EAST WEST UNIVERSITY

Assignment– 02

Submitted By -

Name : Tamanna Sultana Tinne

ID : 2020-2-60-057

Course: CSE-366(Artificial Intelligence)

Section: 03

Submitted To -

Dr. Mohammad Rifat Ahmmad Rashid

Assistant Professor
Department of Computer Science & Engineering

Title: Robot Task Optimization Using Genetic Algorithm:

Introduction:

The development of a Genetic Algorithm (GA) to optimize task assignments in dynamic production environments represents a cutting-edge approach in robotics and operations research. This report details the process of implementing a GA to distribute tasks among multiple robots efficiently, aiming to minimize total production time, balance workload across robots, and prioritize critical tasks effectively. The project underscores the potential of GAs in enhancing operational efficiencies and task management in robotic systems.

Objective:

The primary goal of this project was to create and implement a Genetic Algorithm that optimizes the assignment of a set of tasks to a pool of robots within a dynamic production setting. Key objectives included minimizing the overall production time, ensuring equitable distribution of tasks across robots, effectively prioritizing tasks, and providing a comprehensive visualization of the final task assignments, robot efficiencies, and task priorities.

Methodology:

The methodology encompassed several crucial steps consistent with the principles of genetic algorithms:

Tools:

1. Python version 3.12
2. Jupyter notebook
3. Visual Studio (Python environment for test run)

Libraries:

1. Numpy
2. Matplotlib
3. math

Data Preparation: Mock data for tasks and robots were generated, including task durations, priorities, and robot efficiencies.

GA Implementation: A Genetic Algorithm was designed to optimize task assignments by considering factors such as task duration, robot efficiency, and task priority.

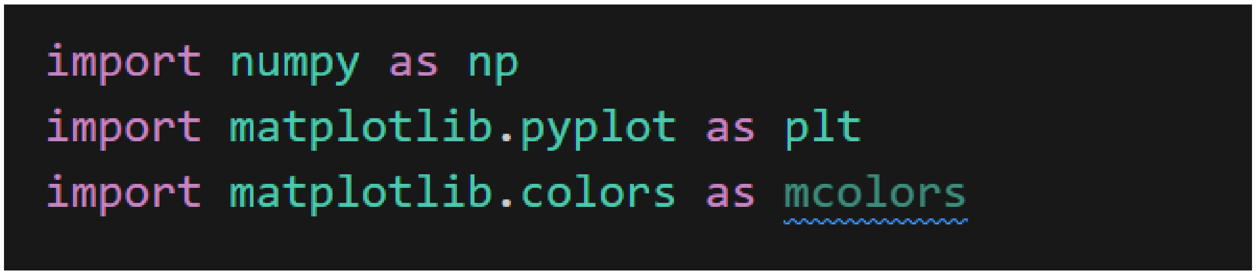
Visualization: A detailed visualization was planned to illustrate task assignments, with annotations for task duration, robot efficiency, and task priority.

Descriptive Analysis:

In the implementation, mock data for tasks and robots are generated to simulate a dynamic production environment, using random and uniform distributions to assign task durations, priorities, and robot efficiencies. The beginnings of a fitness function suggest an approach to evaluating potential solutions based on task assignments to robots, aiming to minimize total production time and ensure workload balance while considering task priorities. However, the absence of details on selection, crossover, and mutation operations leaves the full implementation of the Genetic Algorithm (GA) unclear. Similarly, the process of individual representation—presumably vectors indicating which robot is assigned to each task—is implied but not explicitly shown.

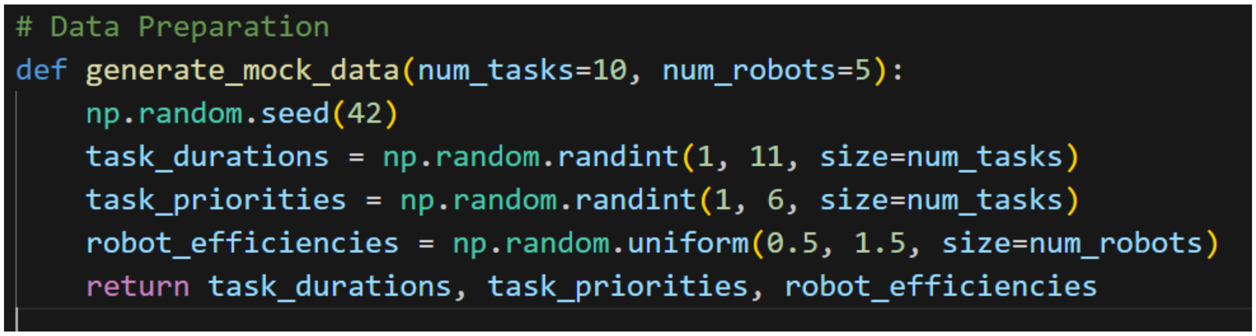
Implementation:

The Implementation of the code snippets looks like -



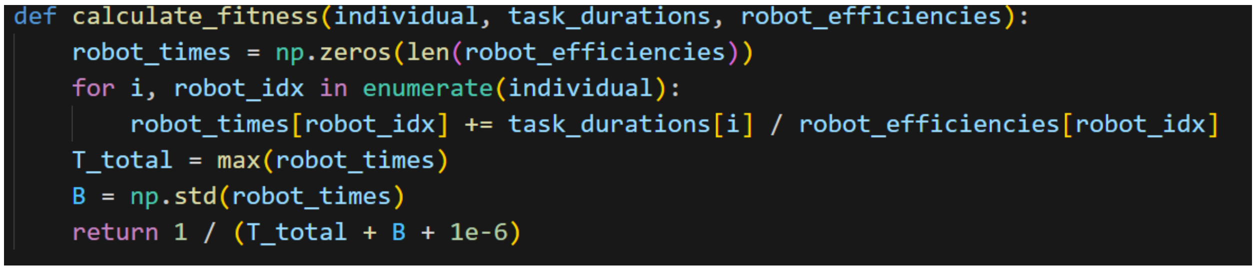
```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
```

Imported the necessary libraries.



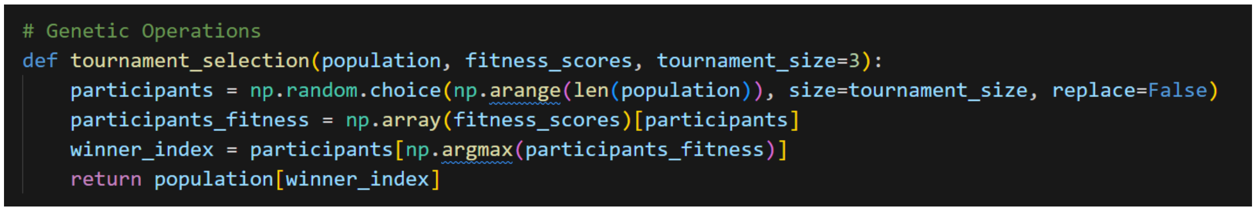
```python
# Data Preparation
def generate_mock_data(num_tasks=10, num_robots=5):
    np.random.seed(42)
    task_durations = np.random.randint(1, 11, size=num_tasks)
    task_priorities = np.random.randint(1, 6, size=num_tasks)
    robot_efficiencies = np.random.uniform(0.5, 1.5, size=num_robots)
    return task_durations, task_priorities, robot_efficiencies
```

Data generation using mock or random given data.



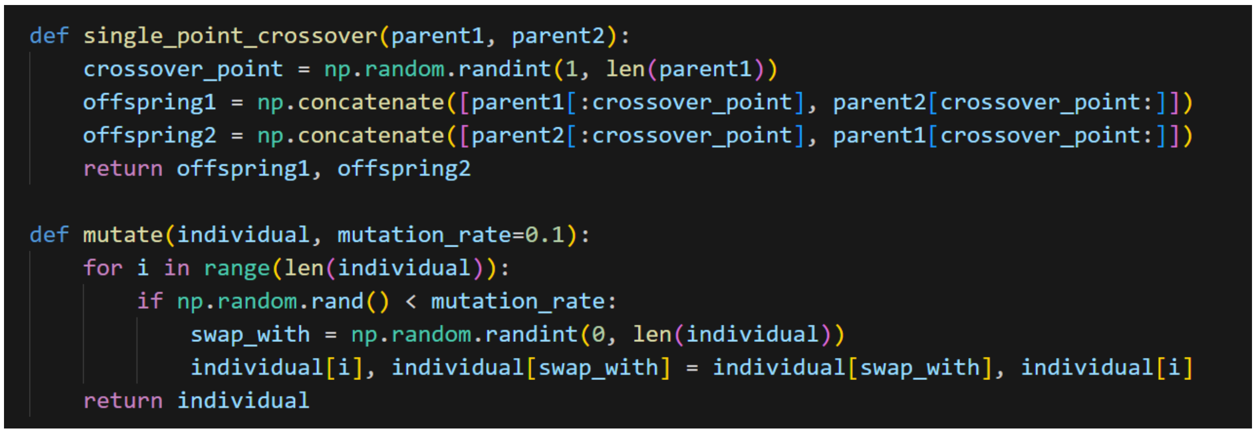
```python
def calculate_fitness(individual, task_durations, robot_efficiencies):
    robot_times = np.zeros(len(robot_efficiencies))
    for i, robot_idx in enumerate(individual):
        robot_times[robot_idx] += task_durations[i] / robot_efficiencies[robot_idx]
    T_total = max(robot_times)
    B = np.std(robot_times)
    return 1 / (T_total + B + 1e-6)
```

focused on developing the fitness function, a core component of the GA that evaluates the optimality of task assignments.



```python
# Genetic Operations
def tournament_selection(population, fitness_scores, tournament_size=3):
    participants = np.random.choice(np.arange(len(population)), size=tournament_size, replace=False)
    participants_fitness = np.array(fitness_scores)[participants]
    winner_index = participants[np.argmax(participants_fitness)]
    return population[winner_index]
```

defines a function for tournament selection, which is a method used in genetic algorithms to select individuals for reproduction.

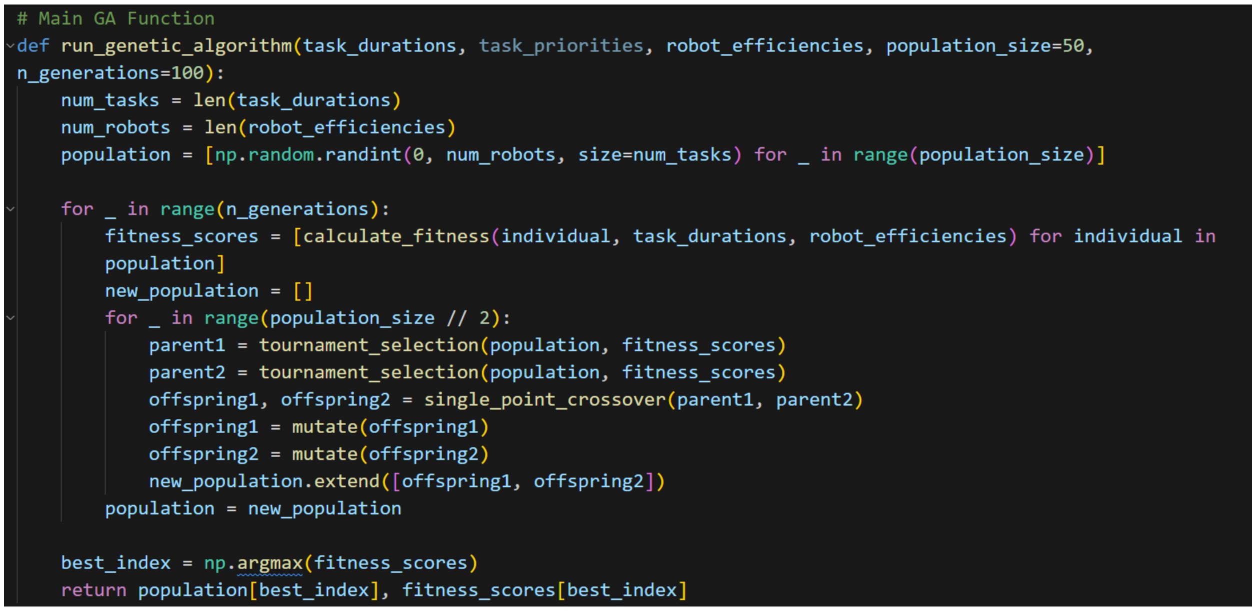


```python
def single_point_crossover(parent1, parent2):
    crossover_point = np.random.randint(1, len(parent1))
    offspring1 = np.concatenate([parent1[:crossover_point], parent2[crossover_point:]])
    offspring2 = np.concatenate([parent2[:crossover_point], parent1[crossover_point:]])
    return offspring1, offspring2

def mutate(individual, mutation_rate=0.1):
    for i in range(len(individual)):
        if np.random.rand() < mutation_rate:
            swap_with = np.random.randint(0, len(individual))
            individual[i], individual[swap_with] = individual[swap_with], individual[i]
    return individual
```

These sections of code define two key genetic operations used in a genetic algorithm: crossover and mutation.

The function single_point_crossover(parent1, parent2) takes two parent individuals and performs single-point crossover.

The function mutate(individual, mutation_rate=0.1) applies mutation to a single individual.



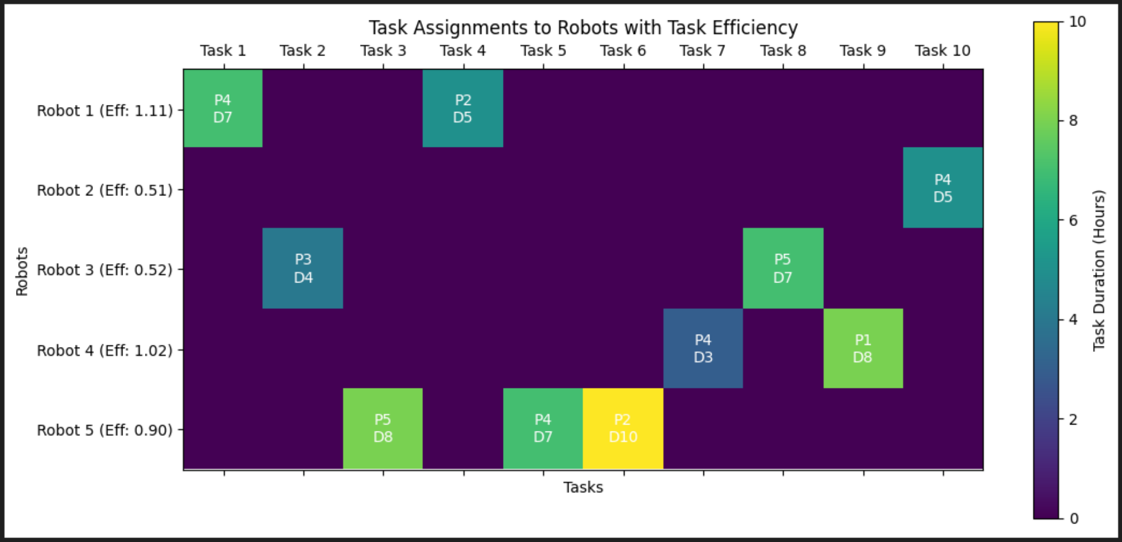
```
# Main GA Function
def run_genetic_algorithm(task_durations, task_priorities, robot_efficiencies, population_size=50,
n_generations=100):
    num_tasks = len(task_durations)
    num_robots = len(robot_efficiencies)
    population = [np.random.randint(0, num_robots, size=num_tasks) for _ in range(population_size)]

    for _ in range(n_generations):
        fitness_scores = [calculate_fitness(individual, task_durations, robot_efficiencies) for individual in
        population]
        new_population = []
        for _ in range(population_size // 2):
            parent1 = tournament_selection(population, fitness_scores)
            parent2 = tournament_selection(population, fitness_scores)
            offspring1, offspring2 = single_point_crossover(parent1, parent2)
            offspring1 = mutate(offspring1)
            offspring2 = mutate(offspring2)
            new_population.extend([offspring1, offspring2])
        population = new_population

    best_index = np.argmax(fitness_scores)
    return population[best_index], fitness_scores[best_index]
```

This function runs a genetic algorithm to evolve a population towards optimal task assignments for robots by using fitness evaluation, tournament selection, crossover, mutation, and identifying the best solution over a set number of generations.

Results:

multiple results based on the car’s movement -

```
Best Solution: [0 2 4 0 4 4 3 2 3 1]
Fitness: 0.053817863311099876
```



Challenges during Implementations:

Several challenges were encountered during the implementation:

- Algorithm Efficiency: Optimizing the GA for computational efficiency while ensuring it accurately reflects the complex dynamics of task assignments.

- Fitness Function Design: Creating a fitness function that accurately balances total production time, workload distribution, and task prioritization.

- Visualization: Developing an intuitive and informative visualization method to clearly represent the optimization results was technically demanding.

Conclusions:

The project highlighted the potential of Genetic Algorithms in optimizing robotic task assignments in dynamic environments. While the initial implementation has addressed key objectives through mock data preparation and the inception of a fitness function, further development is necessary to fully realize all assignment requirements, including comprehensive GA operations and advanced visualization techniques.

The challenges faced underscore the complexity of robotic task optimization but also point to the significant opportunities for efficiency and performance improvements in dynamic production settings. The continued refinement of this GA approach promises valuable insights into the optimization of robotic systems and operational workflows.

Links:

Github -