-force search
    d. End of the loop
3. End of the loop
4. Get the final value of the population and its fitness

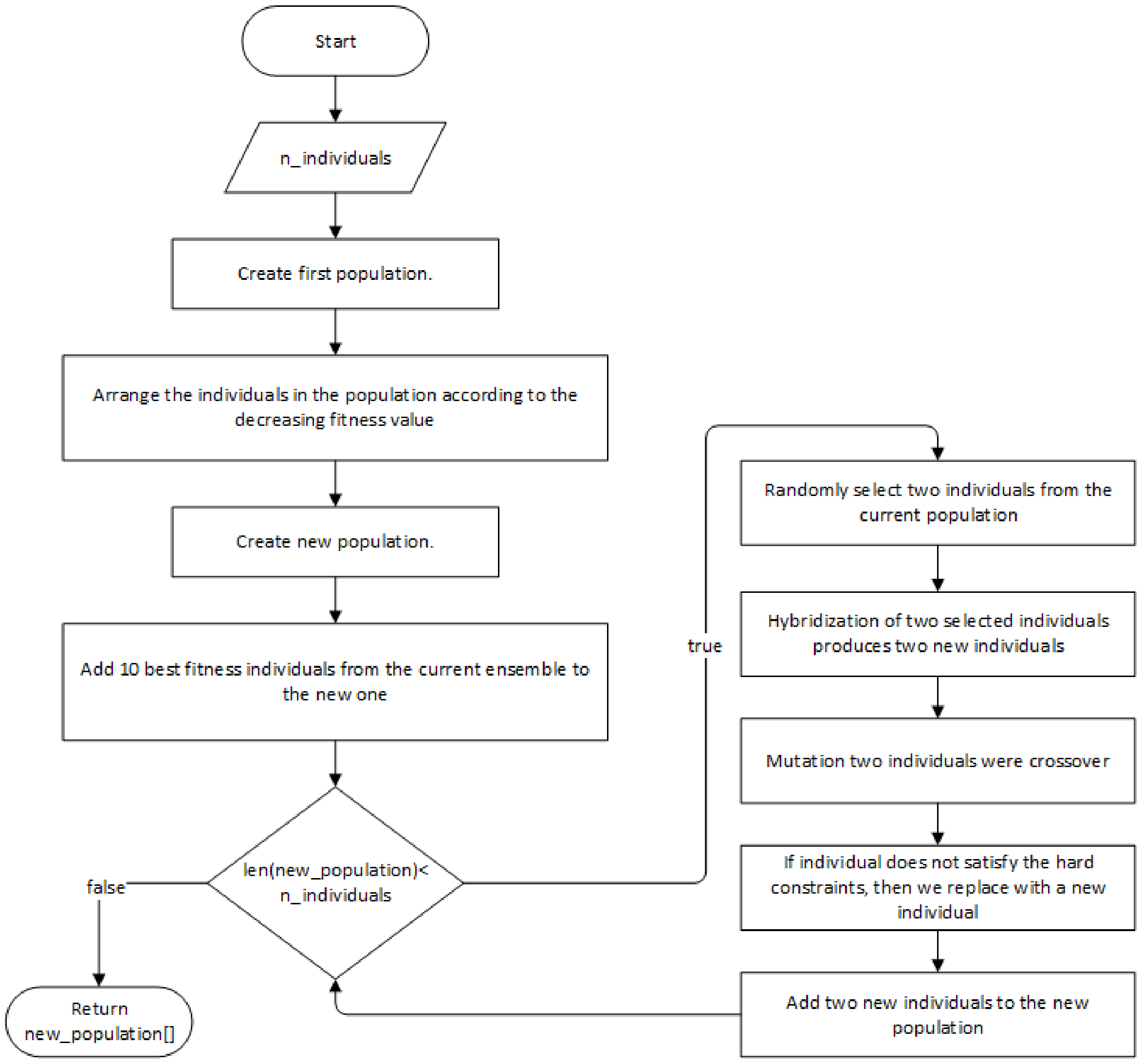

Start
n_individuals
Create first population.
Arrange the individuals in the population according to the decreasing fitness value
Create new population.
Add 10 best fitness individuals from the current ensemble to the new one
len(new_population)< n_individuals
false
Return new_population[]
true
Randomly select two individuals from the current population
Hybridization of two selected individuals produces two new individuals
Mutation two individuals were crossover
If individual does not satisfy the hard constraints, then we replace with a new individual
Add two new individuals to the new population

Figure 3.8 Genetic algorithm flowchart

3.4 Linear Programing (LP)

3.4.1 The mathematical model of the problem

Before delving into the mathematical model, we define the parameters used:

Parameters:

- m: Number of invigilators

- n: Number of shifts.
- l: Vector of size m represents number of shifts each staff must overlook.
- r: Vector of size n represents the required number of invigilators for each shift.

Decision variables:

- $X_{(m,n)}$: (m x n) matrix, is the solution to our problem.
- max: Vector represents the largest position of "l" element on each row.
- min: Vector represents the smallest position of "1" element on each row.

Instead of dividing the shifts by date, we flatted the shifts continuous through time for each invigilator for convenience. This type of representation automatically avoids the hard constraint that each invigilator cannot be at more than one shift at the same time. Followed by the above notation. We are mathematically the hard constraints as bellow:

Each invigilator must meet the required number of exams.

$\sum_{j=1}^{n} X_{i,j} \geq l_j$

Each exam has a certain number of Invigilators.

$\sum_{j=1}^{m} X_{i,j} = r_i$ (2)

Each element in the X array represents whether the "i" invigilator has worked in the "j" shifts or not.

$X_{i,j} \in (0, 1)$ (3)

The main objective of our model is to reduce the number of days each lecturer must be present at school. This number depends on the positions of the first and last elements whose value is "1". Due to this observation, we minimize the sum of all distances from the first element has value ”1” to the last one on each row of the decision variable. The mathematical form is formulated as following:

$$minimize \sum_i \left(\max_j\{jX_{i,j}\} - \min_j\{jX_{i,j}\}\right) \qquad (4)$$

Using objective function (4) combined with equation sets (1), (2), (3) then convert the problem to integer programming, we obtained the mathematical formulation bellow:

$$minimize \sum_i (\max_i - \min_i) \qquad (5)$$

Subject to:

- constraints to find maximum position:

$$\begin{bmatrix} max_i \\ max_i \\ max_i \\ \cdots \\ max_i \end{bmatrix} \geq \begin{bmatrix} 1+a \\ 2+a \\ 3+a \\ \cdots \\ (j+1)+a \end{bmatrix} \begin{bmatrix} X_{i,0} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & X_{i,j} \end{bmatrix} \begin{bmatrix} a \\ a \\ a \\ \cdots \\ a \end{bmatrix}$$

with

- a: very large constant

To shorten the above equation we can define vectors as follow:

$$Max_1 = \begin{bmatrix} max_1 \\ max_1 \\ max_1 \\ \cdots \\ max_1 \end{bmatrix}, \mathrm{Y} = \begin{bmatrix} 1+a \\ 2+a \\ 3+a \\ \cdots \\ (j+1)+a \end{bmatrix}, \mathrm{A} = \begin{bmatrix} a \\ a \\ a \\ \cdots \\ a \end{bmatrix},$$

$$I_i = diag(X_{i,0}, X_{i,1}, X_{i,2}, X_{i,3}, \dots, X_{i,j})$$

Equation has been reduced to:

$$Max_0 \geq YI_0 + A$$

$$\vdots\, i \qquad (6)$$

$$Max_i \geq YI_i + A$$

Similarly, we find the minimum position:

$$Min_i = \begin{bmatrix} min_i \\ min_i \\ min_i \\ \dots \\ min_i \end{bmatrix}, \text{Z} = diag(1-a, 2-a, \dots, (j+1)-a), \text{A} = \begin{bmatrix} a \\ a \\ a \\ \dots \\ a \end{bmatrix}$$

$$I_i = diag(X_{i,0}, X_{i,1}, X_{i,2}, X_{i,3}, \dots, X_{i,j})$$

Reduced equation:

$$Min_0 \leq ZI_0 + A$$

$$\vdots\, i \qquad (7)$$

$$Min_i \leq ZI_i + A$$

From equations (1), (2), (3), (5), (6) and (7), we have the target function and the necessary constraints for optimization.

3.4.2 Finding solution for target function

We will use Google Colab and PuLP to design and implement LP.

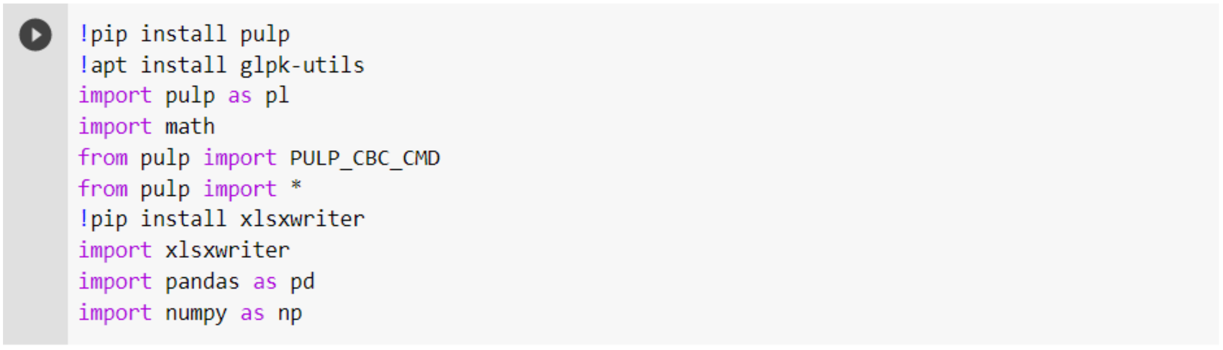
Google Colab is a product from Google Research, it allows running python code through the browser, especially suitable for data analysis, machine learning and education. Colab does not require any computer installation or configuration, everything can be run through the browser, you can use your computing resources from the high-speed CPUs and both GPUs and TPUs are provided for you.

PuLP is a library for the Python scripting language that enables users to describe mathematical programs. PuLP works entirely within the syntax and natural idioms of the Python language by providing Python objects that represent optimization problems and decision variables and allowing constraints to be expressed in a way that is very similar to the original mathematical expression. To keep the syntax as simple and intuitive as possible, PuLP has focused on supporting linear and mixed-integer models.[2]

PULP has many solvers like: 'GLPK_CMD', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'PULP_CBC_CMD', ...

In our optimization problem, we use the solver 'PULP_CBC_CMD' because this solver can find a solution in a short time while the input data is very large.

First, we import and install the required libraries into Colaboratory:

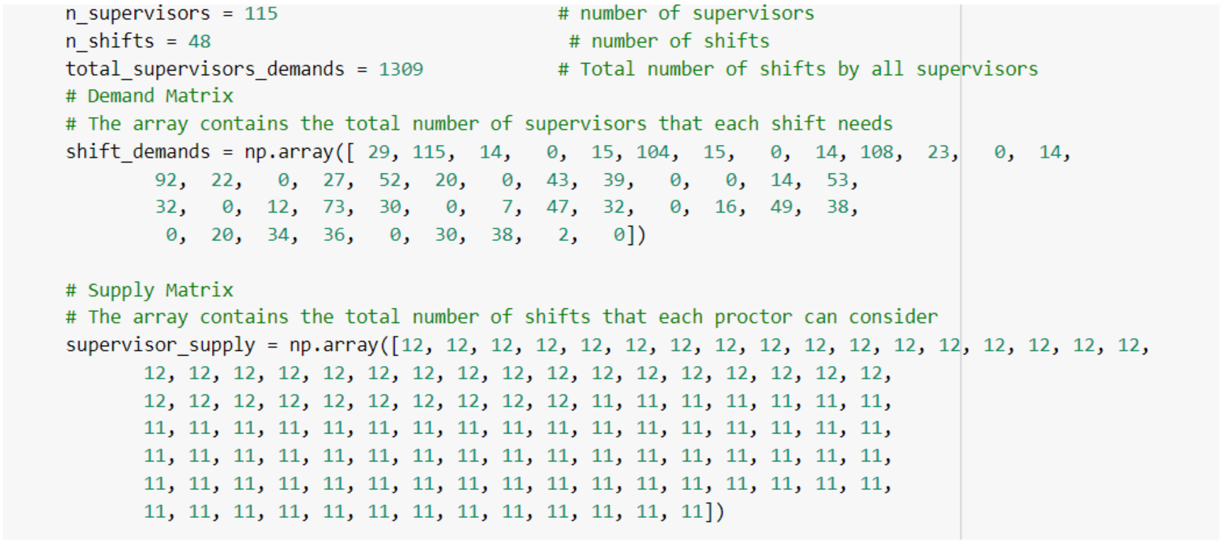


```
!pip install pulp
!apt install glpk-utils
import pulp as pl
import math
from pulp import PULP_CBC_CMD
from pulp import *
!pip install xlsxwriter
import xlsxwriter
import pandas as pd
import numpy as np
```

Figure 3.9 Import pulb code

Input data:

Define the data and assign it to variables that can then be used to include the model, the target function, and the constraints.



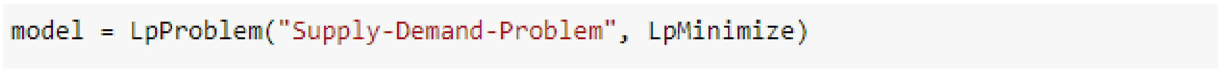
```
n_supervisors = 115                          # number of supervisors
n_shifts = 48                                 # number of shifts
total_supervisors_demands = 1309             # Total number of shifts by all supervisors
# Demand Matrix
# The array contains the total number of supervisors that each shift needs
shift_demands = np.array([ 29, 115,  14,   0,  15, 104,  15,   0,  14, 108,  23,   0,  14,
        92,  22,   0,  27,  52,  20,   0,  43,  39,   0,   0,  14,  53,
        32,   0,  12,  73,  30,   0,   7,  47,  32,   0,  16,  49,  38,
         0,  20,  34,  36,   0,  30,  38,   2,   0])

# Supply Matrix
# The array contains the total number of shifts that each proctor can consider
supervisor_supply = np.array([12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
       12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
       12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 11, 11, 11, 11, 11, 11, 11,
       11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
       11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
       11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
       11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11])
```

Figure 3.10 Define needed data

Model initialization:

We can initialize the model by calling the LpProblem() function. The first argument in the function represents the name we want to give our model. The second argument tells our model whether we want to minimize or maximize our objective function.



```
model = LpProblem("Supply-Demand-Problem", LpMinimize)
```

Figure 3.11 Model initialization

Define and initialize decision variables:

We use classes:

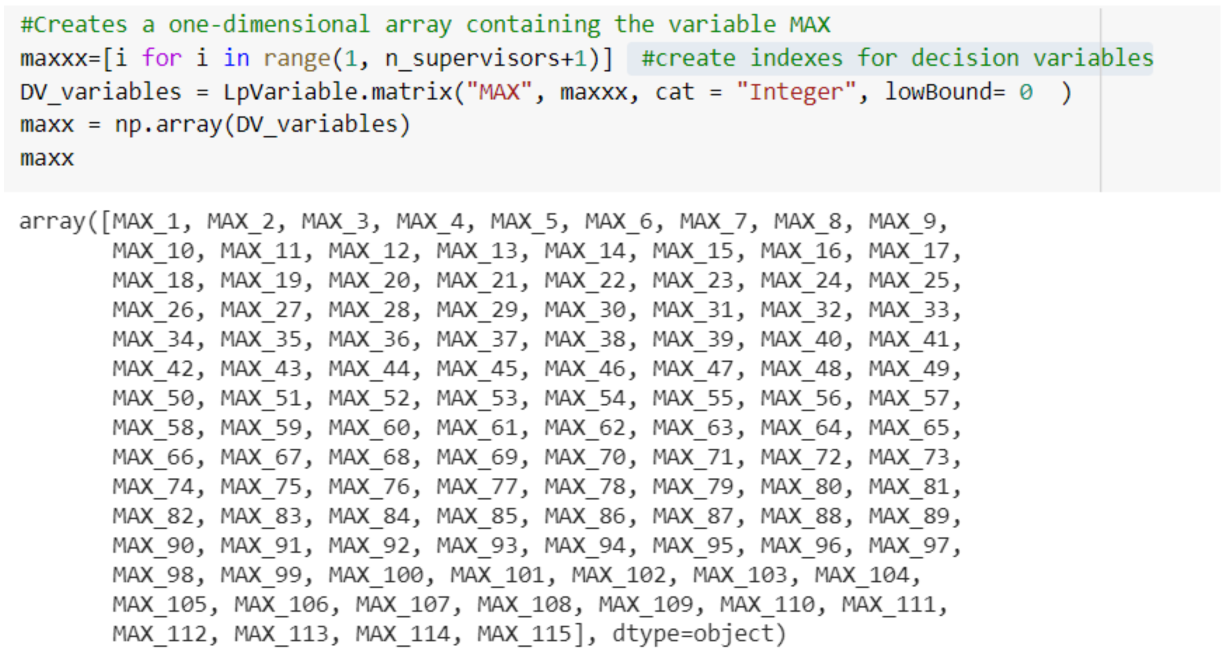
pulp.**LpVariable**(*name*, *lowBound=None*, *upBound=None*, *cat='Continuous'*, *e =None)*

This class models an LP Variable with the specified associated parameters

Parameters:

- name – The name of the variable used in the output .lp file
- lowbound – The lower bound on this variable's range. Default is negative infinity
- upBound – The upper bound on this variable's range. Default is positive infinity
- cat – The category this variable is in, Integer, Binary or Continuous(default)
- e – Used for column based modelling: relates to the variable's existence in the objective function and constraints.

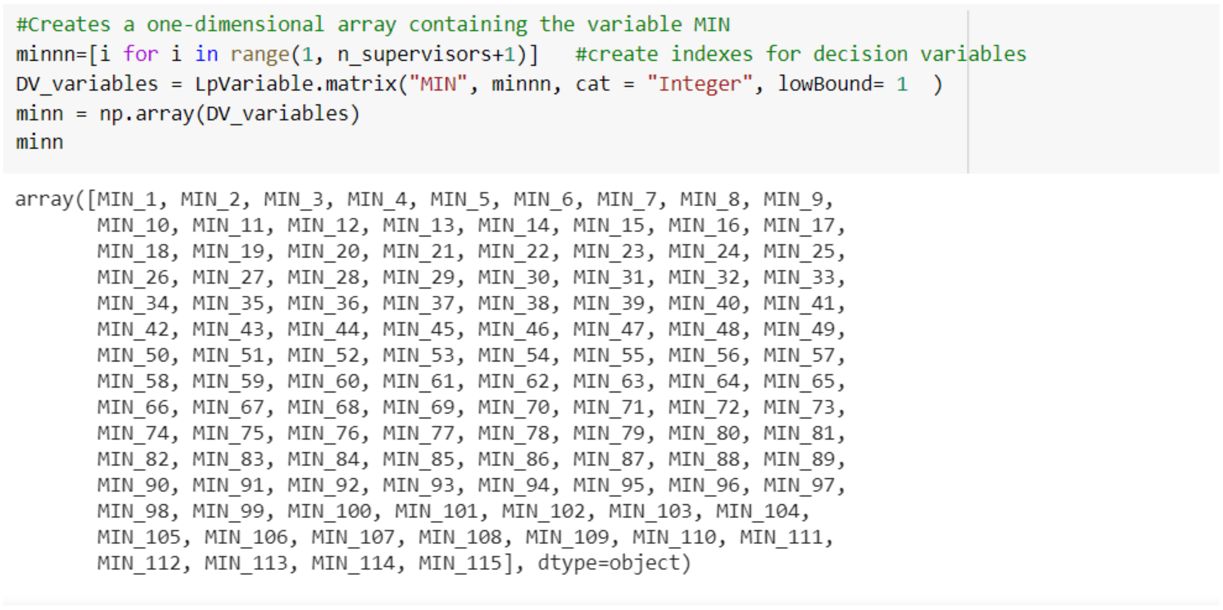
As a first step, we create metrics for the decision variables. We name our decision variables and use the above metrics as arguments to distinguish the decision variables.



```
#Creates a one-dimensional array containing the variable MAX
maxxx=[i for i in range(1, n_supervisors+1)]  #create indexes for decision variables
DV_variables = LpVariable.matrix("MAX", maxxx, cat = "Integer", lowBound= 0  )
maxx = np.array(DV_variables)
maxx
```

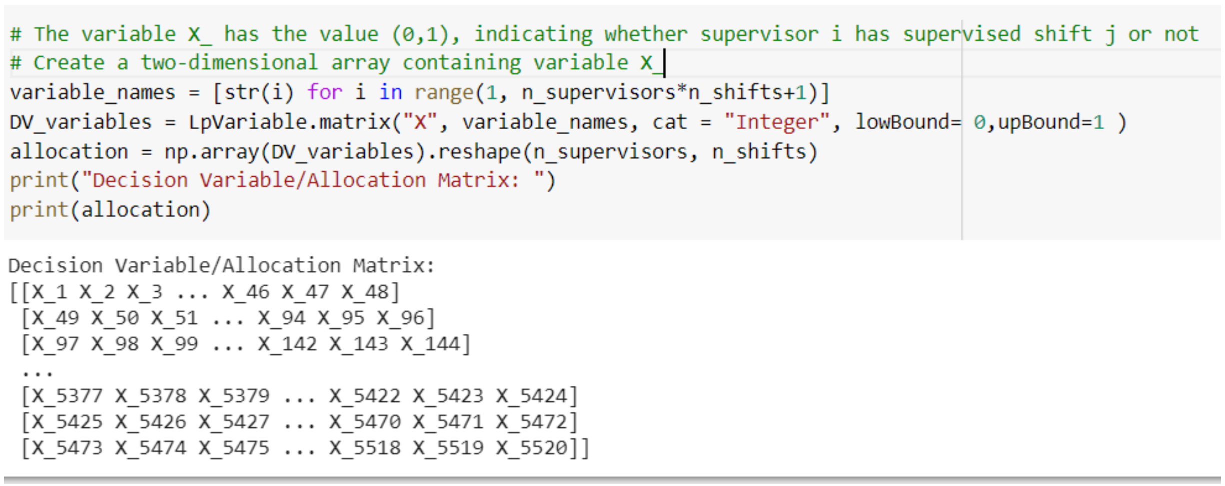
```
array([MAX_1, MAX_2, MAX_3, MAX_4, MAX_5, MAX_6, MAX_7, MAX_8, MAX_9,
       MAX_10, MAX_11, MAX_12, MAX_13, MAX_14, MAX_15, MAX_16, MAX_17,
       MAX_18, MAX_19, MAX_20, MAX_21, MAX_22, MAX_23, MAX_24, MAX_25,
       MAX_26, MAX_27, MAX_28, MAX_29, MAX_30, MAX_31, MAX_32, MAX_33,
       MAX_34, MAX_35, MAX_36, MAX_37, MAX_38, MAX_39, MAX_40, MAX_41,
       MAX_42, MAX_43, MAX_44, MAX_45, MAX_46, MAX_47, MAX_48, MAX_49,
       MAX_50, MAX_51, MAX_52, MAX_53, MAX_54, MAX_55, MAX_56, MAX_57,
       MAX_58, MAX_59, MAX_60, MAX_61, MAX_62, MAX_63, MAX_64, MAX_65,
       MAX_66, MAX_67, MAX_68, MAX_69, MAX_70, MAX_71, MAX_72, MAX_73,
       MAX_74, MAX_75, MAX_76, MAX_77, MAX_78, MAX_79, MAX_80, MAX_81,
       MAX_82, MAX_83, MAX_84, MAX_85, MAX_86, MAX_87, MAX_88, MAX_89,
       MAX_90, MAX_91, MAX_92, MAX_93, MAX_94, MAX_95, MAX_96, MAX_97,
       MAX_98, MAX_99, MAX_100, MAX_101, MAX_102, MAX_103, MAX_104,
       MAX_105, MAX_106, MAX_107, MAX_108, MAX_109, MAX_110, MAX_111,
       MAX_112, MAX_113, MAX_114, MAX_115], dtype=object)
```

Figure 3.12 Create a one-dimensional array containing the variable MAX


```
#Creates a one-dimensional array containing the variable MIN
minnn=[i for i in range(1, n_supervisors+1)]   #create indexes for decision variables
DV_variables = LpVariable.matrix("MIN", minnn, cat = "Integer", lowBound= 1  )
minn = np.array(DV_variables)
minn
```

```
array([MIN_1, MIN_2, MIN_3, MIN_4, MIN_5, MIN_6, MIN_7, MIN_8, MIN_9,
       MIN_10, MIN_11, MIN_12, MIN_13, MIN_14, MIN_15, MIN_16, MIN_17,
       MIN_18, MIN_19, MIN_20, MIN_21, MIN_22, MIN_23, MIN_24, MIN_25,
       MIN_26, MIN_27, MIN_28, MIN_29, MIN_30, MIN_31, MIN_32, MIN_33,
       MIN_34, MIN_35, MIN_36, MIN_37, MIN_38, MIN_39, MIN_40, MIN_41,
       MIN_42, MIN_43, MIN_44, MIN_45, MIN_46, MIN_47, MIN_48, MIN_49,
       MIN_50, MIN_51, MIN_52, MIN_53, MIN_54, MIN_55, MIN_56, MIN_57,
       MIN_58, MIN_59, MIN_60, MIN_61, MIN_62, MIN_63, MIN_64, MIN_65,
       MIN_66, MIN_67, MIN_68, MIN_69, MIN_70, MIN_71, MIN_72, MIN_73,
       MIN_74, MIN_75, MIN_76, MIN_77, MIN_78, MIN_79, MIN_80, MIN_81,
       MIN_82, MIN_83, MIN_84, MIN_85, MIN_86, MIN_87, MIN_88, MIN_89,
       MIN_90, MIN_91, MIN_92, MIN_93, MIN_94, MIN_95, MIN_96, MIN_97,
       MIN_98, MIN_99, MIN_100, MIN_101, MIN_102, MIN_103, MIN_104,
       MIN_105, MIN_106, MIN_107, MIN_108, MIN_109, MIN_110, MIN_111,
       MIN_112, MIN_113, MIN_114, MIN_115], dtype=object)
```

Figure 3.13 Create a one-dimensional array containing the variable MIN

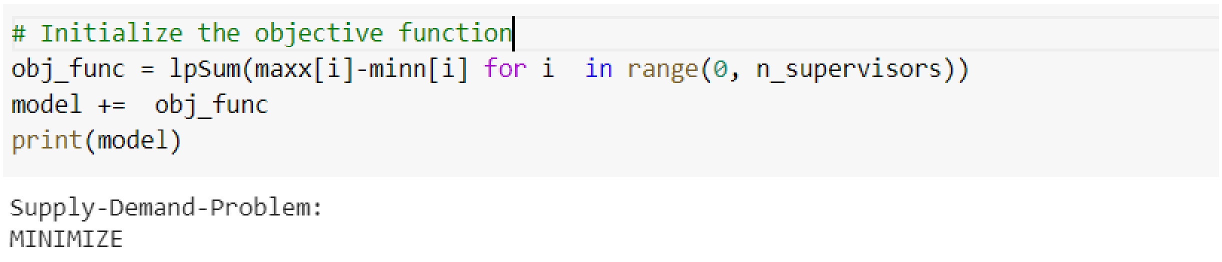

```
# The variable X_ has the value (0,1), indicating whether supervisor i has supervised shift j or not
# Create a two-dimensional array containing variable X_
variable_names = [str(i) for i in range(1, n_supervisors*n_shifts+1)]
DV_variables = LpVariable.matrix("X", variable_names, cat = "Integer", lowBound= 0,upBound=1 )
allocation = np.array(DV_variables).reshape(n_supervisors, n_shifts)
print("Decision Variable/Allocation Matrix: ")
print(allocation)
```

```
Decision Variable/Allocation Matrix:
[[X_1 X_2 X_3 ... X_46 X_47 X_48]
 [X_49 X_50 X_51 ... X_94 X_95 X_96]
 [X_97 X_98 X_99 ... X_142 X_143 X_144]
 ...
 [X_5377 X_5378 X_5379 ... X_5422 X_5423 X_5424]
 [X_5425 X_5426 X_5427 ... X_5470 X_5471 X_5472]
 [X_5473 X_5474 X_5475 ... X_5518 X_5519 X_5520]]
```

Figure 3.14 Create a two-dimensional array containing variable X

Objective function initialization:

lpSum is used alternatively with **sum** function in Python because it is much faster while performing operations with **PuLP** variables and also summarizes the variables well. We further add the objective function to the model using the += shorthand operator.



```
# Initialize the objective function
obj_func = lpSum(maxx[i]-minn[i] for i  in range(0, n_supervisors))
model +=  obj_func
print(model)
```

```
Supply-Demand-Problem:
MINIMIZE
```

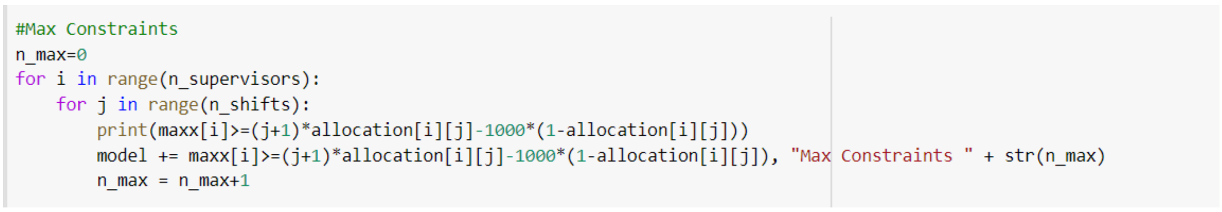
Figure 3.15 Objective function

The target function tries to minimize the cost of the distance from the largest to the smallest.

Create constraints:

- The constraint to find the last position with value "1" in each row

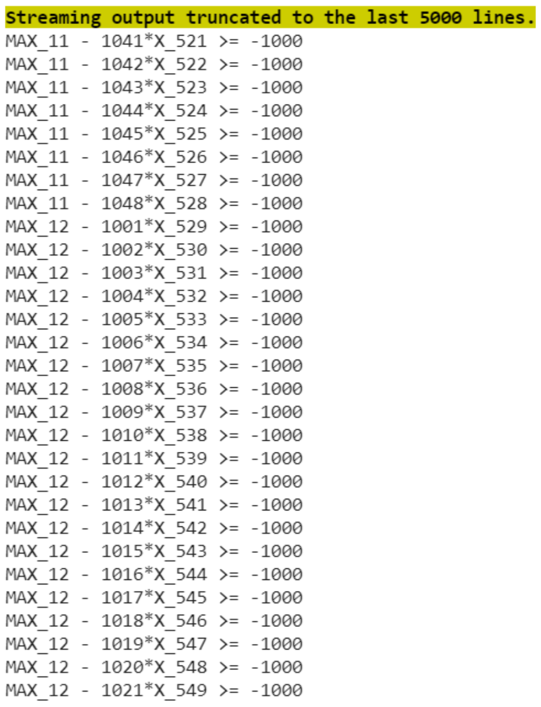
  In the case of this input we set the value of a = 1000



```
#Max Constraints
n_max=0
for i in range(n_supervisors):
    for j in range(n_shifts):
        print(maxx[i]>=(j+1)*allocation[i][j]-1000*(1-allocation[i][j]))
        model += maxx[i]>=(j+1)*allocation[i][j]-1000*(1-allocation[i][j]), "Max Constraints " + str(n_max)
        n_max = n_max+1
```

Figure 3.16 Max constraints

- Output:

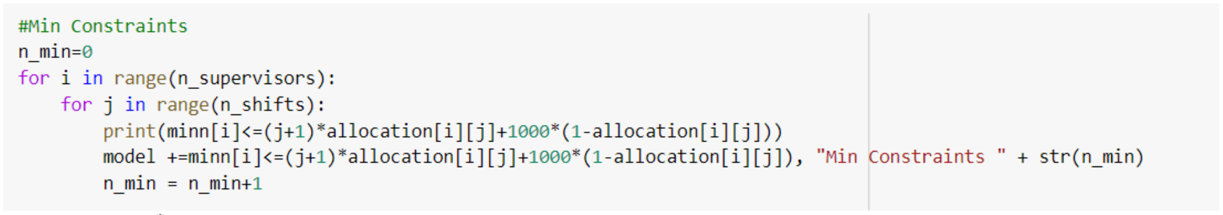


```
Streaming output truncated to the last 5000 lines.
MAX_11 - 1041*X_521 >= -1000
MAX_11 - 1042*X_522 >= -1000
MAX_11 - 1043*X_523 >= -1000
MAX_11 - 1044*X_524 >= -1000
MAX_11 - 1045*X_525 >= -1000
MAX_11 - 1046*X_526 >= -1000
MAX_11 - 1047*X_527 >= -1000
MAX_11 - 1048*X_528 >= -1000
MAX_12 - 1001*X_529 >= -1000
MAX_12 - 1002*X_530 >= -1000
MAX_12 - 1003*X_531 >= -1000
MAX_12 - 1004*X_532 >= -1000
MAX_12 - 1005*X_533 >= -1000
MAX_12 - 1006*X_534 >= -1000
MAX_12 - 1007*X_535 >= -1000
MAX_12 - 1008*X_536 >= -1000
MAX_12 - 1009*X_537 >= -1000
MAX_12 - 1010*X_538 >= -1000
MAX_12 - 1011*X_539 >= -1000
MAX_12 - 1012*X_540 >= -1000
MAX_12 - 1013*X_541 >= -1000
MAX_12 - 1014*X_542 >= -1000
MAX_12 - 1015*X_543 >= -1000
MAX_12 - 1016*X_544 >= -1000
MAX_12 - 1017*X_545 >= -1000
MAX_12 - 1018*X_546 >= -1000
MAX_12 - 1019*X_547 >= -1000
MAX_12 - 1020*X_548 >= -1000
MAX_12 - 1021*X_549 >= -1000
```

Figure 3.17 Output of max constraints

- Constraint to find the first position with value "1" in each row

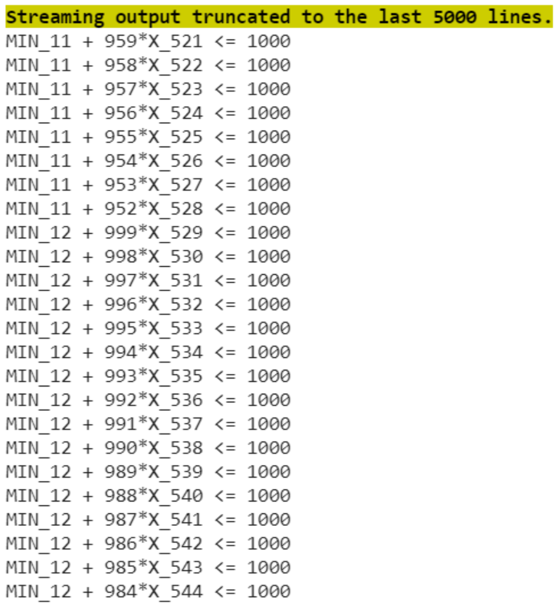
  In the case of this input we set the value of a = 1000



```
#Min Constraints
n_min=0
for i in range(n_supervisors):
    for j in range(n_shifts):
        print(minn[i]<=(j+1)*allocation[i][j]+1000*(1-allocation[i][j]))
        model +=minn[i]<=(j+1)*allocation[i][j]+1000*(1-allocation[i][j]), "Min Constraints " + str(n_min)
        n_min = n_min+1
```

Figure 3.18 Min constraints

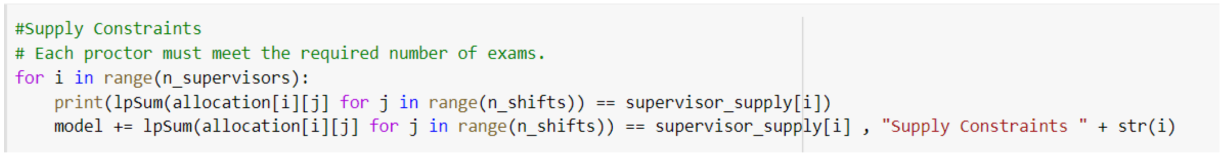
- Output:



```
Streaming output truncated to the last 5000 lines.
MIN_11 + 959*X_521 <= 1000
MIN_11 + 958*X_522 <= 1000
MIN_11 + 957*X_523 <= 1000
MIN_11 + 956*X_524 <= 1000
MIN_11 + 955*X_525 <= 1000
MIN_11 + 954*X_526 <= 1000
MIN_11 + 953*X_527 <= 1000
MIN_11 + 952*X_528 <= 1000
MIN_12 + 999*X_529 <= 1000
MIN_12 + 998*X_530 <= 1000
MIN_12 + 997*X_531 <= 1000
MIN_12 + 996*X_532 <= 1000
MIN_12 + 995*X_533 <= 1000
MIN_12 + 994*X_534 <= 1000
MIN_12 + 993*X_535 <= 1000
MIN_12 + 992*X_536 <= 1000
MIN_12 + 991*X_537 <= 1000
MIN_12 + 990*X_538 <= 1000
MIN_12 + 989*X_539 <= 1000
MIN_12 + 988*X_540 <= 1000
MIN_12 + 987*X_541 <= 1000
MIN_12 + 986*X_542 <= 1000
MIN_12 + 985*X_543 <= 1000
MIN_12 + 984*X_544 <= 1000
```

Figure 3.19 output of min constraint

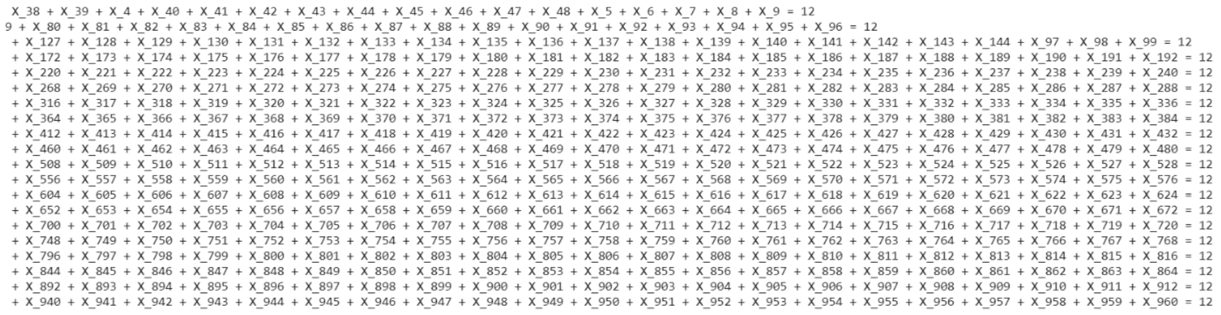
- The constraint on calculating the validity of X satisfies the constraint (1)



```
#Supply Constraints
# Each proctor must meet the required number of exams.
for i in range(n_supervisors):
    print(lpSum(allocation[i][j] for j in range(n_shifts)) == supervisor_supply[i])
    model += lpSum(allocation[i][j] for j in range(n_shifts)) == supervisor_supply[i] , "Supply Constraints " + str(i)
```

Figure 3.20 Supply constraints

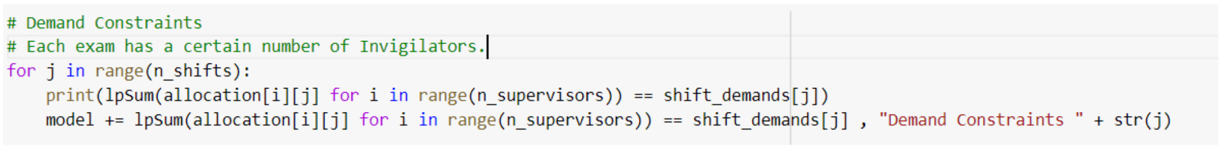
- Output:



```
 X_38 + X_39 + X_4 + X_40 + X_41 + X_42 + X_43 + X_44 + X_45 + X_46 + X_47 + X_48 + X_5 + X_6 + X_7 + X_8 + X_9 = 12
9 + X_80 + X_81 + X_82 + X_83 + X_84 + X_85 + X_86 + X_87 + X_88 + X_89 + X_90 + X_91 + X_92 + X_93 + X_94 + X_95 + X_96 = 12
 + X_127 + X_128 + X_129 + X_130 + X_131 + X_132 + X_133 + X_134 + X_135 + X_136 + X_137 + X_138 + X_139 + X_140 + X_141 + X_142 + X_143 + X_144 + X_97 + X_98 + X_99 = 12
 + X_172 + X_173 + X_174 + X_175 + X_176 + X_177 + X_178 + X_179 + X_180 + X_181 + X_182 + X_183 + X_184 + X_185 + X_186 + X_187 + X_188 + X_189 + X_190 + X_191 + X_192 = 12
 + X_220 + X_221 + X_222 + X_223 + X_224 + X_225 + X_226 + X_227 + X_228 + X_229 + X_230 + X_231 + X_232 + X_233 + X_234 + X_235 + X_236 + X_237 + X_238 + X_239 + X_240 = 12
 + X_268 + X_269 + X_270 + X_271 + X_272 + X_273 + X_274 + X_275 + X_276 + X_277 + X_278 + X_279 + X_280 + X_281 + X_282 + X_283 + X_284 + X_285 + X_286 + X_287 + X_288 = 12
 + X_316 + X_317 + X_318 + X_319 + X_320 + X_321 + X_322 + X_323 + X_324 + X_325 + X_326 + X_327 + X_328 + X_329 + X_330 + X_331 + X_332 + X_333 + X_334 + X_335 + X_336 = 12
 + X_364 + X_365 + X_366 + X_367 + X_368 + X_369 + X_370 + X_371 + X_372 + X_373 + X_374 + X_375 + X_376 + X_377 + X_378 + X_379 + X_380 + X_381 + X_382 + X_383 + X_384 = 12
 + X_412 + X_413 + X_414 + X_415 + X_416 + X_417 + X_418 + X_419 + X_420 + X_421 + X_422 + X_423 + X_424 + X_425 + X_426 + X_427 + X_428 + X_429 + X_430 + X_431 + X_432 = 12
 + X_460 + X_461 + X_462 + X_463 + X_464 + X_465 + X_466 + X_467 + X_468 + X_469 + X_470 + X_471 + X_472 + X_473 + X_474 + X_475 + X_476 + X_477 + X_478 + X_479 + X_480 = 12
 + X_508 + X_509 + X_510 + X_511 + X_512 + X_513 + X_514 + X_515 + X_516 + X_517 + X_518 + X_519 + X_520 + X_521 + X_522 + X_523 + X_524 + X_525 + X_526 + X_527 + X_528 = 12
 + X_556 + X_557 + X_558 + X_559 + X_560 + X_561 + X_562 + X_563 + X_564 + X_565 + X_566 + X_567 + X_568 + X_569 + X_570 + X_571 + X_572 + X_573 + X_574 + X_575 + X_576 = 12
 + X_604 + X_605 + X_606 + X_607 + X_608 + X_609 + X_610 + X_611 + X_612 + X_613 + X_614 + X_615 + X_616 + X_617 + X_618 + X_619 + X_620 + X_621 + X_622 + X_623 + X_624 = 12
 + X_652 + X_653 + X_654 + X_655 + X_656 + X_657 + X_658 + X_659 + X_660 + X_661 + X_662 + X_663 + X_664 + X_665 + X_666 + X_667 + X_668 + X_669 + X_670 + X_671 + X_672 = 12
 + X_700 + X_701 + X_702 + X_703 + X_704 + X_705 + X_706 + X_707 + X_708 + X_709 + X_710 + X_711 + X_712 + X_713 + X_714 + X_715 + X_716 + X_717 + X_718 + X_719 + X_720 = 12
 + X_748 + X_749 + X_750 + X_751 + X_752 + X_753 + X_754 + X_755 + X_756 + X_757 + X_758 + X_759 + X_760 + X_761 + X_762 + X_763 + X_764 + X_765 + X_766 + X_767 + X_768 = 12
 + X_796 + X_797 + X_798 + X_799 + X_800 + X_801 + X_802 + X_803 + X_804 + X_805 + X_806 + X_807 + X_808 + X_809 + X_810 + X_811 + X_812 + X_813 + X_814 + X_815 + X_816 = 12
 + X_844 + X_845 + X_846 + X_847 + X_848 + X_849 + X_850 + X_851 + X_852 + X_853 + X_854 + X_855 + X_856 + X_857 + X_858 + X_859 + X_860 + X_861 + X_862 + X_863 + X_864 = 12
 + X_892 + X_893 + X_894 + X_895 + X_896 + X_897 + X_898 + X_899 + X_900 + X_901 + X_902 + X_903 + X_904 + X_905 + X_906 + X_907 + X_908 + X_909 + X_910 + X_911 + X_912 = 12
 + X_940 + X_941 + X_942 + X_943 + X_944 + X_945 + X_946 + X_947 + X_948 + X_949 + X_950 + X_951 + X_952 + X_953 + X_954 + X_955 + X_956 + X_957 + X_958 + X_959 + X_960 = 12
```

Figure 3.21 output of supply constraint

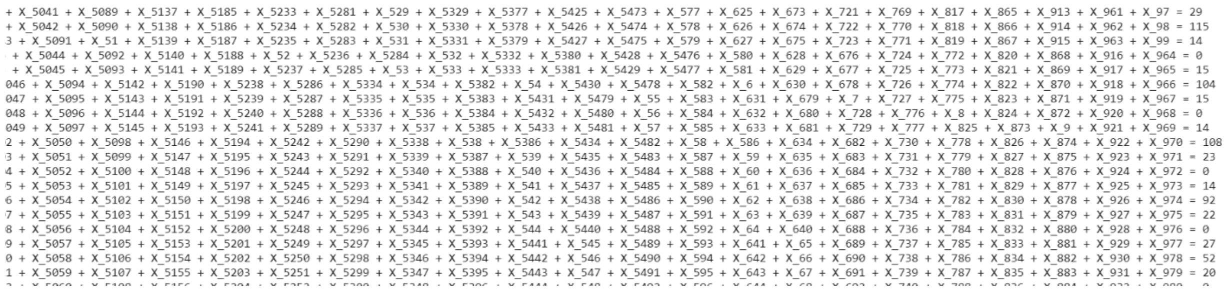
- The constraint on calculating the validity of X satisfies the constraint (2)



```
# Demand Constraints
# Each exam has a certain number of Invigilators.
for j in range(n_shifts):
    print(lpSum(allocation[i][j] for i in range(n_supervisors)) == shift_demands[j])
    model += lpSum(allocation[i][j] for i in range(n_supervisors)) == shift_demands[j] , "Demand Constraints " + str(j)
```

Figure 3.22 demand constraints

- Output:



```
+ X_5041 + X_5089 + X_5137 + X_5185 + X_5233 + X_5281 + X_529 + X_5329 + X_5377 + X_5425 + X_5473 + X_577 + X_625 + X_673 + X_721 + X_769 + X_817 + X_865 + X_913 + X_961 + X_97 = 29
+ X_5042 + X_5090 + X_5138 + X_5186 + X_5234 + X_5282 + X_530 + X_5330 + X_5378 + X_5426 + X_5474 + X_578 + X_626 + X_674 + X_722 + X_770 + X_818 + X_866 + X_914 + X_962 + X_98 = 115
3 + X_5091 + X_51 + X_5139 + X_5187 + X_5235 + X_5283 + X_531 + X_5331 + X_5379 + X_5427 + X_5475 + X_579 + X_627 + X_675 + X_723 + X_771 + X_819 + X_867 + X_915 + X_963 + X_99 = 14
 + X_5044 + X_5092 + X_5140 + X_5188 + X_52 + X_5236 + X_5284 + X_532 + X_5332 + X_5380 + X_5428 + X_5476 + X_580 + X_628 + X_676 + X_724 + X_772 + X_820 + X_868 + X_916 + X_964 = 0
 + X_5045 + X_5093 + X_5141 + X_5189 + X_5237 + X_5285 + X_53 + X_533 + X_5333 + X_5381 + X_5429 + X_5477 + X_581 + X_629 + X_677 + X_725 + X_773 + X_821 + X_869 + X_917 + X_965 = 15
046 + X_5094 + X_5142 + X_5190 + X_5238 + X_5286 + X_5334 + X_534 + X_5382 + X_54 + X_5430 + X_5478 + X_582 + X_6 + X_630 + X_678 + X_726 + X_774 + X_822 + X_870 + X_918 + X_966 = 104
047 + X_5095 + X_5143 + X_5191 + X_5239 + X_5287 + X_5335 + X_535 + X_5383 + X_5431 + X_5479 + X_55 + X_583 + X_631 + X_679 + X_7 + X_727 + X_775 + X_823 + X_871 + X_919 + X_967 = 15
048 + X_5096 + X_5144 + X_5192 + X_5240 + X_5288 + X_5336 + X_536 + X_5384 + X_5432 + X_5480 + X_56 + X_584 + X_632 + X_680 + X_728 + X_776 + X_8 + X_824 + X_872 + X_920 + X_968 = 0
049 + X_5097 + X_5145 + X_5193 + X_5241 + X_5289 + X_5337 + X_537 + X_5385 + X_5433 + X_5481 + X_57 + X_585 + X_633 + X_681 + X_729 + X_777 + X_825 + X_873 + X_9 + X_921 + X_969 = 14
2 + X_5050 + X_5098 + X_5146 + X_5194 + X_5242 + X_5290 + X_5338 + X_538 + X_5386 + X_5434 + X_5482 + X_58 + X_586 + X_634 + X_682 + X_730 + X_778 + X_826 + X_874 + X_922 + X_970 = 108
3 + X_5051 + X_5099 + X_5147 + X_5195 + X_5243 + X_5291 + X_5339 + X_5387 + X_539 + X_5435 + X_5483 + X_587 + X_59 + X_635 + X_683 + X_731 + X_779 + X_827 + X_875 + X_923 + X_971 = 23
4 + X_5052 + X_5100 + X_5148 + X_5196 + X_5244 + X_5292 + X_5340 + X_5388 + X_540 + X_5436 + X_5484 + X_588 + X_60 + X_636 + X_684 + X_732 + X_780 + X_828 + X_876 + X_924 + X_972 = 0
5 + X_5053 + X_5101 + X_5149 + X_5197 + X_5245 + X_5293 + X_5341 + X_5389 + X_541 + X_5437 + X_5485 + X_589 + X_61 + X_637 + X_685 + X_733 + X_781 + X_829 + X_877 + X_925 + X_973 = 14
6 + X_5054 + X_5102 + X_5150 + X_5198 + X_5246 + X_5294 + X_5342 + X_5390 + X_542 + X_5438 + X_5486 + X_590 + X_62 + X_638 + X_686 + X_734 + X_782 + X_830 + X_878 + X_926 + X_974 = 92
7 + X_5055 + X_5103 + X_5151 + X_5199 + X_5247 + X_5295 + X_5343 + X_5391 + X_543 + X_5439 + X_5487 + X_591 + X_63 + X_639 + X_687 + X_735 + X_783 + X_831 + X_879 + X_927 + X_975 = 22
8 + X_5056 + X_5104 + X_5152 + X_5200 + X_5248 + X_5296 + X_5344 + X_5392 + X_544 + X_5440 + X_5488 + X_592 + X_64 + X_640 + X_688 + X_736 + X_784 + X_832 + X_880 + X_928 + X_976 = 0
9 + X_5057 + X_5105 + X_5153 + X_5201 + X_5249 + X_5297 + X_5345 + X_5393 + X_5441 + X_545 + X_5489 + X_593 + X_641 + X_65 + X_689 + X_737 + X_785 + X_833 + X_881 + X_929 + X_977 = 27
0 + X_5058 + X_5106 + X_5154 + X_5202 + X_5250 + X_5298 + X_5346 + X_5394 + X_5442 + X_546 + X_5490 + X_594 + X_642 + X_66 + X_690 + X_738 + X_786 + X_834 + X_882 + X_930 + X_978 = 52
1 + X_5059 + X_5107 + X_5155 + X_5203 + X_5251 + X_5299 + X_5347 + X_5395 + X_5443 + X_547 + X_5491 + X_595 + X_643 + X_67 + X_691 + X_739 + X_787 + X_835 + X_883 + X_931 + X_979 = 20
```

Figure 3.23 output of demand constraints

Complete the optimal model

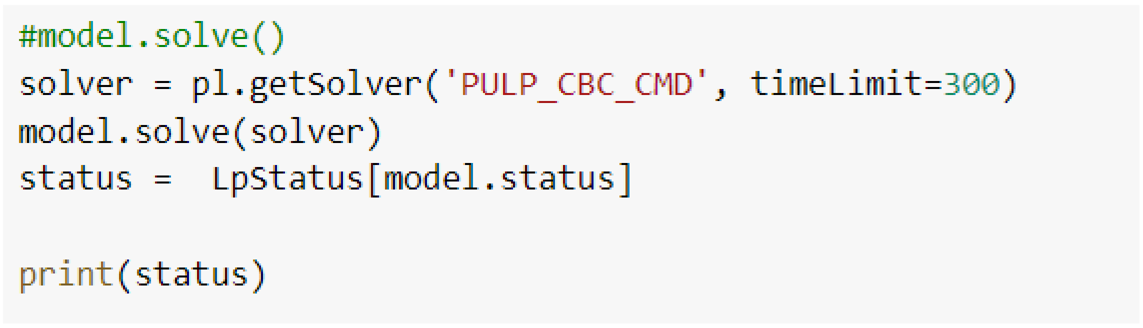
```
model.writeLP("Supply_demand_prob.lp")
```

Save the model in a file named "Supply_demand_prob.lp" so that the model can be viewed and tested easily.

Use solver to find the optimal solution for the model

Because the problem has very large input data, in order to find the solution we must set the solver runtime. Here we use pulp.getSolver (Solver_name, timeLimit):

- Solver_name: is the name of the solver that you want to use
- timeLimit: the possible time at which the solver can give a solution that satisfies the constraints.

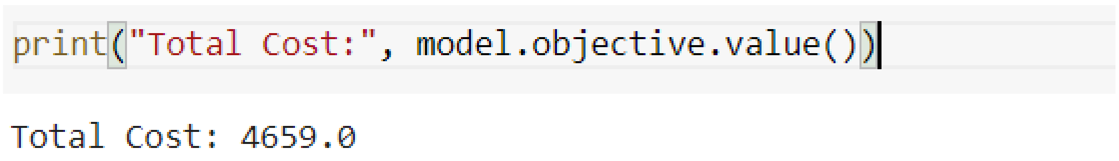


```python
#model.solve()
solver = pl.getSolver('PULP_CBC_CMD', timeLimit=300)
model.solve(solver)
status =  LpStatus[model.status]

print(status)
```

Figure 3.24 Solver code

Here are the results calculated for the objective function:

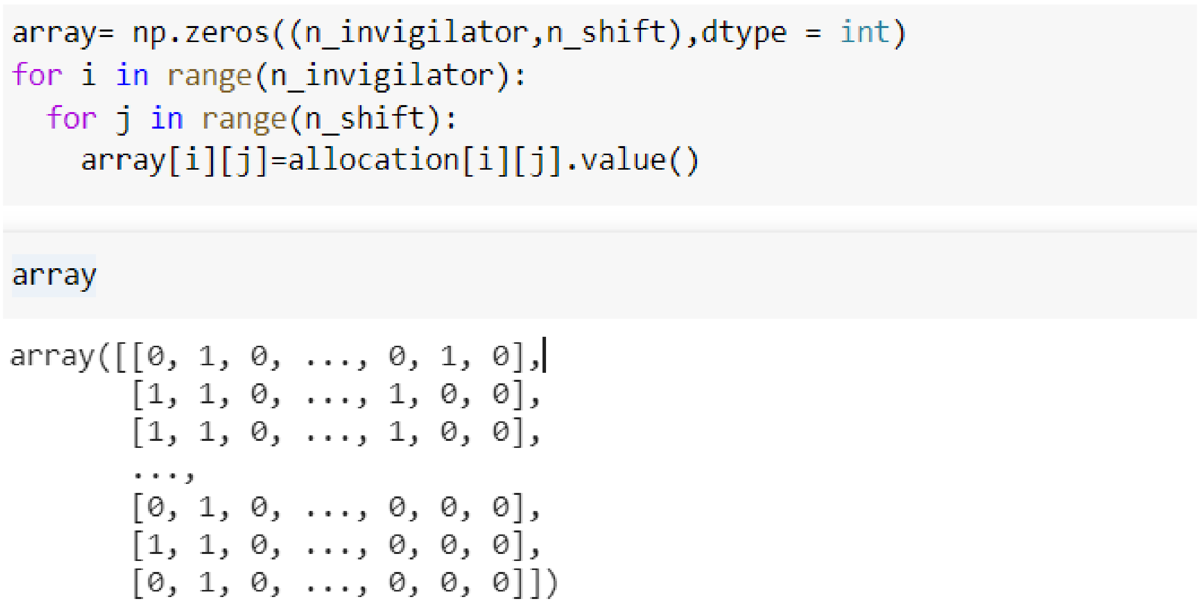


```python
print("Total Cost:", model.objective.value())
```

```
Total Cost: 4659.0
```

Figure 3.25 Calculated result

We can output the array of variables X:



```python
array= np.zeros((n_invigilator,n_shift),dtype = int)
for i in range(n_invigilator):
  for j in range(n_shift):
    array[i][j]=allocation[i][j].value()
```

```python
array
```

```
array([[0, 1, 0, ..., 0, 1, 0],
       [1, 1, 0, ..., 1, 0, 0],
       [1, 1, 0, ..., 1, 0, 0],
       ...,
       [0, 1, 0, ..., 0, 0, 0],
       [1, 1, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0]])
```

Figure 3.26 output array of variables X

3.5 Result comparison

We used the fitness score to evaluate the quality the 2 methods (Genetic algorithm and Linear programming). The result is presented in the table below.

Method	Fitness score
Genetic algorithm (GA)	11311
Linear programming (LP)	11458

Table 3.3 the results of 2 methods

The integer programming model obtained the best result on fitness score with limited running time to 235s.

At the same time limited, which is 235 seconds, we can get the fitness results of two methods. Then we can calculate the number of days that each invigilator must go to school to work.

Below is the days at school distribution chart of invigilators using the two methods mentioned above:

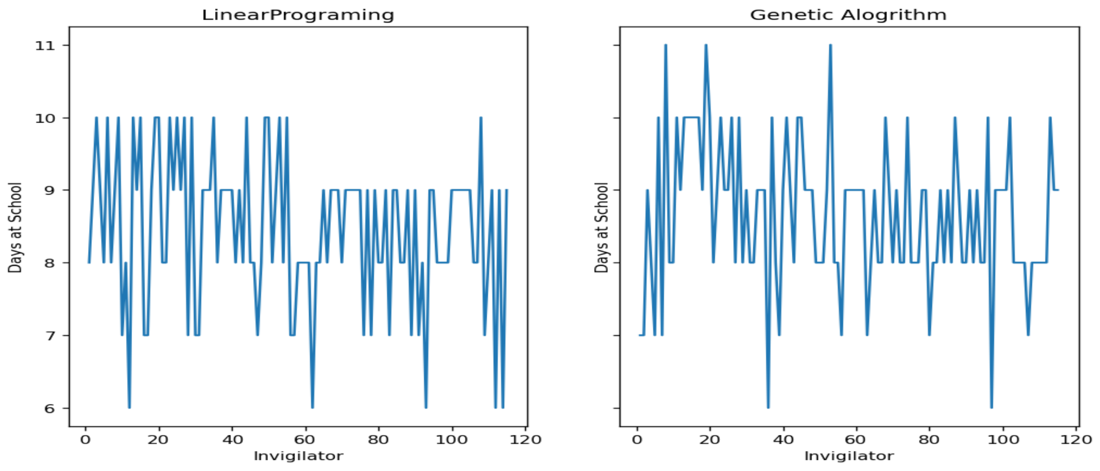


Figure 3.27 comparison between the days at school of invigilators of 2 methods

Overall, it can be seen that the days at school distribution chart for invigilator in the figure for Linear Programming is quite even, from 7 to 10 days. Although the Genetic Algorithm has a better fitness score, there are some invigilators who have to be at school 11 days.

The time for genetic algorithm to calculate the results is quite fast because it uses the exhaust algorithm to create the correct solution. In a short period of time, Genetic Algorithm gives somewhat better results, but if we give Linear Programming more time to calculate, the fitness score will be 8162 while Genetic Algorithm only calculates up to the value 11275 at best.

Below is the days at school distribution chart of invigilators with time limited to 3000 seconds (50 minutes):

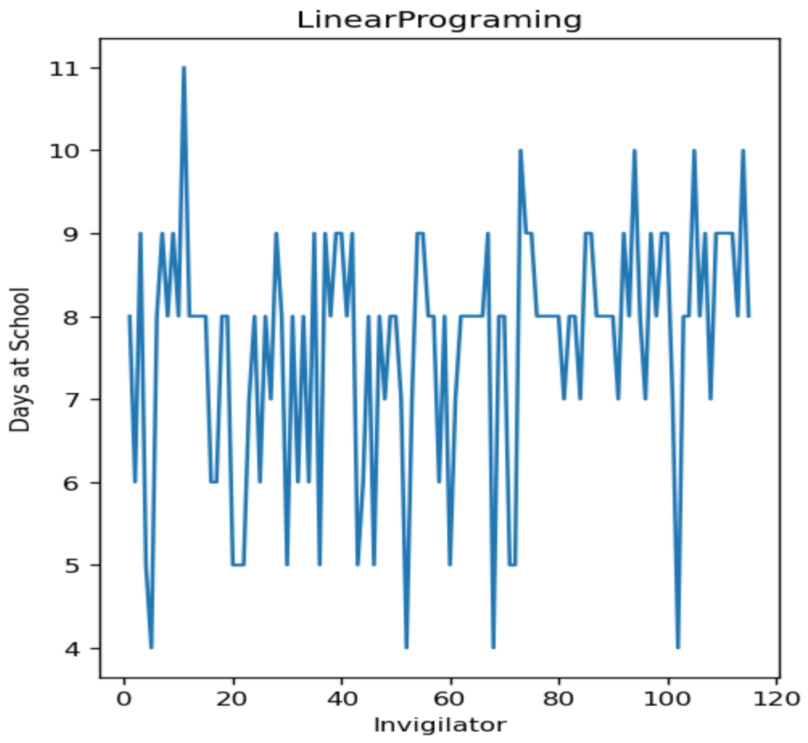


Figure 3.28 the days at school distribution chart of invigilator for LP, with more time limited

It can be seen that the days at school of invigilator have been significantly reduced. Instead of a minimum of 6 days, now only 4 days are needed, and the maximum allocation is 5 to 9 days.

Hence, from the results obtained and the time limits, we can see that the 2 methods have their strengths and weaknesses. Linear programming can give much better results if you give it time to calculate. As for Genetic Algorithm, it uses random to find the solution so it can find the solution quickly, but in terms of optimization, it is not as good as Linear Programming.

Chapter 4: APPLICATION ANALYSIS AND IMPLEMENTATION

4.1 Requirement analysis

4.1.1 Functional requirements

Some main functions of the application:

- Import invigilator dataset.
- Import exam schedule dataset.
- View imported data (invigilator, Courses) and final output Schedule.
- Run algorithm (GA execution)

4.1.2 Non-functional requirements

- Friendly UI, easy to use.
- Can be easily maintain or add more features.
- Application work perfectly, with an acceptable output.

4.1.3 Storage requirements

The application will store:

- Invigilators information
- Courses information
- Invigilators roster

4.2 Use case diagram

4.2.1 General use case diagram

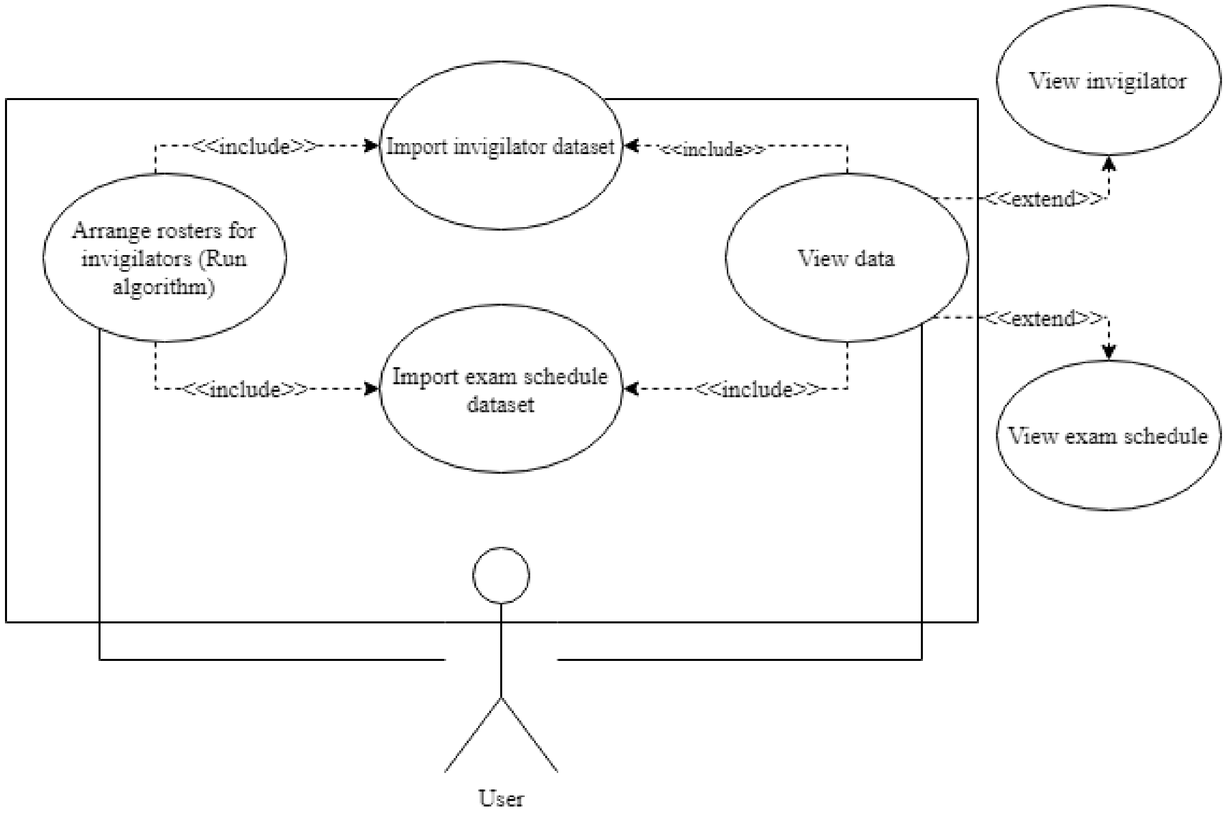


Figure 4.1 General use case diagram

4.2.2 List of use cases

No.	**Use case name**	**Meaning**
1	Import invigilator dataset	Users import invigilator dataset to database.
2	Import exam schedule dataset	Users import exam schedule dataset to database.
3	Arrange rosters for invigilator	Run genetic algorithm to schedule rosters for invigilator.
4	View data	Users can view all of the imported data.
5	View invigilator	Users can view invigilator information and they can select specific invigilator to inspect.
6	View exam schedule	Users can view exam schedule that has been imported.

Table 4.1 List of use case table

4.3 Main activity diagram

4.3.1 Import invigilator dataset

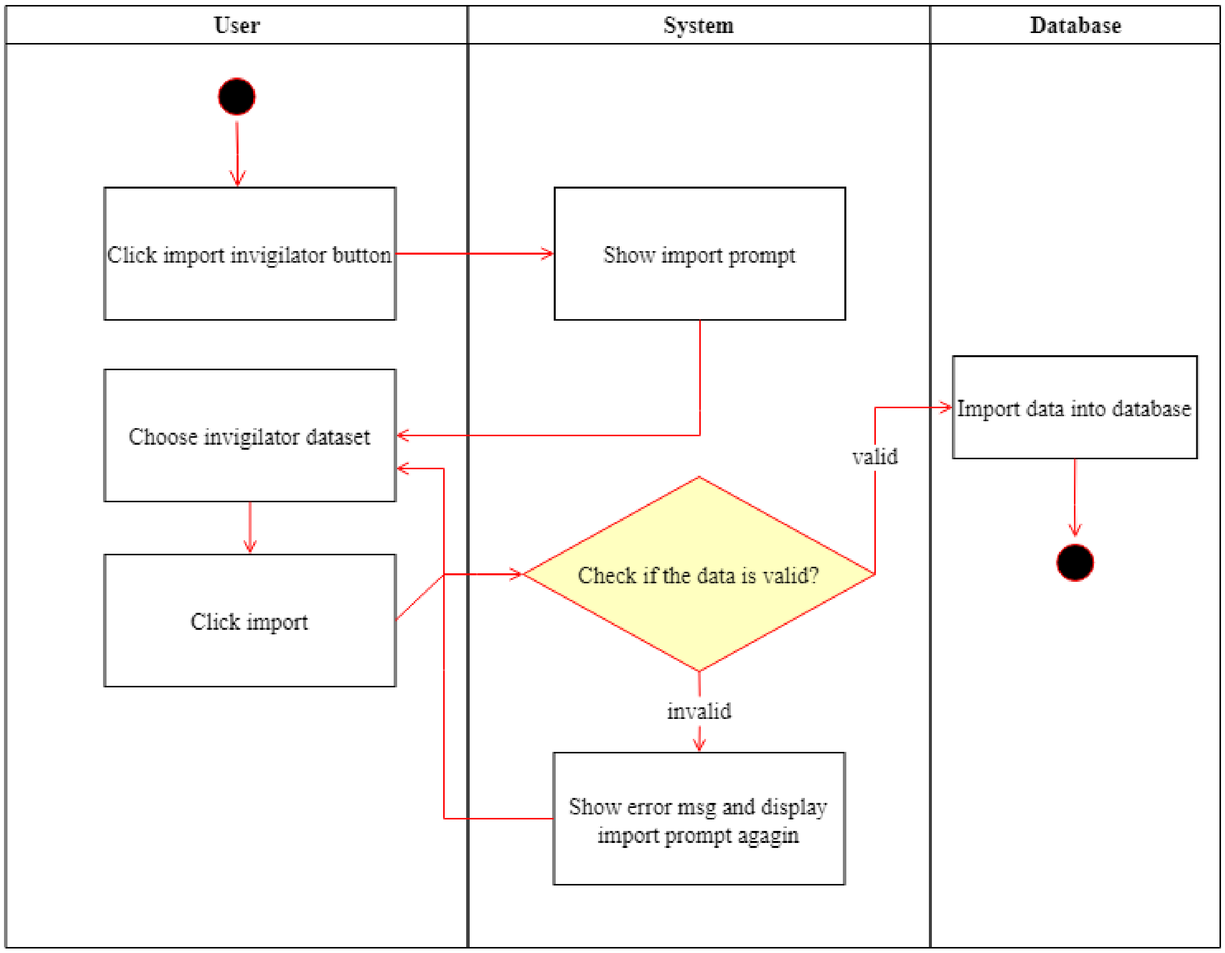
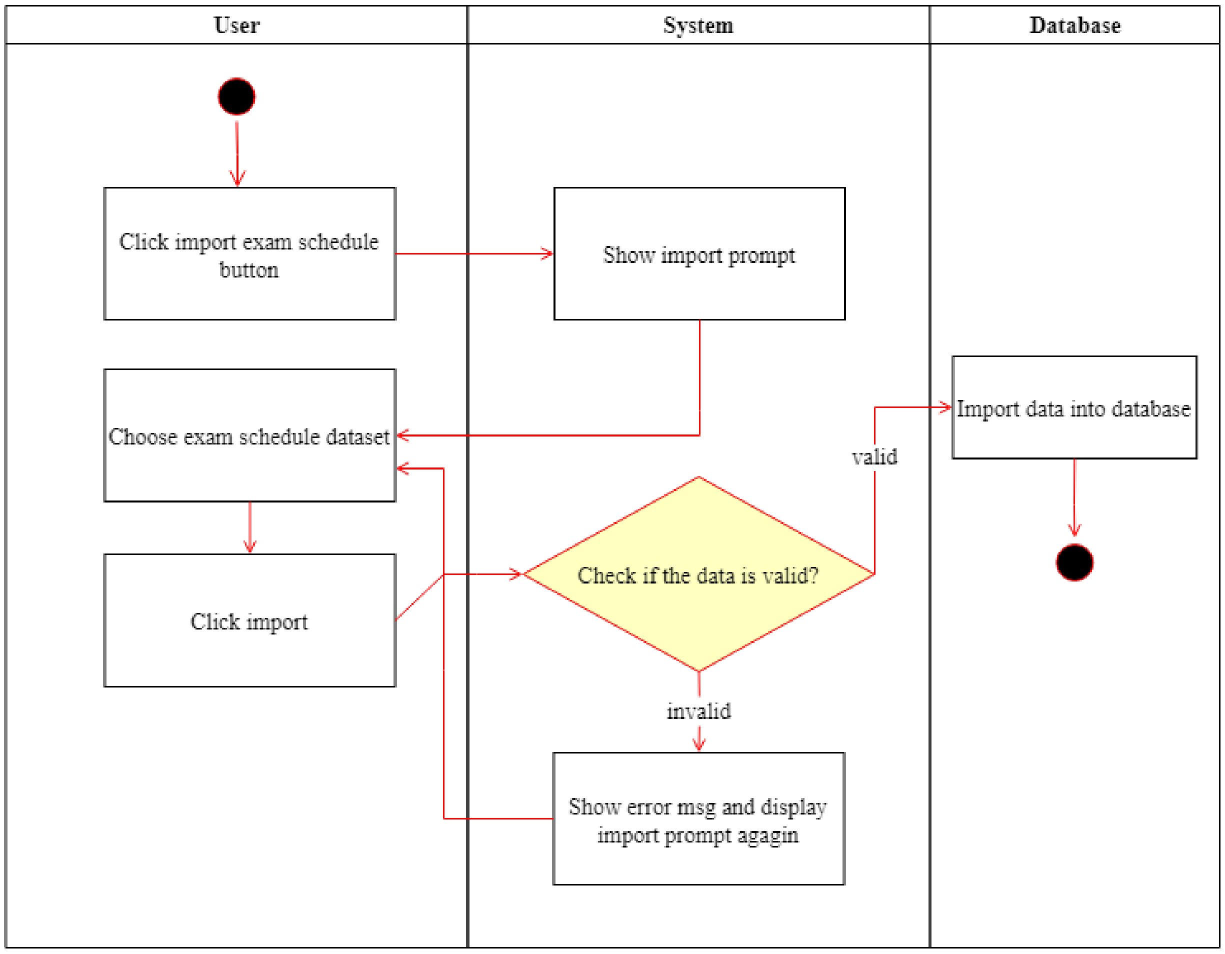

User
System
Database
Click import invigilator button
Show import prompt
Choose invigilator dataset
Import data into database
valid
Check if the data is valid?
Click import
invalid
Show error msg and display import prompt agagin

Figure 4.2 Import invigilator dataset activity diagram

4.3.2 Import exam schedule dataset



User
System
Database
Click import exam schedule button
Show import prompt
Choose exam schedule dataset
Import data into database
valid
Check if the data is valid?
Click import
invalid
Show error msg and display import prompt agagin

Figure 4.3 Import exam schedule dataset activity diagram

4.3.3 Arrange roster for invigilator

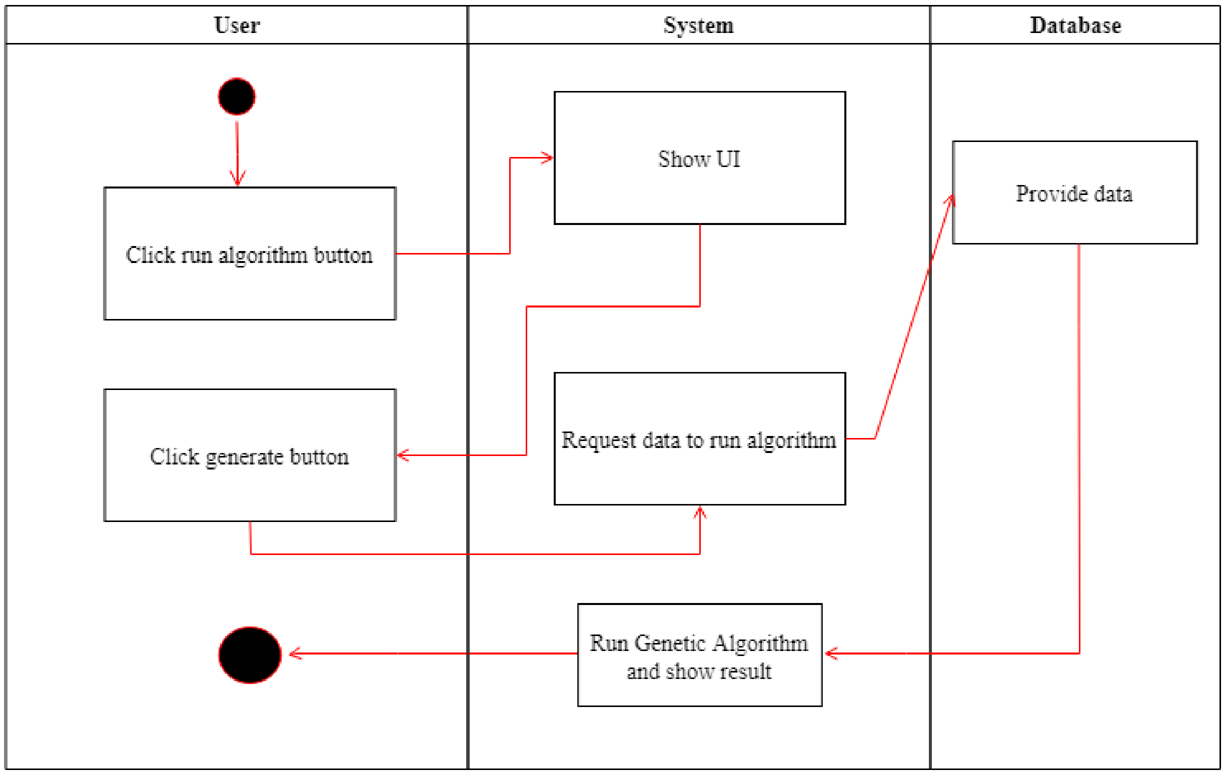

User
System
Database
Show UI
Provide data
Click run algorithm button
Request data to run algorithm
Click generate button
Run Genetic Algorithm and show result

Figure 4.4 Arrange roster for invigilator activity diagram

4.3.4 View data

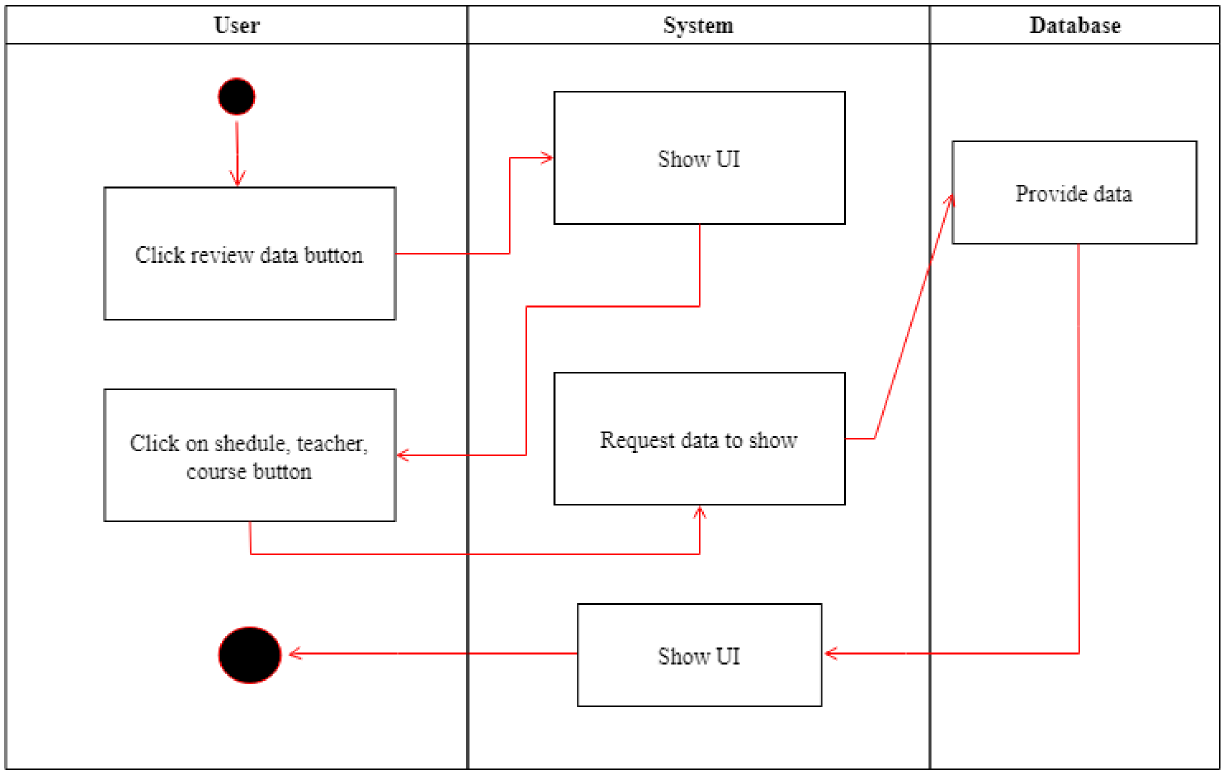


Figure 4.5 View data activity diagram

4.4 Database design

4.4.1 Entity relationship diagram

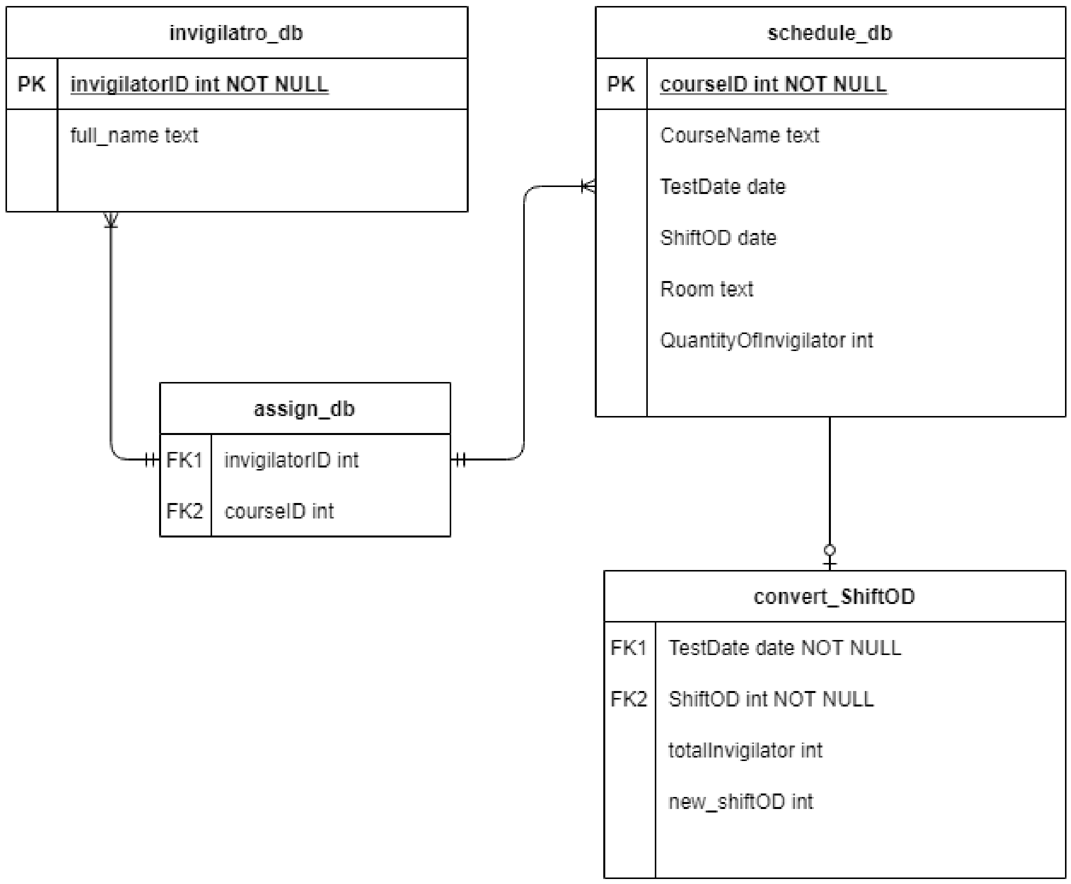


Figure 4.6 Entity relationship diagram

4.4.2 Tables description

4.4.2.1 invigilator_db table

No.	**Name**	**Data type**	**Description**
1	invigilatorID	INT	ID of invigilators.
2	Full_Name	TEXT	Full name of the invigilators.

Table 4.2 invigilator_db description table

4.4.2.2 schedule_db table

No.	**Name**	**Data type**	**Description**
1	courseID	INT	ID of the course
2	CourseName	TEXT	Name of the course
3	TestDate	DATE	Day that the test occurs
4	ShiftOD	INT	Shift of the test day
5	Room	TEXT	Room name
6	QuantityOfInvigilator	INT	Number of invigilators

Table 4.3 schedule_db description table

4.4.2.3 assign_db table

No.	Name	Data type	Description
1	invigilatorID	INT	ID of invigilator
2	courseID	INT	ID of the course

Table 4.4 assign_db description table

4.4.2.4 convert_ShiftOD table

No.	Name	Data type	Description
1	TestDate	DATE	Day that the test occurs
2	ShiftOD	INT	Shift of the test day
3	totalOfInvigilator	INT	Total of inviglator for a day
4	new_ShiftOD	INT	Copied of ShiftOD

Table 4.5 covert_ShiftOD description table

4.5 Application implementation

Based on the results obtained from previous chapters, we have successfully applied and implemented genetic algorithms to create a graphic user interface application that we named "Invigilator roster generator".

4.5.1 Application components

- Python: mostly used for coding
- SQLite: is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.
- PyCharm Community Edition: is an integrated development environment (IDE) used in computer programming, specifically for the Python language.
- Jupyter Notebook: is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

4.5.2 Graphic User Interface

4.5.2.1 Welcome UI

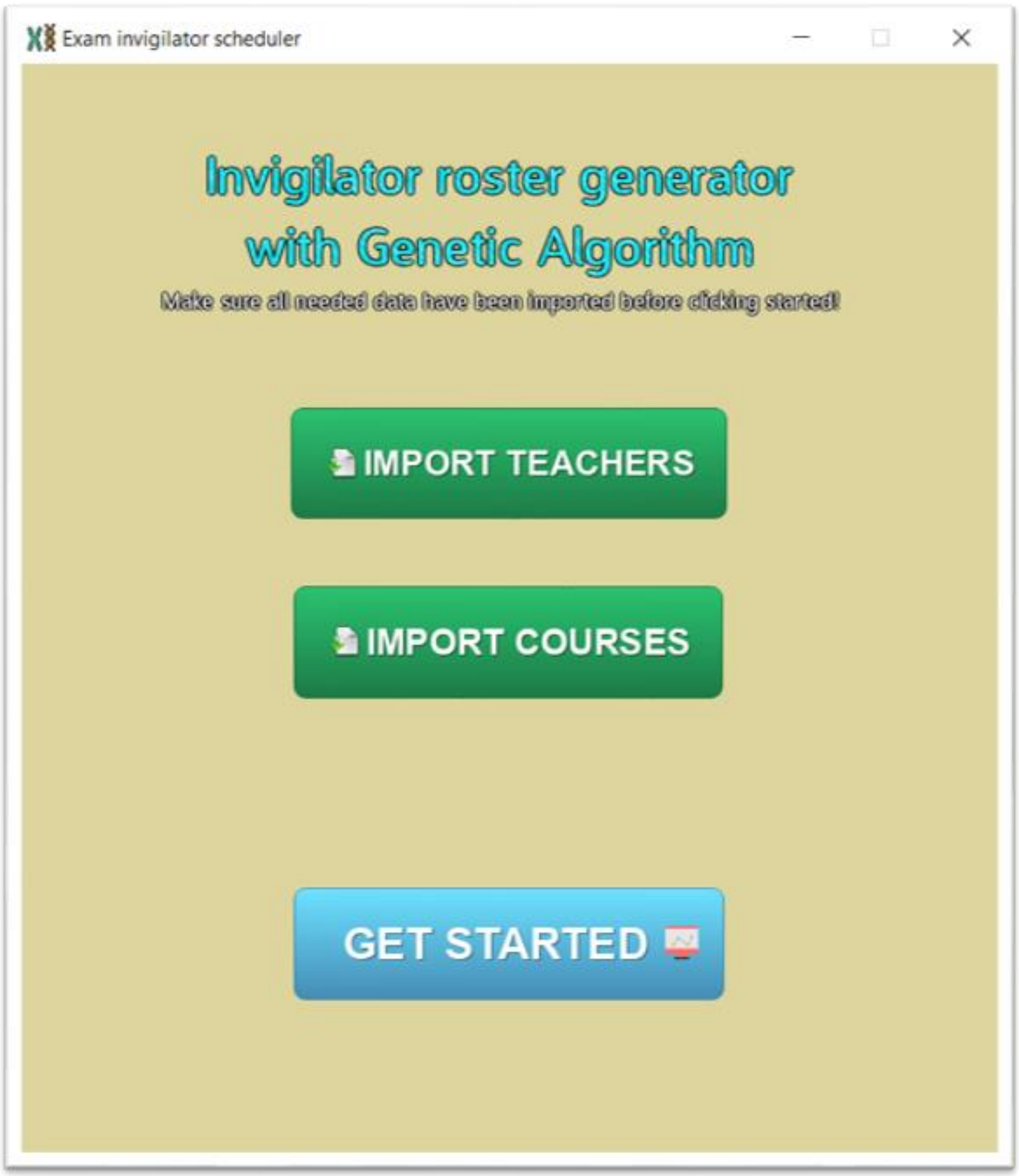

Exam invigilator scheduler
Invigilator roster generator
with Genetic Algorithm
Make sure all needed data have been imported before clicking started!
IMPORT TEACHERS
IMPORT COURSES
GET STARTED

Figure 4.7 welcome UI. This is the first window you see when opening the application.

4.5.2.2 Home UI

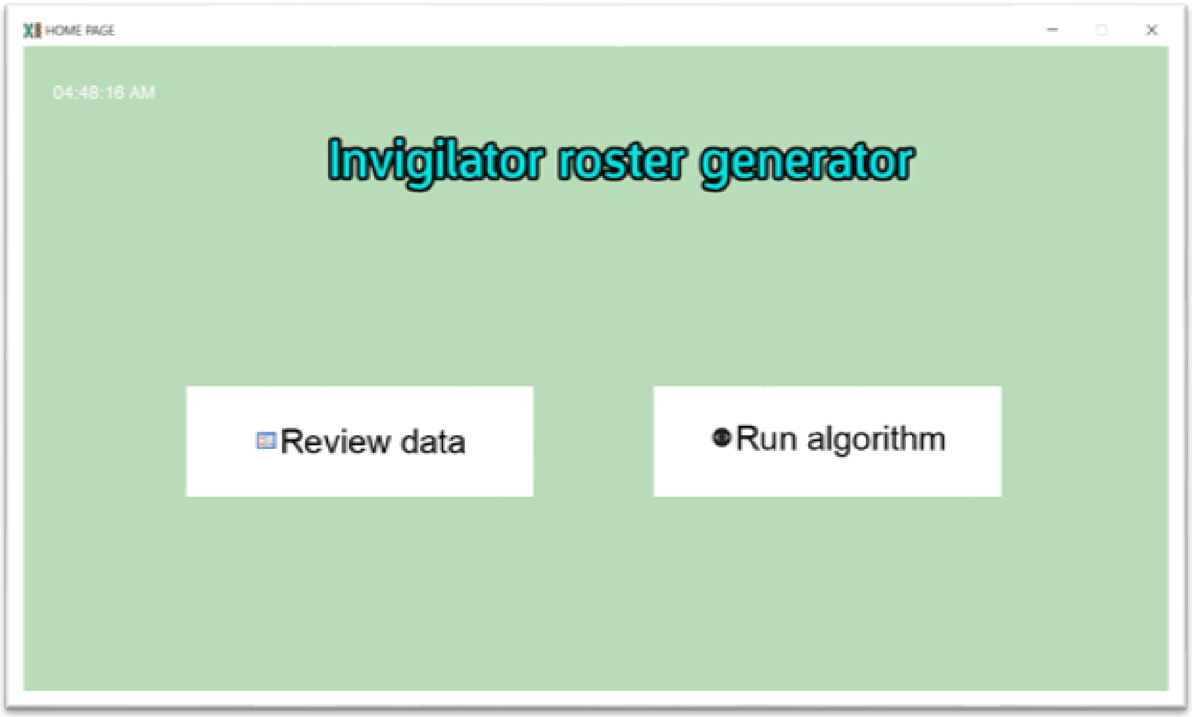
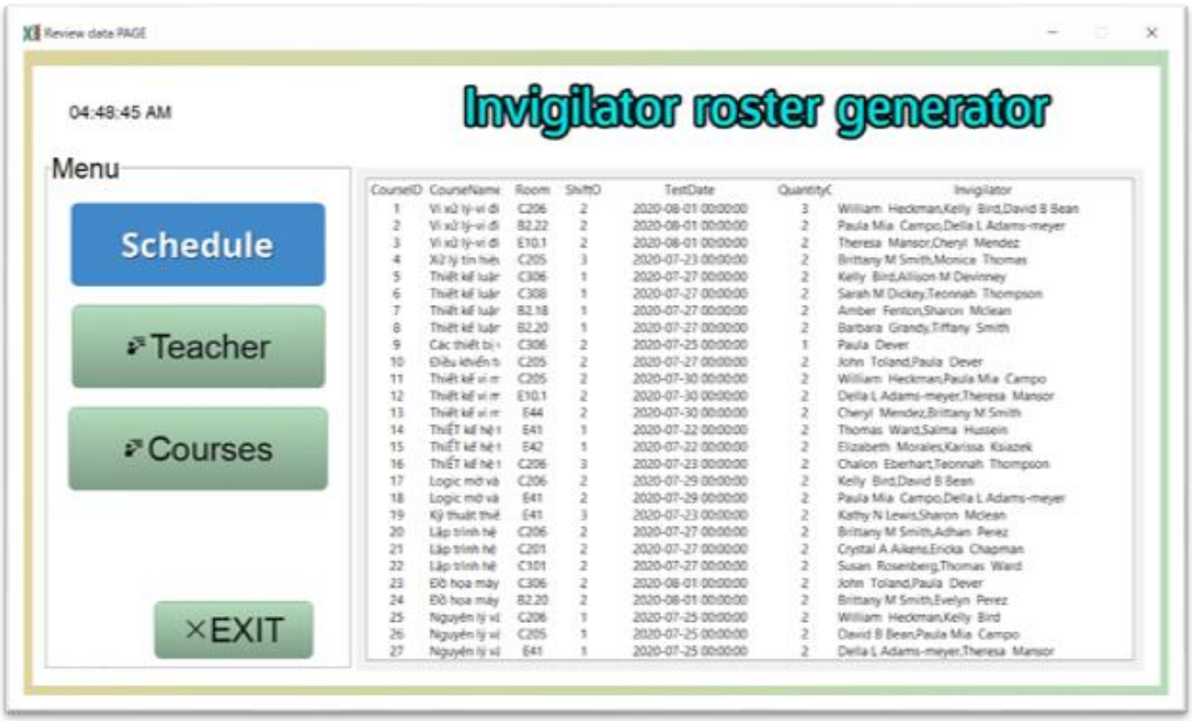


Figure 4.8 Home UI. This is the window when you click Get Started from the welcomeUI (all needed data must be imported in order to click Get Started)

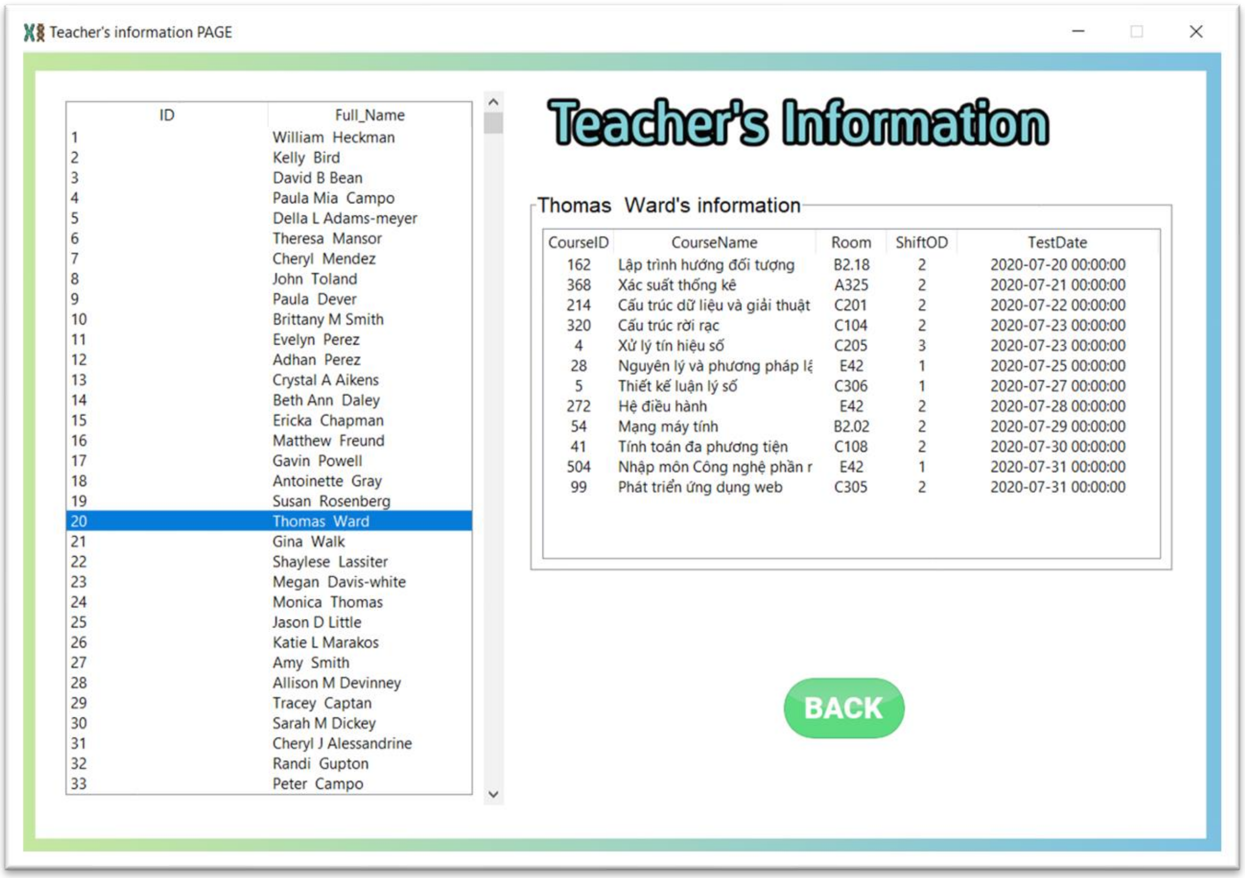
4.5.2.3 Review data UI



CourseID	CourseName	Room	ShiftO	TestDate	QuantityC	Invigilator
1	Vi xử lý-vi đi	C206	2	2020-08-01 00:00:00	3	William Heckman,Kelly Bird,David B Bean
2	Vi xử lý-vi đi	B2.22	2	2020-08-01 00:00:00	2	Paula Mia Campo,Della L Adams-meyer
3	Vi xử lý-vi đi	E10.1	2	2020-08-01 00:00:00	2	Theresa Mansor,Cheryl Mendez
4	Xử lý tín hiệu	C205	3	2020-07-23 00:00:00	2	Brittany M Smith,Monica Thomas
5	Thiết kế luậr	C306	1	2020-07-27 00:00:00	2	Kelly Bird,Allison M Devinney
6	Thiết kế luậr	C308	1	2020-07-27 00:00:00	2	Sarah M Dickey,Teonnah Thompson
7	Thiết kế luậr	B2.18	1	2020-07-27 00:00:00	2	Amber Fenton,Sharon Mclean
8	Thiết kế luậr	B2.20	1	2020-07-27 00:00:00	2	Barbara Grandy,Tiffany Smith
9	Các thiết bị	C306	2	2020-07-25 00:00:00	1	Paula Dever
10	Điều khiển t	C205	2	2020-07-27 00:00:00	2	John Toland,Paula Dever
11	Thiết kế vi m	C205	2	2020-07-30 00:00:00	2	William Heckman,Paula Mia Campo
12	Thiết kế vi m	E10.1	2	2020-07-30 00:00:00	2	Della L Adams-meyer,Theresa Mansor
13	Thiết kế vi m	E44	2	2020-07-30 00:00:00	2	Cheryl Mendez,Brittany M Smith
14	ThiẾT kế hệ t	E41	1	2020-07-22 00:00:00	2	Thomas Ward,Salma Hussein
15	ThiẾT kế hệ t	E42	1	2020-07-22 00:00:00	2	Elizabeth Morales,Karissa Ksiazek
16	ThiẾT kế hệ t	C206	3	2020-07-23 00:00:00	2	Chalon Eberhart,Teonnah Thompson
17	Logic mờ và	C206	2	2020-07-29 00:00:00	2	Kelly Bird,David B Bean
18	Logic mờ và	E41	2	2020-07-29 00:00:00	2	Paula Mia Campo,Della L Adams-meyer
19	Kỹ thuật thiế	E41	3	2020-07-23 00:00:00	2	Kathy N Lewis,Sharon Mclean
20	Lập trình hệ	C206	2	2020-07-27 00:00:00	2	Brittany M Smith,Adhan Perez
21	Lập trình hệ	C201	2	2020-07-27 00:00:00	2	Crystal A Aikens,Ericka Chapman
22	Lập trình hệ	C101	2	2020-07-27 00:00:00	2	Susan Rosenberg,Thomas Ward
23	Đồ họa máy	C306	2	2020-08-01 00:00:00	2	John Toland,Paula Dever
24	Đồ họa máy	B2.20	2	2020-08-01 00:00:00	2	Brittany M Smith,Evelyn Perez
25	Nguyên lý vi	C206	1	2020-07-25 00:00:00	2	William Heckman,Kelly Bird
26	Nguyên lý vi	C205	1	2020-07-25 00:00:00	2	David B Bean,Paula Mia Campo
27	Nguyên lý vi	E41	1	2020-07-25 00:00:00	2	Della L Adams-meyer,Theresa Mansor

Figure 4.9 Review data UI. This is the window when you click Review data from the HomeUI. The results table only appear if you have run the algorithm or this is the second time you open the apps.

4.5.2.4 Review teacher data UI



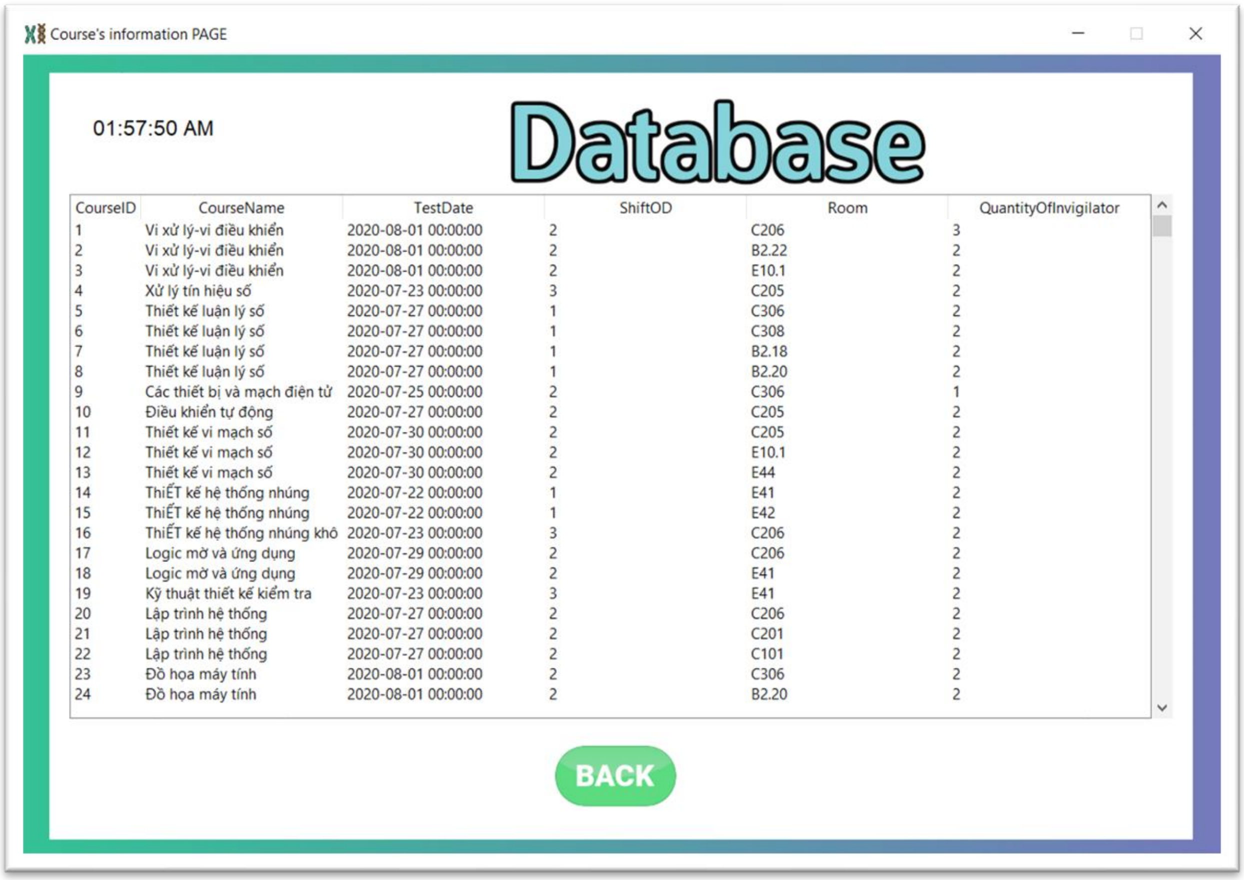
ID	Full_Name
1	William Heckman
2	Kelly Bird
3	David B Bean
4	Paula Mia Campo
5	Della L Adams-meyer
6	Theresa Mansor
7	Cheryl Mendez
8	John Toland
9	Paula Dever
10	Brittany M Smith
11	Evelyn Perez
12	Adhan Perez
13	Crystal A Aikens
14	Beth Ann Daley
15	Ericka Chapman
16	Matthew Freund
17	Gavin Powell
18	Antoinette Gray
19	Susan Rosenberg
20	Thomas Ward
21	Gina Walk
22	Shaylese Lassiter
23	Megan Davis-white
24	Monica Thomas
25	Jason D Little
26	Katie L Marakos
27	Amy Smith
28	Allison M Devinney
29	Tracey Captan
30	Sarah M Dickey
31	Cheryl J Alessandrine
32	Randi Gupton
33	Peter Campo

Thomas Ward's information

CourseID	CourseName	Room	ShiftOD	TestDate
162	Lập trình hướng đối tượng	B2.18	2	2020-07-20 00:00:00
368	Xác suất thống kê	A325	2	2020-07-21 00:00:00
214	Cấu trúc dữ liệu và giải thuật	C201	2	2020-07-22 00:00:00
320	Cấu trúc rời rạc	C104	2	2020-07-23 00:00:00
4	Xử lý tín hiệu số	C205	3	2020-07-23 00:00:00
28	Nguyên lý và phương pháp lậ	E42	1	2020-07-25 00:00:00
5	Thiết kế luận lý số	C306	1	2020-07-27 00:00:00
272	Hệ điều hành	E42	2	2020-07-28 00:00:00
54	Mạng máy tính	B2.02	2	2020-07-29 00:00:00
41	Tính toán đa phương tiện	C108	2	2020-07-30 00:00:00
504	Nhập môn Công nghệ phần r	E42	1	2020-07-31 00:00:00
99	Phát triển ứng dụng web	C305	2	2020-07-31 00:00:00

Figure 4.10 Review teacher data UI. This is the window when you click Teacher from the ReviewUI. The teacher(invigilator)'s information only appear if you have run the algorithm or this is the second time you open the apps.

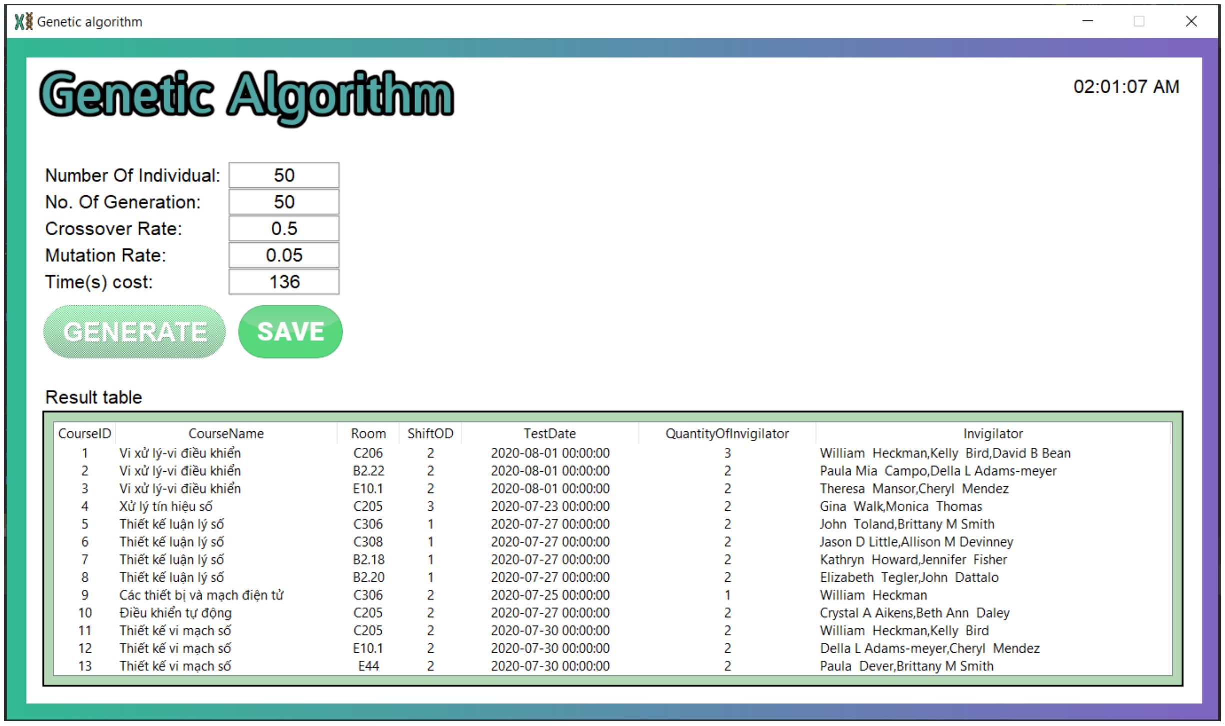
4.5.2.5 Review exam schedule data UI



CourseID	CourseName	TestDate	ShiftOD	Room	QuantityOfInvigilator
1	Vi xử lý-vi điều khiển	2020-08-01 00:00:00	2	C206	3
2	Vi xử lý-vi điều khiển	2020-08-01 00:00:00	2	B2.22	2
3	Vi xử lý-vi điều khiển	2020-08-01 00:00:00	2	E10.1	2
4	Xử lý tín hiệu số	2020-07-23 00:00:00	3	C205	2
5	Thiết kế luận lý số	2020-07-27 00:00:00	1	C306	2
6	Thiết kế luận lý số	2020-07-27 00:00:00	1	C308	2
7	Thiết kế luận lý số	2020-07-27 00:00:00	1	B2.18	2
8	Thiết kế luận lý số	2020-07-27 00:00:00	1	B2.20	2
9	Các thiết bị và mạch điện tử	2020-07-25 00:00:00	2	C306	1
10	Điều khiển tự động	2020-07-27 00:00:00	2	C205	2
11	Thiết kế vi mạch số	2020-07-30 00:00:00	2	C205	2
12	Thiết kế vi mạch số	2020-07-30 00:00:00	2	E10.1	2
13	Thiết kế vi mạch số	2020-07-30 00:00:00	2	E44	2
14	ThiẾT kế hệ thống nhúng	2020-07-22 00:00:00	1	E41	2
15	ThiẾT kế hệ thống nhúng	2020-07-22 00:00:00	1	E42	2
16	ThiẾT kế hệ thống nhúng khô	2020-07-23 00:00:00	3	C206	2
17	Logic mờ và ứng dụng	2020-07-29 00:00:00	2	C206	2
18	Logic mờ và ứng dụng	2020-07-29 00:00:00	2	E41	2
19	Kỹ thuật thiết kế kiểm tra	2020-07-23 00:00:00	3	E41	2
20	Lập trình hệ thống	2020-07-27 00:00:00	2	C206	2
21	Lập trình hệ thống	2020-07-27 00:00:00	2	C201	2
22	Lập trình hệ thống	2020-07-27 00:00:00	2	C101	2
23	Đồ họa máy tính	2020-08-01 00:00:00	2	C306	2
24	Đồ họa máy tính	2020-08-01 00:00:00	2	B2.20	2

Figure 4.11 Review exam schedule data UI. This is the window when you click Courses from the ReviewUI.

4.5.2.6 Genetic Algorithm UI



Number Of Individual: 50
No. Of Generation: 50
Crossover Rate: 0.5
Mutation Rate: 0.05
Time(s) cost: 136

Result table

CourseID	CourseName	Room	ShiftOD	TestDate	QuantityOfInvigilator	Invigilator
1	Vi xử lý-vi điều khiển	C206	2	2020-08-01 00:00:00	3	William Heckman,Kelly Bird,David B Bean
2	Vi xử lý-vi điều khiển	B2.22	2	2020-08-01 00:00:00	2	Paula Mia Campo,Della L Adams-meyer
3	Vi xử lý-vi điều khiển	E10.1	2	2020-08-01 00:00:00	2	Theresa Mansor,Cheryl Mendez
4	Xử lý tín hiệu số	C205	3	2020-07-23 00:00:00	2	Gina Walk,Monica Thomas
5	Thiết kế luận lý số	C306	1	2020-07-27 00:00:00	2	John Toland,Brittany M Smith
6	Thiết kế luận lý số	C308	1	2020-07-27 00:00:00	2	Jason D Little,Allison M Devinney
7	Thiết kế luận lý số	B2.18	1	2020-07-27 00:00:00	2	Kathryn Howard,Jennifer Fisher
8	Thiết kế luận lý số	B2.20	1	2020-07-27 00:00:00	2	Elizabeth Tegler,John Dattalo
9	Các thiết bị và mạch điện tử	C306	2	2020-07-25 00:00:00	1	William Heckman
10	Điều khiển tự động	C205	2	2020-07-27 00:00:00	2	Crystal A Aikens,Beth Ann Daley
11	Thiết kế vi mạch số	C205	2	2020-07-30 00:00:00	2	William Heckman,Kelly Bird
12	Thiết kế vi mạch số	E10.1	2	2020-07-30 00:00:00	2	Della L Adams-meyer,Cheryl Mendez
13	Thiết kế vi mạch số	E44	2	2020-07-30 00:00:00	2	Paula Dever,Brittany M Smith

Figure 4.13 Genetic algorithm UI. This is the window when you click Run algorithm from the HomeUI.

Chapter 5: CONCLUSION AND FUTURE WORK

5.1 Results

Through the process of completion the thesis, we have achieved enough knowledge about optimization algorithms in general, we know how to apply algorithms to solve some optimal problem such as the exam invigilator assignment problem.

Ability to implement and apply genetic algorithm to make an application for exam invigilator assignment.

5.2 Strong points, drawbacks, and future work

5.2.1 Strong points

- Our application responds smoothly to the stated constraints in the thesis, include hard and soft constraints.
- Stable application, friendly user interface, easy to use.
- Scheduled results are stored locally and easily retrieved.

5.2.2 Drawbacks

- Due to the random nature of the genetic algorithm, the outcome sometimes does not fully fulfill the problem.
- In practical, the requirements of scheduling problem are quite rich while our current application only meets the basic requirement.
- Application execution on large datasets is quite slow. To solve this we must reduce the number of generations to make it run faster.
- Application function is still limited.

5.2.3 Future work

There are many different approaches and experiments have not been conducted in this thesis due to the lack of time. There are many adjustments we could do to improve our application and we want to implement them some day in the near future:

- Refactor GUI to improve user experience.
- Improve some function to make it more user friendly.
- Continue improve fitness function.
- Consider additional constraints.
- Combining genetic algorithms with other techniques such as Neural network, fuzzy logic, ... to improve efficiency for the application.

REFERENCES

[1] Mitchell, Melanie. An Introduction to Genetic Algorithms. MIT Press, 1996.

[2] "Adaptive Learning: Fly the Brainy Skies." Wired, vol.10, no.3 (March 2002).

[3] Stuart Mitchell, Stuart Mitchell Consulting, Michael O'Sullivan, Iain Dunning: PuLP: A Linear Programming Toolkit for Python, September 5, 2011

[4] Anita Thengade, Rucha Donal: Genetic Algorithm – Survey Paper, April 2012

[5] Introduction to genetic algorithm [Online]

https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

[6] Juypter Notebook [Online]

https://jupyter-notebook.readthedocs.io/en/stable/

[7] NumPy [Online]

https://numpy.org/doc/

[8] Python [Online]

https://www.python.org/doc/

[9] PyCharm [Online]

https://www.jetbrains.com/pycharm/

[10] Pandas [Online]

https://pandas.pydata.org/docs/

[11] Tkinter [Online]

https://www.tutorialspoint.com/python/python_gui_programming.htm

https://docs.python.org/3/library/tkinter.html