Shayaan Hasnain Ahmad

20I-0647

Section A

AI – Assignment 2

The code starts by defining the problem parameters, including the number of courses, halls, timeslots, and the maximum hours a hall can be used. It also defines a list of conflicts between courses, represented as tuples of three elements (c1, c2, time), where c1 and c2 are the courses that conflict, and time is the timeslot where the conflict occurs.

The code then defines a representation for a solution to the scheduling problem, which is a list of tuples (course, timeslot, hall). It also defines several helper functions, such as generate_random_solution, which generates a random solution; calculate_conflicts, which calculates the number of conflicts in a solution; calculate_penalty, which calculates the penalty for a solution; and calculate_fitness, which calculates the fitness of a solution, defined as the sum of conflicts and penalties.

The code then defines the main genetic algorithm function, genetic_algorithm, which takes as input the population size, tournament size, crossover probability, mutation probability, and the number of generations to run. The function starts by generating an initial population of random solutions. It then iterates over a fixed number of generations, creating a new population by selecting parents from the current population using tournament selection, applying single-point crossover and mutation operators to the parents to generate new children, and selecting the best child to replace the parent in the next generation. The function returns the best solution and its fitness at the end of the algorithm.

Finally, the code runs the genetic_algorithm function with some default parameters and prints the best solution and its fitness. The default parameters generate a population of 100 solutions, use a tournament size of 5, a crossover probability of 0.8, a mutation probability of 0.1, and run for 100 generations.

FUNCTIONS EXPLAINED:

1. generate_random_solution(): This function generates a random solution to the scheduling problem. It does this by randomly assigning each course to a timeslot and a hall.

2. calculate_conflicts(solution): This function calculates the number of conflicts in a given solution. It does this by checking the conflicts list, which specifies which courses cannot be scheduled at the same time and in the same hall. If any two conflicting courses are scheduled at the same time and in the same hall in the given solution, then a conflict is counted.

3. calculate_penalty(solution): This function calculates the penalty for a given solution. The penalty is based on two factors: (1) the total number of hours a hall is used, and (2) the number of conflicts between courses scheduled in the same timeslot but different halls. If a hall is used for more than MAX_HALL_HOURS hours, then a penalty of 10 is incurred. If two conflicting courses are scheduled in the same timeslot but different halls, then a penalty of 100 is incurred.

4. calculate_fitness(solution): This function calculates the fitness of a given solution. It does this by adding the number of conflicts and the penalty together.

- The fitness value is calculated as the sum of two components: the number of conflicts in the solution and the penalty for the solution. The number of conflicts is calculated by the calculate_conflicts function, which counts the number of conflicting course assignments in the solution. A conflict occurs when two courses that have a conflict are assigned to the same timeslot and different halls. The penalty is calculated by the calculate_penalty function, which calculates the penalty for the solution based on two criteria. The first criterion is that no hall can be used for more than MAX_HALL_HOURS hours in total. The second criterion is that if two conflicting courses are assigned to the same timeslot, they must be assigned to different halls. If a course violates either of these criteria, a penalty is added to the total penalty for the solution. The fitness value is the sum of the number of conflicts and the penalty, so a lower fitness value indicates a better solution. The genetic algorithm aims to minimize the fitness value over multiple generations to find the best solution to the scheduling problem.

5. generate_population(size): This function generates a population of size random solutions. It does this by calling generate_random_solution() size times.

6. tournament_selection(population, size): This function performs tournament selection on a given population. It does this by selecting a random sample of size solutions from the population, and then returning the solution with the lowest fitness.

7. single_point_crossover(parent1, parent2, probability): This function performs single-point crossover on two parent solutions. It does this by selecting a random crossover point between the first and last courses, and then swapping the courses after that point between the two parents to create two children. The crossover probability determines whether or not the crossover operation is performed.

8. mutation(solution, probability): This function mutates a given solution by randomly changing one course. The mutation probability determines whether or not the mutation operation is performed.

- First, the function checks if the mutation should be applied by comparing a randomly generated number between 0 and 1 with the given probability. If the randomly generated number is greater than probability, then the function simply returns the original solution without making any changes. Otherwise, it selects a random course from the solution and generates random values for the timeslot and hall. Then it replaces the existing values for the selected course with the new randomly generated values to create a mutated solution. Finally, the mutated solution is returned.

9. genetic_algorithm(population_size, tournament_size, crossover_probability, mutation_probability, num_generations): This function performs the genetic algorithm to solve the scheduling problem. It does this by first generating a population of population_size random solutions, and then iterating through num_generations generations of selection, crossover, and mutation. In each generation, two parents are selected using tournament selection, and then single-point crossover and mutation are applied to create two children. The resulting population is then selected by tournament selection to create the next generation. The algorithm terminates after num_generations generations, and the best solution found is returned.

10. Main function -> This is the main block of the code, which is executed when the script is run. It sets the parameters for the genetic algorithm, calls genetic_algorithm() to solve the scheduling problem, and prints out the best solution and fitness found.