اسم الطالب	رقم الطالب	المستوي الدراسي	القسم/الشعبة
مؤمن محمود عبدالمنعم ابو المعاطي	20210976	الثالث	cs
مريم احمد محمد ابو الفضل	20210905	الثالث	cs
وردة خالد ضاحي مصطفي	20211038	الثالث	cs
فاطمة الزهراء ابراهيم يوسف	20210660	الثالث	cs
عبدالله سامي عبدالعزيز	20210547	الثالث	cs
ميرا هشام حلمي	20210980	الثالث	cs
عبدالرحمن محمود	20210530	الثالث	cs

Roles of teams

Gui:Abdallah Samy

Code:Moamn Mahmoud,Fatma elzahraa,Abdelrahman

Documentation:Mira Hesham,Mariam Ahmed,Warda Khaled

Github link

https://github.com/AbdalllahSamy/AI_Project-Faculty-timetable-scheduling-

Solving a Faculty's Timetable Scheduling Problem using Genetic Algorithms.

1-*Project idea in details*

-the project aims to develop a timetable scheduling system for a faculty using genetic algorithm .

the goal is to automate and optimize the process of generating an efficient timetable for courses ,classrooms ,and faculty members, taking into account various constraints and preferences.

2-Similar applications in the market

1-Timetable of hospital by Genetic Algorithms:

Nurse scheduling is a type of manpower allocation problem that tries to satisfy hospital managers' objectives and nurses' preferences as much as possible by generating fair shift schedules. This paper presents a nurse scheduling problem based on a real case study and proposes two meta-heuristics - a differential evolution algorithm (DE) and a greedy randomized adaptive search procedure (GRASP) - to solve it. To investigate the efficiency of the proposed algorithms, two problems are solved. Furthermore, some comparison metrics are applied to examine the reliability of the proposed algorithms. The computational results in this paper show that the proposed DE outperforms the GRASP.

2-A multi-stage stochastic programming approach for blood supply chain planning:

Perishability of blood products and uncertainty in donation and demand sizes complicate the blood supply chain planning. This paper presents a novel biobjective mixed-integer model for integrated collection, production/screening, distribution, and routing planning of blood products, and seeks to simultaneously optimize the total cost and freshness of transported blood products to hospitals. To cope with inherent uncertainty of input data, a multi-stage stochastic programming approach with a combined scenario tree is presented. Due to the high complexity of the problem, a novel hybrid multi-objective selfadaptive differential evolution algorithm is developed, which benefits from the variable neighborhood search with fuzzy dominance sorting (hereafter it is briefly called MSDV). MSDV is validated through comparing its performance with two of the most common multiobjective evolutionary algorithms (i.e. MOICA and NSGA-II). Applicability of the proposed decision model is also tested through a real case study. Our results show that the solution efficiency of a network can be balanced with its effectiveness through customer satisfaction. Further, several sensitivity analyses are carried out to provide valuable managerial insights.

3-A Differential Evolution Algorithm with Dual Populations for Solving Periodic Railway Timetable Scheduling Problem:

Railway timetable scheduling is a fundamental operational problem in the railway industry and has significant influence on the quality of service provided by the transport system. This paper explores the periodic railway timetable scheduling (PRTS) problem, with the objective to minimize the average waiting time of the transfer passengers. Unlike traditional PRTS models that only involve service lines with fixed cycles, this paper presents a more flexible model by allowing the cycle of service lines and the number of transfer passengers to vary with the time period. An enhanced differential evolution (DE) algorithm with dual populations, termed “dual-population DE” (DP-DE), was developed to solve the PRTS problem, yielding highquality solutions. In the DP-DE, two populations cooperate during the evolution; the first focuses on global search by adopting parameter settings and operators that help maintain population diversity, while the second one focuses on speeding up convergence by adopting parameter settings and operators that are good for local fine tuning. A novel bidirectional migration operator is proposed to share the search experience between the two populations. The proposed DP-DE has

been applied to optimize the timetable of the Guangzhou Metro system in Mainland China and six artificial periodic railway systems. Two conventional deterministic algorithms and seven highly regarded evolutionary algorithms are used for comparison. The comparison results reveal that the performance of DP-PE is very promising.

3 - *A Literature Review of Academic publications (papers/books/articles) relevant to the problem*

1- https://books.google.com/books?hl=ar&lr=&id=Gah5EAAAQBAJ&oi=fnd&pg=PP1&dq=papers/books/articles+about+Solving+a+Faculty%27s+Timetable+Scheduling+Problem++using+Differential+Evolution.&ots=8UfmV2lw6c&sig=IyHxRniIWDFPNDX74xAHpycaARc

2- https://search.proquest.com/openview/9ff56e928e777087e641e3d2e1b29b0a/1?pq-origsite=gscholar&cbl=18750&diss=y

3- https://search.proquest.com/openview/7602d8a3cf221bae578852defe1a7ef7/1?pq-origsite=gscholar&cbl=18750&diss=y

4- https://search.proquest.com/openview/9ff56e928e777087e641e3d2e1b29b0a/1?pq-origsite=gscholar&cbl=18750&diss=y

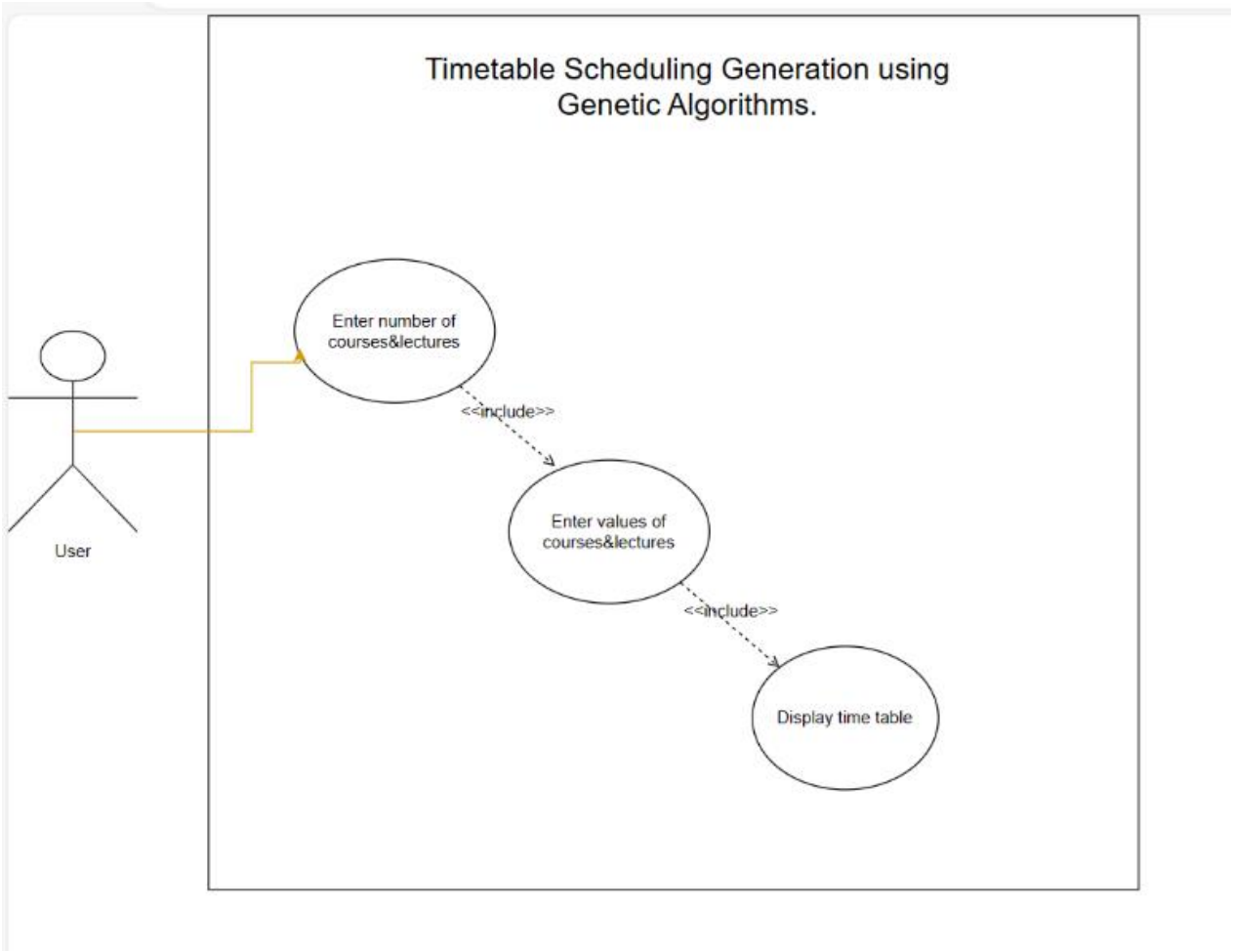
5-
https://www.culturalmanagement.ac.rs/uploads/research_file_1/28a929dff9f841865311525b43d844701e158497.pdf

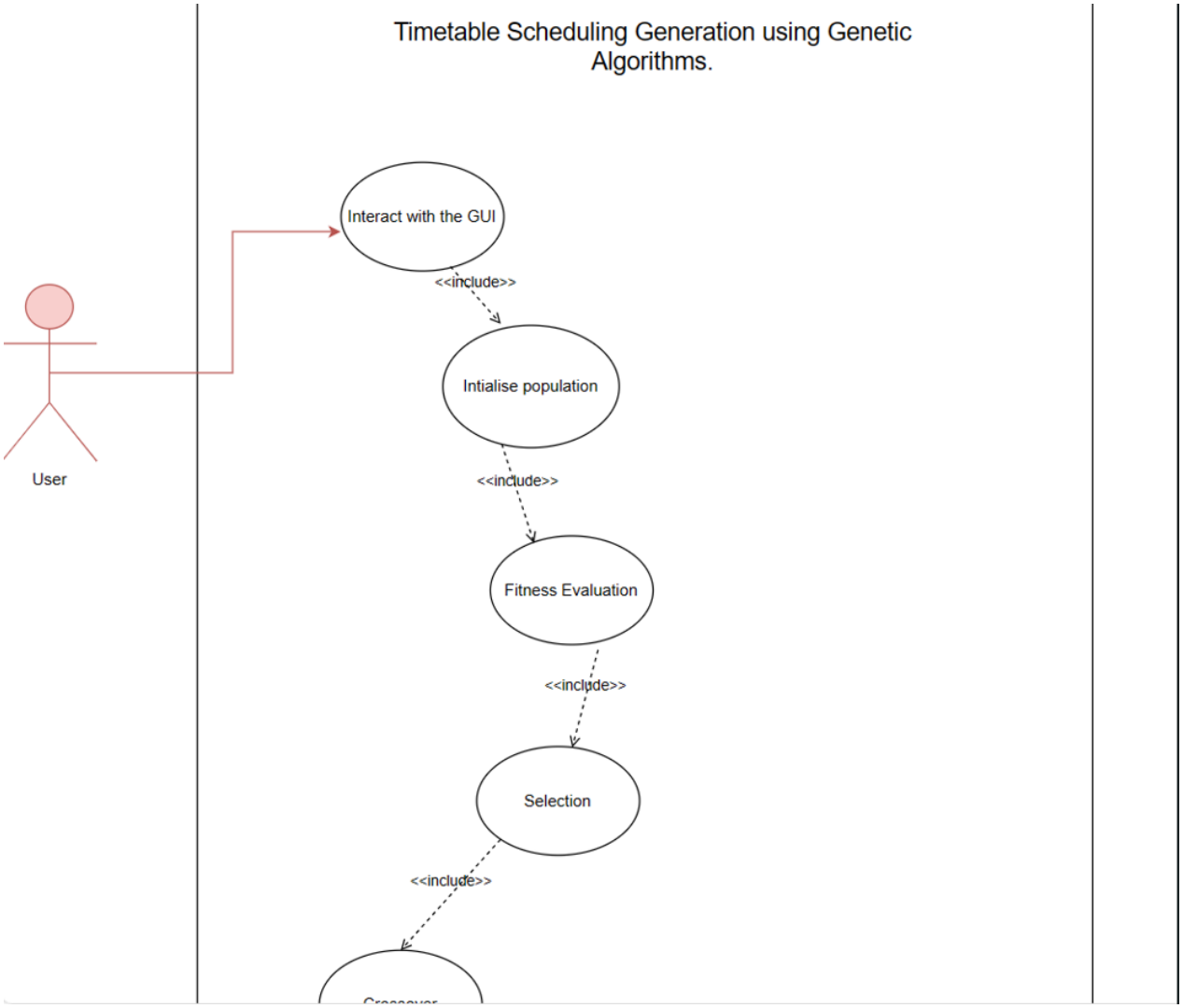
4-*Main functionalities*

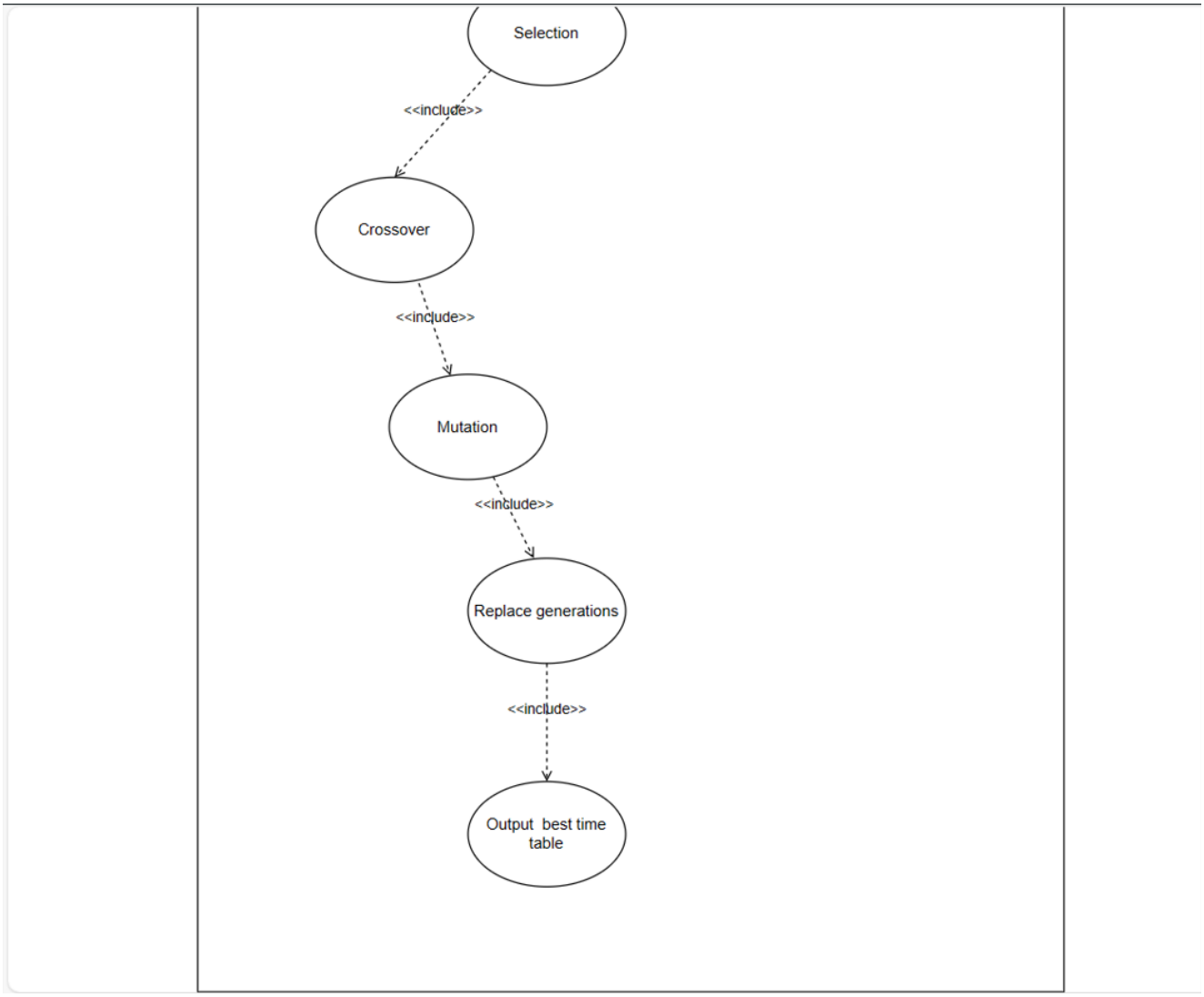
-the developed system will provide the following functionalities:

- *input and management of data:* the system will allow the input and management of data related to courses, classroom, faculty members, and scheduling constrains.
- *constraint modeling:* the system will incorporate the various constraints and preferences involved in timetable scheduling, such as course timings, classroom availability, faculty preferences, and course prerequisites.
- *timetable generation:* the system will utilize the differential evolution algorithm to generate an optimized timetable that satisfies the constraints and preferences.

o_ *timetable visualization:* the system will provide visual representations of the generated timetable , allowed users to easily view and analyze the schedules.




Timetable Scheduling Generation using Genetic Algorithms.
Interact with the GUI
<<include>>
Intialise population
<<include>>
Fitness Evaluation
<<include>>
Selection
<<include>>
User



5- Details of the algorithms/approaches used and the results of the experiments

Solving a faculty's timetable scheduling problem using Genetic Algorithms (GAs) involves several steps. Genetic Algorithms are optimization algorithms inspired by the process of natural selection and genetics. Here is a general outline of the steps you might follow:

Define the Problem:

Constraints: -

1) No two courses are assigned to the same day and time slot.
2) Achieve required hours for each course.

Objectives: -

1) Minimize Conflicts
2) Optimize Timetable Structure

Identify the variables:

```
course_entries = []  # List to store entry widgets for courses
lecturer_entries = []  # List to store entry widgets for lecturers
courses = []  # List to store course names
lecturers = []  # List to store lecturer names
weekdays = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday"]
time_slots = ["Slot1", "Slot2", "Slot3", "Slot4", "Slot5"]
assignment = []
```

Representation of Chromosomes:

Suitable chromosome representation for the solution. Each chromosome in the population represents a potential timetable. That representation ensures feasibility and meets the defined constraints

```
# Chromosome representation
def generate_chromosome():
    chromosome = []

    for lecturer in lecturers:
        for course in courses:
            # Randomly assign a time slot and day for each course
            time_slot = random.choice(time_slots)
            day = random.choice(days)
            chromosome.append((lecturer, course, time_slot, day))

    return chromosome

# Example usage
chromosome = generate_chromosome()
print("Chromosome:\n", chromosome)
```

```
Chromosome:
 [('Lecturer1', 'Course1', 'Slot5', 'Wednesday'), ('Lecturer1', 'Course2', 'Slot2', 'Sunday'), ('Lecturer1', 'Course3', 'Slot4', 'Wednesday'), ('Lecturer
2', 'Course1', 'Slot4', 'Monday'), ('Lecturer2', 'Course2', 'Slot5', 'Monday'), ('Lecturer2', 'Course3', 'Slot4', 'Monday'), ('Lecturer3', 'Course1', 'Sl
ot1', 'Wednesday'), ('Lecturer3', 'Course2', 'Slot5', 'Tuesday'), ('Lecturer3', 'Course3', 'Slot3', 'Thursday')]
```

Initialization:

Generate an initial population of chromosomes randomly and satisfy the constraints of the problem.

```python
# Function to initialize a population
def initialize_population( population_size):
    population = []

    for _ in range(population_size):
        chromosome = generate_chromosome()
        population.append(chromosome)

    return population

# Example usage
population_size = 5
population = initialize_population(population_size)

# Print the initialized population
for i, chromosome in enumerate(population, start=1):
    print(f"Chromosome {i}: {chromosome}\n")
```

```
Chromosome 1: [('Lecturer1', 'Course1', 'Slot3', 'Monday'), ('Lecturer1', 'Course2', 'Slot2', 'Thursday'), ('Lecturer1', 'Course3', 'Slot1', 'Thursday'),
('Lecturer2', 'Course1', 'Slot4', 'Wednesday'), ('Lecturer2', 'Course2', 'Slot5', 'Wednesday'), ('Lecturer2', 'Course3', 'Slot3', 'Thursday'), ('Lecturer
3', 'Course1', 'Slot4', 'Monday'), ('Lecturer3', 'Course2', 'Slot2', 'Tuesday'), ('Lecturer3', 'Course3', 'Slot3', 'Wednesday')]

Chromosome 2: [('Lecturer1', 'Course1', 'Slot1', 'Wednesday'), ('Lecturer1', 'Course2', 'Slot4', 'Wednesday'), ('Lecturer1', 'Course3', 'Slot3', 'Monda
y'), ('Lecturer2', 'Course1', 'Slot3', 'Tuesday'), ('Lecturer2', 'Course2', 'Slot3', 'Sunday'), ('Lecturer2', 'Course3', 'Slot2', 'Wednesday'), ('Lecture
r3', 'Course1', 'Slot3', 'Wednesday'), ('Lecturer3', 'Course2', 'Slot3', 'Thursday'), ('Lecturer3', 'Course3', 'Slot3', 'Monday')]

Chromosome 3: [('Lecturer1', 'Course1', 'Slot2', 'Tuesday'), ('Lecturer1', 'Course2', 'Slot1', 'Sunday'), ('Lecturer1', 'Course3', 'Slot2', 'Sunday'),
('Lecturer2', 'Course1', 'Slot3', 'Monday'), ('Lecturer2', 'Course2', 'Slot1', 'Thursday'), ('Lecturer2', 'Course3', 'Slot3', 'Tuesday'), ('Lecturer3',
'Course1', 'Slot3', 'Tuesday'), ('Lecturer3', 'Course2', 'Slot5', 'Tuesday'), ('Lecturer3', 'Course3', 'Slot4', 'Monday')]

Chromosome 4: [('Lecturer1', 'Course1', 'Slot3', 'Sunday'), ('Lecturer1', 'Course2', 'Slot4', 'Sunday'), ('Lecturer1', 'Course3', 'Slot5', 'Thursday'),
('Lecturer2', 'Course1', 'Slot4', 'Monday'), ('Lecturer2', 'Course2', 'Slot4', 'Thursday'), ('Lecturer2', 'Course3', 'Slot5', 'Sunday'), ('Lecturer3', 'C
ourse1', 'Slot5', 'Wednesday'), ('Lecturer3', 'Course2', 'Slot3', 'Thursday'), ('Lecturer3', 'Course3', 'Slot5', 'Monday')]

Chromosome 5: [('Lecturer1', 'Course1', 'Slot2', 'Wednesday'), ('Lecturer1', 'Course2', 'Slot1', 'Tuesday'), ('Lecturer1', 'Course3', 'Slot4', 'Thursda
y'), ('Lecturer2', 'Course1', 'Slot2', 'Wednesday'), ('Lecturer2', 'Course2', 'Slot4', 'Tuesday'), ('Lecturer2', 'Course3', 'Slot5', 'Sunday'), ('Lecture
r3', 'Course1', 'Slot2', 'Wednesday'), ('Lecturer3', 'Course2', 'Slot2', 'Monday'), ('Lecturer3', 'Course3', 'Slot5', 'Tuesday')]
```

Fitness Function:

Define a fitness function that evaluates how well a timetable meets the objectives and constraints of the scheduling problem

The fitness

function should

assign lower

values for better

solution

```python
# Fitness function
def fitness(chromosome, course_requirements):
    total_penalty = 0

    # Dictionary to keep track of assigned hours for each lecturer and course
    assigned_hours = {(lecturer, course): 0
                      for lecturer in set(lecturer for lecturer, _, _, _ in chromosome)
                      for course in set(course for _, course, _, _ in chromosome)}

    # Calculate penalties based on assigned hours and course requirements
    for assignment in chromosome:
        lecturer, course, time_slot, day = assignment
        assigned_hours[(lecturer, course)] += 1

        # Check if the assigned hours exceed the course requirements
        required_hours = course_requirements.get((lecturer, course), 0)
        penalty = max(0, assigned_hours[(lecturer, course)] - required_hours)
        total_penalty += penalty

        # Check for schedule conflicts (same time slot and day)
        conflicts = [other_assignment for other_assignment in chromosome
                     if other_assignment != assignment and other_assignment[2:] == (time_slot, day)]
        total_penalty += len(conflicts)

    # Lower fitness value is better, so return the negative of the total penalty
    return -total_penalty
```

Selection:

Roulette wheel selection (selection mechanism) is used to choose parents for reproduction based on their fitness

Select chromosomes with the best fitness values to increase the likelihood of good solutions being passed to the next generation.

```python
# Roulette wheel selection based on fitness
def roulette_wheel_selection(population, fitness_values):
    selected_population = []
    total_fitness = sum(fitness_values)

    for _ in range(len(population)):
        pick = random.uniform(0, total_fitness)
        current_sum = 0
        for i, fitness_value in enumerate(fitness_values):
            current_sum += fitness_value
            if current_sum >= pick:
                selected_population.append(population[i])
                break

    return selected_population
```

Crossover (Recombination):

Apply crossover operators to create offspring from selected parents. Crossover combines information from two parent chromosomes to create new solutions.

Ensure that crossover respects our constraints and does not generate infeasible solutions

```
# Crossover: One-Point Crossover
def one_point_crossover(parent1, parent2):
    crossover_point = random.randint(0, len(parent1) - 1)
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]
    return child1, child2

```

Mutation:

Apply mutation operators to introduce small random changes to the offspring chromosomes. Mutation helps explore the search space.

Control the mutation rate to balance exploration and exploitation

```
# Mutation: Swap Mutation
def swap_mutation(individual):
    mutated_individual = copy.deepcopy(individual)
    mutation_point1, mutation_point2 = random.sample(range(len(mutated_individual)), 2)
    mutated_individual[mutation_point1],
    mutated_individual[mutation_point2] = mutated_individual[mutation_point2],
    mutated_individual[mutation_point1]
    return mutated_individual
```

Replacement:

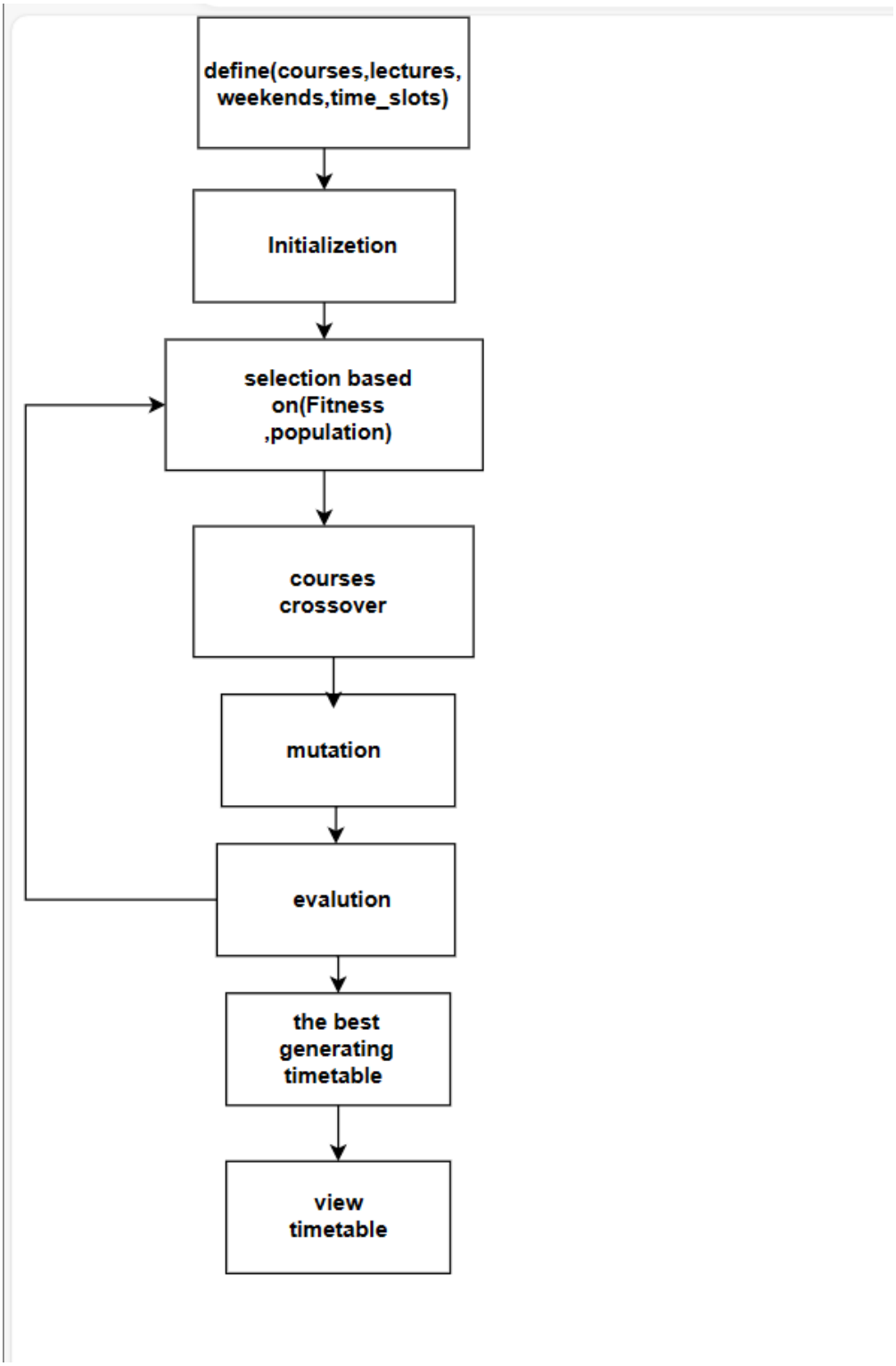
Replace the old generation with the new one. This can involve elitism (keeping the best individuals from the previous generation) to ensure the preservation of good solutions

```python
# Replacement: Generational Replacement
def generational_replacement(current_population, offspring_population, course_requirements):
    combined_population = current_population + offspring_population
    combined_fitness = [fitness(individual, course_requirements) for individual in combined_population]
    combined_population, combined_fitness = zip(*sorted(zip(combined_population, combined_fitness), key=lambda x: x[1], reverse=True))
    return combined_population[:len(current_population)]
```

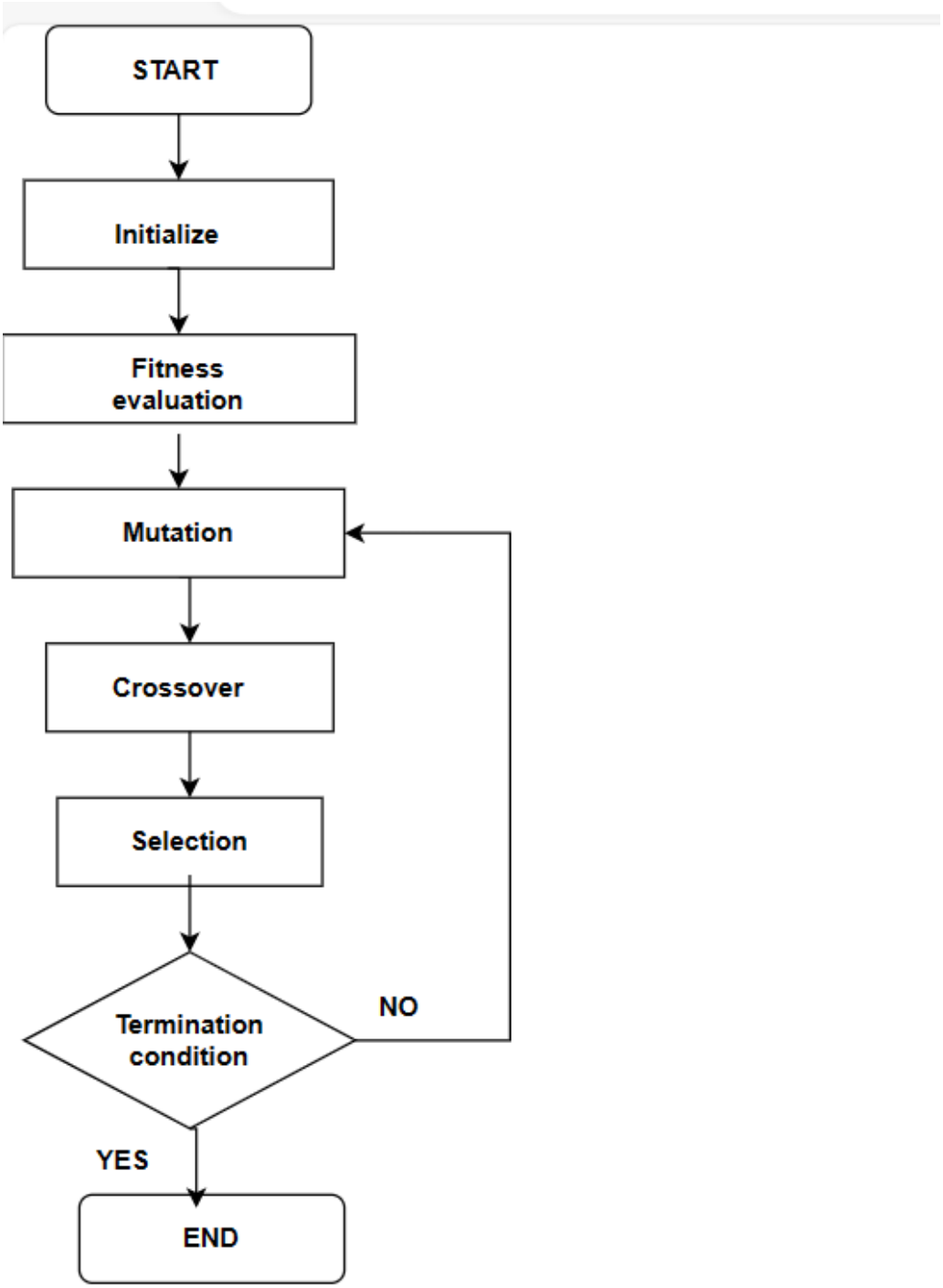
Termination Criteria:

After repeating from step 5 to step 8, define termination criteria by choosing a specific number of generations then calculate the fitness of each chromosome and choose the best fitness from them. Validate the final timetable solution to ensure it meets all constraints and requirements

Block diagram



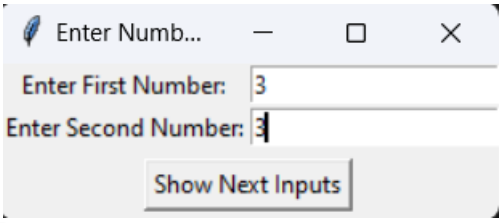
Flowchart

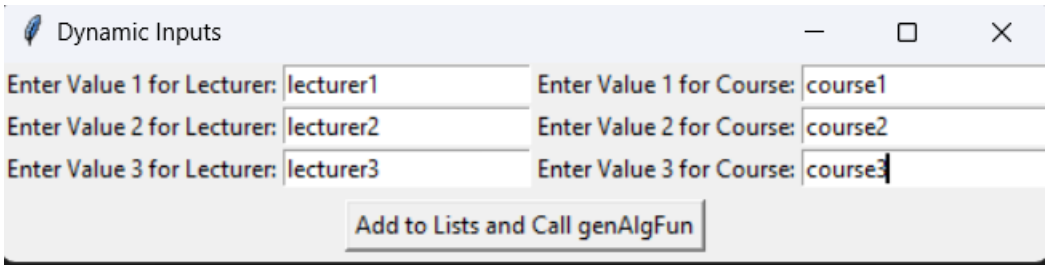

START
Initialize
Fitness evaluation
Mutation
Crossover
Selection
Termination condition
NO
YES
END

6-Experiments & Results

Steps to test the AI program: -

Put the required inputs as you like



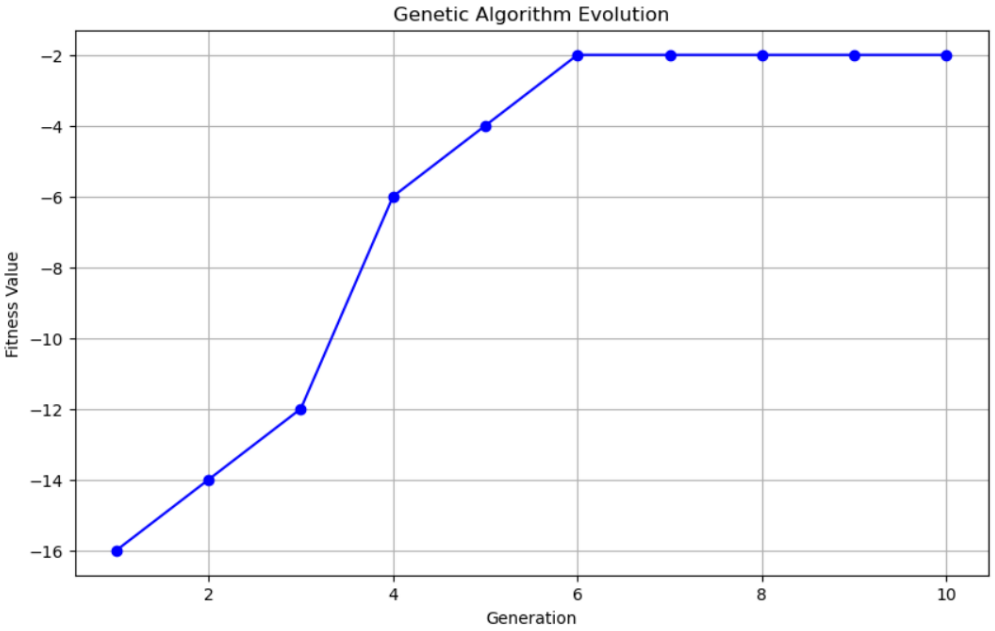


1. Click on the button to generate a timetable.
2. Show the best timetable.

School Timetable

	Slot1	Slot2	Slot3	Slot4	Slot5
Sunday					
Monday	lecturer1 - course1				lecturer3 - course3
Tuesday					
Wednesday		lecturer3 - course1	lecturer2 - course3	lecturer2 - course2	lecturer1 - course2
Thursday			lecturer1 - course3	lecturer3 - course2	lecturer2 - course1

fitness values along 10 generations



7-Analysis, Discussion, and Future Work

Analysis of the results, what are the insights?

- The Genetic Algorithm converges to a timetable that minimizes conflicts, achieves the required hours for each course, and satisfies faculty availability.
- Through analysis, it is observed that certain time slots are more challenging to schedule, leading to the exploration of scheduling strategies for those periods.
- The algorithm successfully balances the workload among faculty members and achieves the required hours for each course.
- Visualization of the final timetable reveals patterns in class distribution and course requirements.

Feedback from stakeholders indicates satisfaction with the generated timetables, and they find the schedule practical and feasible

8-development environment

Tools :jupyter

Prgramming Language : python

Python Libraries : matplotlib.pyplot as plt

numpy as np'
import tkinter as tk
from tkinter import messagebox
import random
import copy

9- the advantages / disadvantages

Advantages:

Flexibility and Adaptability:

The genetic algorithm is flexible and can handle various scheduling scenarios and constraints.

It adapts to different input sizes and evolves schedules over multiple generations.

Optimization Potential:

Genetic algorithms are well-suited for optimization problems, and in this case, for academic schedule optimization.

The algorithm aims to find a solution that minimizes penalties related to course requirements and conflicts.

Parallelism:

Genetic algorithms inherently support parallel processing, allowing for potential parallel execution of fitness evaluations and evolution steps.

Disadvantages:

Random Initialization:

The initial population is randomly generated, which can lead to a wide range of initial solutions.

It might result in slower convergence or suboptimal solutions in the early generations.

Limited Local Search:

The algorithm primarily relies on global exploration through crossover and mutation.

There's limited local search, which might affect the algorithm's ability to fine-tune solutions in the vicinity of promising regions.

Tuning Parameters:

The algorithm involves parameters such as population size, mutation rate, and crossover probability.

Poorly tuned parameters can impact convergence speed and solution quality.

No Elitism Strategy:

The absence of an elitism strategy might lead to the loss of good solutions from one generation to the next.

Elitism could preserve the best individuals, preventing premature convergence to suboptimal solutions.

Behavior Analysis:

Random Initialization Impact:

The random initialization of the population can significantly influence the algorithm's behavior.

Suboptimal initial populations might require more generations to converge to better solutions.

Crossover and Mutation Effects:

The behavior of the algorithm is influenced by the one-point crossover and swap mutation strategies.

Adjustments to these strategies could impact the exploration and exploitation balance.

Future Modifications:

Parameter Tuning:

Experiment with different population sizes, mutation rates, and crossover probabilities to find optimal combinations.

Use techniques like grid search or metaheuristic optimization for parameter tuning.

Local Search Mechanism:

Introduce a local search mechanism to exploit promising regions of the solution space.

This could involve exploring neighboring solutions and refining existing solutions.

Elitism Strategy:

Implement an elitism strategy to preserve the best individuals from one generation to the next.

This can enhance convergence speed and solution quality.

Dynamic Scheduling Constraints:

Consider incorporating dynamic scheduling constraints based on real-world scenarios.

For example, adjusting lecturer availability or course requirements dynamically.

Experiment with Other Operators:

Explore other crossover and mutation operators to diversify the search space.

Experiment with adaptive operators that adjust during the evolution process.

Hybrid Approaches:

Combine genetic algorithms with other optimization techniques or heuristics for hybrid approaches.

This could enhance the algorithm's ability to handle specific aspects of the scheduling problem