National University
of computer and emerging sciences

ID: 21k-4842
AI Project
Spring'24
Course Instructor: Dr. Ahmed Raza.
Lab Instructor: Ms. Maira Hashmi

Introduction

In this report, we explore our methodology for enhancing timetable scheduling through the utilization of genetic algorithms. We provide a detailed overview of our problem-solving strategy, customized to cater to the unique requirements of our university. By systematically examining data preparation, algorithmic design, and solution deployment, we introduce a robust framework with the potential to transform academic scheduling, fostering an environment conducive to academic success and excellence in learning.

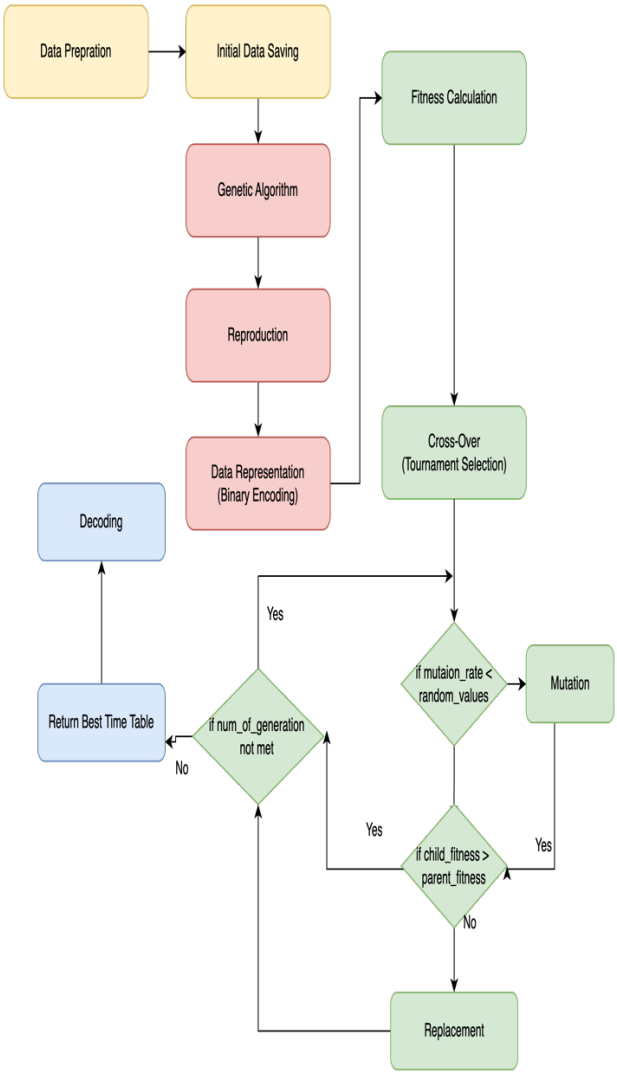
Objective

The primary objective of our timetable scheduling process is to design a system that optimizes resource allocation, minimizes conflicts, and enhances overall efficiency within the academic setting. By neatly organizing course schedules, assigning professors, and allocating classrooms, we aim to create a timetable that ensures smooth academic operations throughout the semester. Specifically, our timetable aims to achieve the following:

1) Minimize clashes between sections, professors, and rooms to facilitate uninterrupted teaching and learning.
2) Optimize resource utilization by effectively allocating classrooms based on section strength and course requirements.
3) Ensure fair distribution of teaching assignments among professors while adhering to workload constraints.
4) Provide students and faculty with a structured timetable that promotes productivity and enhances the learning experience.

Methodology

The following section explains the methodology undertaken to solve the Timetable scheduling Algorithm using Genetic Algorithm.



Data Preparation

Initialize lists for time slots, days, sections, course names, professors, and rooms.

Randomly assign courses to sections, determine their strengths, and designate them as theory or lab-based.

Assign professors to teach courses while ensuring each professor teaches a maximum of three courses.

Initial Data Saving

Save the generated data about courses, sections, professors, and rooms to Excel files for later use.

Genetic Algorithm Setup

Define parameters such as population size, number of generations, mutation rate, and crossover rate.

Create functions for generating initial populations, evaluating fitness, selecting parents, performing crossover, and introducing mutations.

Main Genetic Algorithm Loop

Generate an initial population of potential timetables.

Evaluate the fitness of each timetable based on constraints like avoiding conflicts and meeting requirements.

Select fittest individuals from the population.

Perform crossover between selected parents to create children.

Introduce mutations to explore new potential solutions.

Repeat the process for a predefined number of generations.

Fitness Calculation

Calculate the fitness of each timetable by considering constraints like professor, section, and room schedule conflicts, as well as the distribution of theory and lab courses.

Selection, Crossover, and Mutation

Select the fittest individuals from the population using tournament selection.

Perform crossover between selected parents to create children, introducing genetic diversity.

Introduce mutations to explore new potential solutions and prevent premature convergence.

Decoding Timetable

Decode the binary chromosome representation of the best timetable obtained from the genetic algorithm to obtain readable information about course scheduling.

Final Timetable Saving

Save the optimized timetable for each day into CSV files for reference and implementation.

Experiment: Evaluating Genetic Algorithm Performance with Different Population Sizes

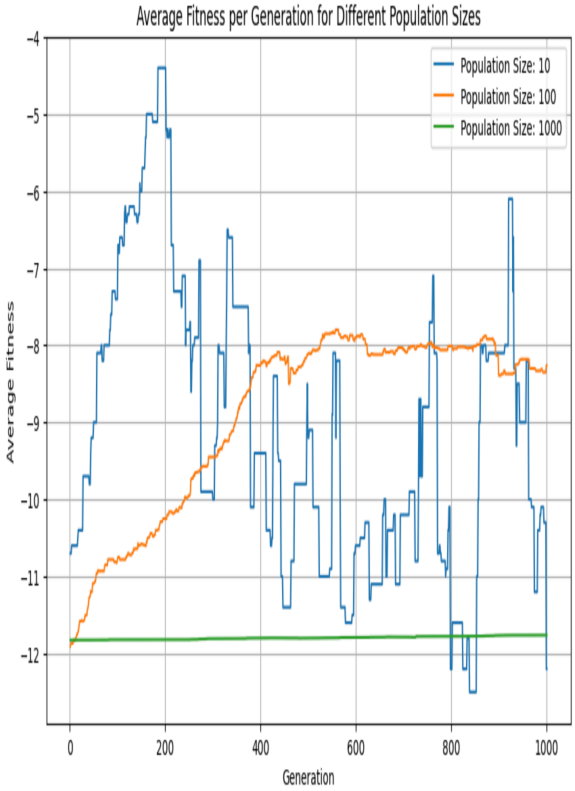
Purpose

The objective of this experiment is to evaluate the performance of the genetic algorithm used for timetable generation under varying population sizes. By varying the population size—i.e., the number of individuals in each generation—we aim to understand how this parameter affects the algorithm's convergence behavior and solution quality.

Methodology

Population Sizes: I selected three different population sizes: 10, 100, and 1000 individuals per generation.

Genetic Algorithm Execution For each population size, we executed the genetic algorithm with a fixed number of generations (1000) to ensure consistency in the evaluation.



Data Collection We collected the average fitness per generation throughout the execution of the algorithm for each population size.

Results

The generated graphs illustrate the average fitness per generation for each population size. By examining these graphs, we can observe how the algorithm's performance varies with different population sizes.

Population Size 10: This small population size results in rapid convergence but may lead to premature convergence to suboptimal solutions due to limited diversity. As can be seen from the graph there is huge variation across generations, the population reached it best fitness value i-e approximately -3 in the 200th generation. However, due to limited diversity it diverged, and the population further got worse in the successive generations.

Population Size 100: With a moderate population size, the algorithm strikes a balance between exploration and exploitation, leading to more robust convergence behavior. As can been seen from the graph the population size did increase the average fitness per population and has given the best result in all the three chosen population sizes.

Population Size 1000: A larger population size provides increased diversity and exploration capacity, potentially leading to better solutions, but may require more computational resources. However, the experimentation revealed that there was not any performance enhancement in the population throughout the generations.

Experiment Analysis: Average Fitness per Generation for Different Numbers of Generations

Purpose

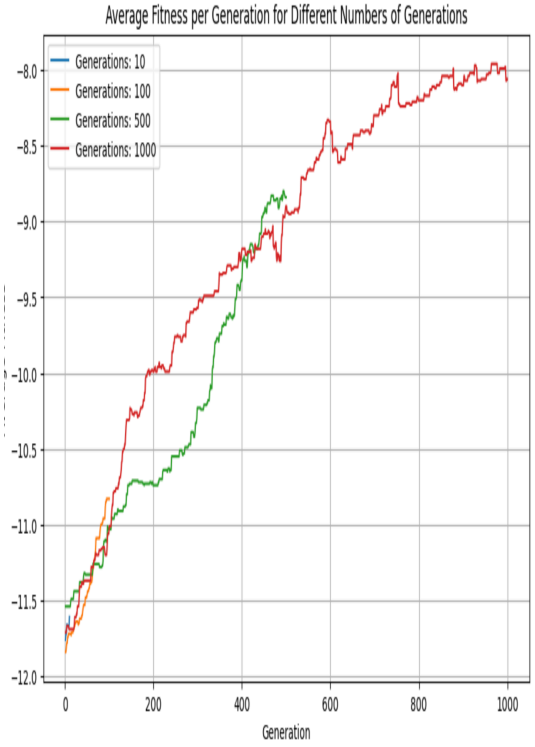
In this experiment, we explored the impact of varying the number of generations on the performance of our genetic algorithm while keeping the population size constant at 100. The objective was to investigate how the convergence behavior and solution quality of the algorithm change with different numbers of generations.

Methodology

We defined a function run_experiment(num_generations) to execute the genetic algorithm with different numbers of generations.

We selected four numbers of generations: 10, 100, 500, and 1000, representing a range from a relatively low number to a higher number of iterations.

For each number of generations, we ran the genetic algorithm and recorded the average fitness per generation.



Results:

As illustrated in the plot, each line represents the average fitness per generation for a specific number of generations.

Initially, with a small number of generations (e.g., 10), the average fitness fluctuates significantly, indicating insufficient time for the algorithm to converge to an optimal solution.

As the number of generations increases, the average fitness stabilizes and tends to improve steadily. This trend suggests that increasing the number of generations allows the algorithm more opportunities to explore the solution space and refine the solution.

However, it must be noted that in no experiment we reached to the optimal solution the best fitness we achieved through our experiment was -6.45 per population which indicates there is a room for improving the algorithm.

Conclusion

From our experiments with the genetic algorithm, we learned that having around 100 individuals in each generation works best for creating timetables. It strikes a good balance between exploring new possibilities and sticking with what's working. Increasing the population size to 1000 didn't really help much, and it took more computer power. When we kept the population size the same but changed the number of generations, we saw that more generations meant more stable progress. However, we never quite reached the perfect timetable, showing there's still room to improve our algorithm. Overall, these experiments show that finding the right balance between population size and number of generations is crucial for making good timetables.