Report
ARTIFICIAL INTELLIGENCE
Assignment # 2

Ahmed Baig
20i-1884
CS-B

Code Explanation:

This code implements a genetic algorithm to solve an exam scheduling problem. The problem involves scheduling exams for a set of courses in a limited number of time slots and exam halls, while avoiding scheduling conflicting courses in the same time slot.

The code starts by defining the necessary input data, including the list of courses, available time slots, exam halls, maximum number of hours per hall, and conflicting pairs of courses.

Next, several functions are defined to create a random solution, calculate the fitness of a solution, initialize a population of random solutions, select parents for crossover using tournament selection, apply single-point crossover to two parents, and apply mutation to a solution.

The **run_genetic_algorithm()** function is the main function that runs the genetic algorithm. It takes as input the population size, tournament size, crossover probability, mutation probability, and number of generations. It initializes a population of random solutions, selects parents for crossover, applies crossover and mutation to create a new population, and repeats this process for a certain number of generations. Finally, it returns the best solution found and its fitness.

The code then sets the input parameters and runs the **run_genetic_algorithm()** function to find the best solution to the exam scheduling problem. It prints the best solution found and its fitness.

Overall, the code implements a genetic algorithm to find an optimal solution to a complex exam scheduling problem by randomly generating solutions, evaluating their fitness, and using selection, crossover, and mutation operators to evolve better solutions over multiple generations.

Variables and Functions:

1. **import random**: This imports the random module, which is used later to randomly select items from lists.

2. **courses = ['C1', 'C2', 'C3', 'C4', 'C5']**: This is a list of courses that need to be scheduled.
3. **time_slots = ['T1', 'T2', 'T3']**: This is a list of time slots that are available for scheduling the exams.
4. **exam_halls = ['H1', 'H2']**: This is a list of exam halls that are available for conducting the exams.
5. **hall_max_hours = {'H1': 6, 'H2': 6}**: This is a dictionary that specifies the maximum number of hours that can be used for each exam hall.
6. **conflicting_pairs = [('C1', 'C2'), ('C1', 'C4'), ('C2', 'C5'), ('C3', 'C4'), ('C4', 'C5')]**: This is a list of tuples that specifies pairs of courses that cannot be scheduled in the same time slot.
7. **create_random_solution()**: This function creates a random solution by randomly assigning a time slot and an exam hall to each course.
8. **calculate_fitness(solution)**: This function calculates the fitness of a given solution. The fitness is calculated based on the number of conflicts between courses that are scheduled in the same time slot, and the number of hours used by each exam hall. The fitness is a measure of how good the solution is.
9. **initialize_population(population_size)**: This function creates a population of random solutions.
10. **select_parents(population, tournament_size)**: This function selects parents for crossover using a tournament selection method. It randomly selects a subset of the population and selects the best solution from that subset.
11. **apply_single_point_crossover(parent1, parent2)**: This function applies single point crossover to two parents to create two children.
12. **apply_mutation(solution, mutation_probability)**: This function applies mutation to a given solution. It randomly selects a course and assigns a new exam hall to it with a certain probability.
13. **run_genetic_algorithm(population_size, tournament_size, crossover_probability, mutation_probability, generations)**: This function runs the genetic algorithm. It initializes a population, selects parents for crossover, applies crossover and mutation, and creates a new population. It repeats this process for a certain number of generations and returns the best solution found.
14. **population_size=100**: This specifies the size of the population.
15. **tournament_size=5**: This specifies the size of the tournament used for parent selection.

16. **crossover_probability=0.8**: This specifies the probability of crossover.
17. **mutation_probability=0.1**: This specifies the probability of mutation.
18. **generations=100**: This specifies the number of generations.
19. **best_solution, best_fitness = run_genetic_algorithm(population_size, tournament_size, crossover_probability, mutation_probability, generations)**: This runs the genetic algorithm and stores the best solution and its fitness.
20. **print(best_solution)**: This prints the best solution found.
21. **print(f"Fitness: {best_fitness}")**: This prints the fitness of the best solution found.