**Name: Hamza**

**Surname: Madi**

**Student Number:222009986**

**(a) Chromosome and Population Size**

In the initial population, each chromosome represents a solution where each task is assigned to a person. A chromosome is structured as a list of integers, where the index corresponds to a task, and the value at each index represents the person assigned to that task. For example, if there are 5 tasks and 5 people, a possible chromosome could be **[2, 0, 4, 3, 1] ,** where task 0 is assigned to person 2, task 1 is assigned to person 0, and so on.

For this problem, I chose a population size of 50. This ensures a diverse set of solutions in each generation while keeping the computational load manageable. Each chromosome in the population is generated randomly at the start of the algorithm using the generate chromosome function, which ensures that no two chromosomes are identical in the initial population.

**(b) Fitness Function**

The fitness function evaluates how well a chromosome solves the task-person assignment problem. For each chromosome, I calculate the total score by summing the performance scores of the assigned person for each task. The performance scores are stored in a matrix where each row represents a person, and each column represents a task. The fitness of a chromosome is the sum of these scores for the assigned tasks. A higher score indicates a better solution, where the assignments match individuals to tasks where they perform optimally. The fitness function is implemented in the fitness method.

**(c) Selection Method**

For selection, I used tournament selection, a popular method in genetic algorithms. In this approach, I randomly select three chromosomes from the population and compare their fitness values. The chromosome with the highest fitness is selected to be part of the next generation. This process is repeated until I have selected enough chromosomes for the mating pool. The advantage of tournament selection is that it maintains diversity while also favouring higher-quality solutions, allowing better-performing chromosomes to have a higher chance of reproduction.

**(d) Mutation Operator and Rate**

The mutation operator is designed to introduce variability into the population and prevent premature convergence. I implemented mutation by randomly selecting two positions in a chromosome and swapping the assigned persons at these positions. This process is only applied to a chromosome with a probability equal to the mutation rate.

In my implementation, I used a mutation rate of 0.1 (10%), meaning that on average, 1 in 10 chromosomes undergoes mutation in each generation. This helps maintain genetic diversity without disrupting the overall convergence toward an optimal solution.

**(e) Crossover Operator and Rate**

For crossover, I used single-point crossover, where two parent chromosomes exchange segments of their genetic material. Specifically, I select a random crossover point and create two offspring by swapping the segments of the parent chromosomes after this point. For instance, if the parents are [2, 0, 4, 3, 1] and [1, 3, 0, 2, 4], and the crossover point is 2, the offspring could be [2, 0, 0, 2, 4] and [1, 3, 4, 3, 1].

The crossover rate I used is 0.8 (80%), meaning that 80% of the population undergoes crossover during reproduction. This high rate ensures that most of the population benefits from the genetic diversity introduced by crossover, while the remaining 20% of chromosomes are carried over from the previous generation without change.

**(f) Termination Criterion**

The termination criterion I used is based on a fixed number of generations. The algorithm runs for 1000 generations, after which it terminates, regardless of whether an optimal solution has been found. I chose this criterion to ensure the algorithm runs for a sufficient duration to explore the solution space while avoiding excessive computation time.