Assignment: Robot Task Optimization Using Genetic Algorithm

Objective

The goal of this assignment is to develop and implement a Genetic Algorithm (GA) to optimize the assignment of multiple robots to a set of tasks in a dynamic production environment. Your primary objectives are to minimize the total production time, ensure a balanced workload across robots, and prioritize critical tasks effectively. Additionally, you will create a detailed visualization to illustrate the final task assignments, robot efficiencies, and task priorities.

Assignment Details

1. **Background**:
    - You have a set of tasks, each with a specified duration and priority.
    - A pool of robots is available, each with a unique efficiency factor.
    - The production environment is dynamic, with tasks and priorities potentially changing over time.
2. **Tasks**:
    - **Data Preparation**: Generate mock data for tasks (including durations and priorities) and robots (including efficiency factors).

      Sample Code:

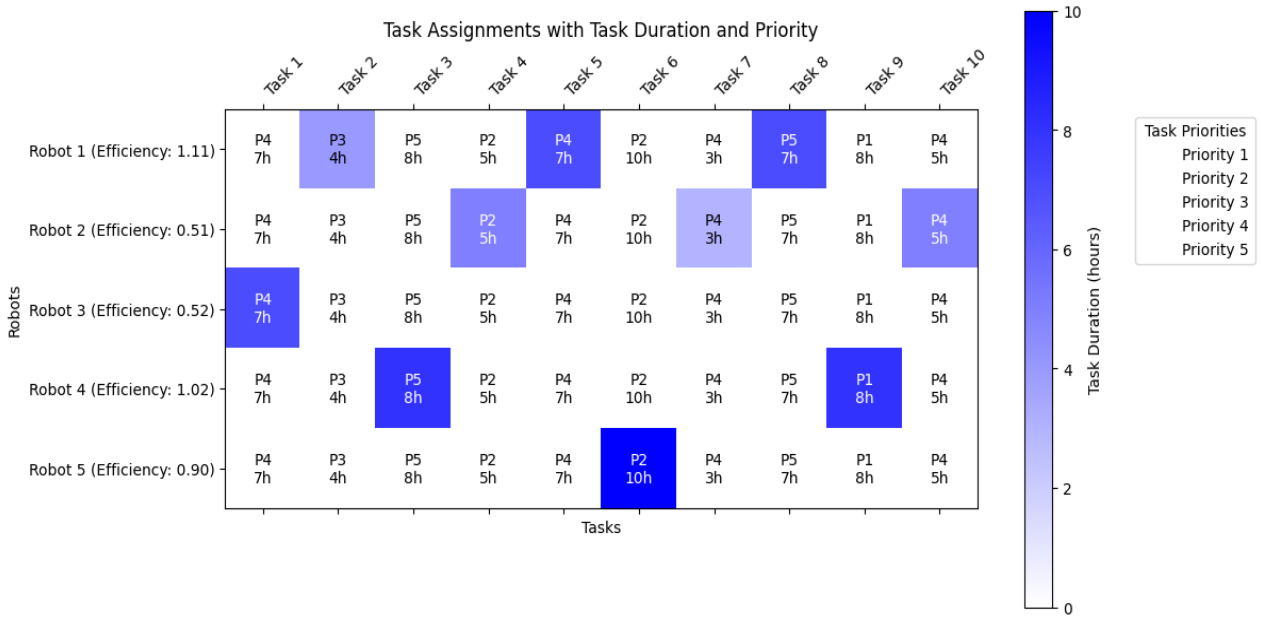
```
# Generate mock data
task_durations = np.random.randint(1, 11, size=10)  # Task durations
task_priorities = np.random.randint(1, 6, size=10)  # Task priorities
robot_efficiencies = np.random.uniform(0.5, 1.5, size=5) # Robot
efficiencies
```

    - **GA Implementation**: Implement a Genetic Algorithm to optimize task assignments considering task duration, robot efficiency, and task priority.

      Sample Code:

```
# Initialize GA parameters
population_size = 50
n_generations = 100
population = [np.random.randint(0, len(robot_efficiencies),
size=len(task_durations)) for _ in range(population_size)]
```

    - **Visualization**: Create a grid visualization of the task assignments highlighting key information.



3. **Genetic Algorithm Components**:

   - **Individual Representation**: Represent each potential solution as a vector where each element indicates the robot assigned to each task.

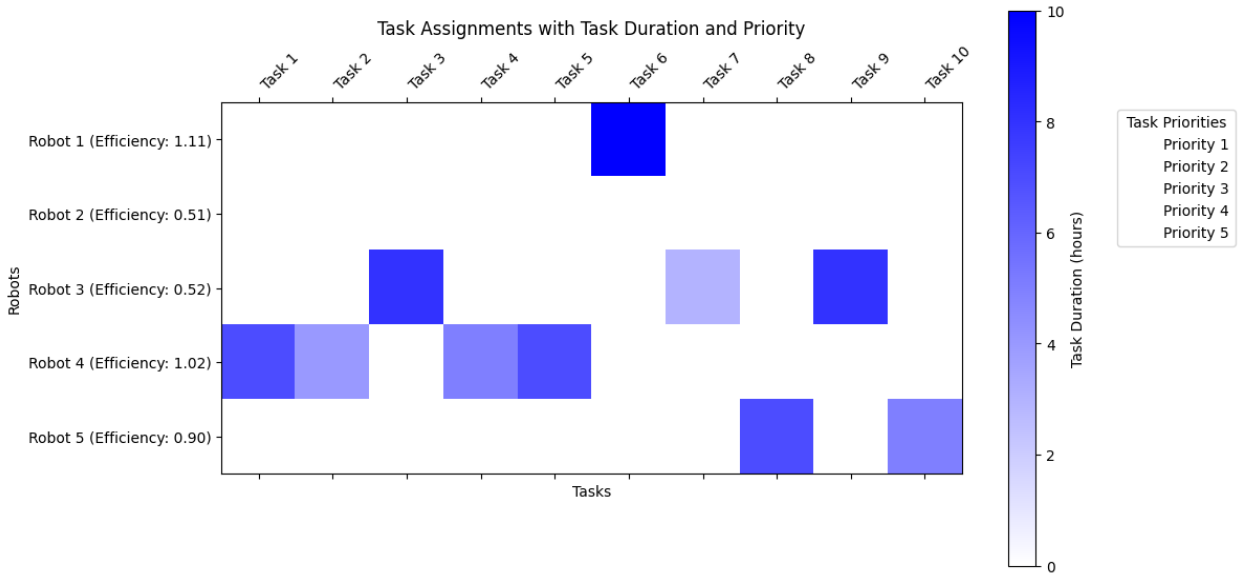
     Individual I is represented as a vector of N integers, where N is the number of tasks, and each integer I_n (where $1 \le n \le N$) corresponds to the ID of the robot assigned to task n.

     $$I=[r_1,r_2,\ldots,r_N]$$

   - **Fitness Function**:
     - Calculate the total production time, *Ttotal*, as the maximum time taken by any robot based on its assigned tasks and efficiency.
     - Compute workload balance, *B*, as the standard deviation of the total times across all robots.
     - Define the fitness function, *F*, to minimize both *Ttotal* and *B*, incorporating task priorities.
     - **Check the part fitness function calculations.**

   - **Selection, Crossover, and Mutation**:
     - Implement these genetic operations to evolve your population towards optimal solutions.
     - **Check the individual sections.**

4. **Visualization**:
    - Create a grid where each row represents a robot and each column represents a task.
    - Use color intensity to indicate task duration, with annotations for significant durations.
    - Annotate each row with the robot's efficiency and each column with the task's priority.
    - Check the example code section where the code is partially done.
    - **Annotate each cell with task priority and duration**



5. **Analysis and Report**:
    - Analyze how robot efficiency and task priority influenced the GA's optimization process.
    - Discuss the workload distribution among robots and identify any potential for further optimization.

Requirements

- **Python Notebook**: Implement the GA and visualization in Python. Use libraries such as NumPy for calculations and Matplotlib for visualization.

  Within your repository, create a folder named `Assignment_2` to specifically house all the files related to this assignment.

- **Report**: Include a detailed report with your submission, covering your approach, implementation details, challenges faced, and a comprehensive analysis of the results. Provide insights into how the GA helped in optimizing task assignments and the implications of your findings.

Evaluation Criteria

- **Correctness and Efficiency of the GA**: The implemented GA should correctly optimize task assignments according to the given objectives. Efficiency in terms of computational resources and time will also be considered.
- **Quality and Clarity of Visualization**: The visualization should clearly and accurately represent the optimized task assignments, including the additional details specified (robot efficiency, task priority, and task duration).
- **Depth of Analysis**: The report should provide a deep analysis of the optimization process, the impact of robot efficiencies and task priorities on the outcomes, and a critical evaluation of the workload distribution among robots.

Fitness Function Calculation

The fitness function aims to minimize the total production time while ensuring a balanced workload across robots and prioritizing critical tasks. It can be decomposed into several components:

1. Total Production Time (T_{total})

The total production time is determined by the robot that finishes last, taking into account the efficiency of each robot. For each robot r, the time T_r spent on its assigned tasks is the sum of the durations of these tasks divided by the robot's efficiency. The total production time is the maximum time any robot takes to complete its tasks.

$$T_r = \sum_{n \in tasks(r)} \frac{D_n X P_n}{E_r}$$

$$T_{total} = \max(T_1, T_2, \dots, T_R)$$

where:

- $tasks(r)$ is the set of tasks assigned to robot r,
- D_n is the duration of task n,
- P_n is the priority weight of task n,
- E_r is the efficiency of robot r,
- R is the total number of robots.

2. Workload Balance (B)

The workload balance penalizes uneven distribution of work among robots. It can be quantified using the standard deviation of the total times across all robots.

$$B = \sigma(T_1, T_2, \dots, T_R)$$

where σ is the standard deviation.

3. Fitness Function (F)

The fitness function combines T_{total} and B to evaluate the quality of an assignment. Since we aim to minimize both the total production time and the imbalance in workload, a simple approach could be to take the inverse of their sum.

$$F(I) = T_{total} + B$$

Selection

The selection process is crucial for guiding the GA towards optimal solutions by choosing individuals from the current population to breed the next generation. Two common methods are:

1. **Tournament Selection**:
    - Randomly select a small subset of individuals from the population to form a "tournament."
    - The individual with the highest fitness within this group is selected as a parent for crossover.
    - Repeat the process to select another parent.
2. **Roulette Wheel Selection**:
    - Assign each individual a selection probability proportional to its fitness relative to the total fitness of the population.
    - Spin the roulette wheel, where the chance of landing on an individual corresponds to its selection probability.

Code Help:

1. **Tournament Selection Process**:
    - **Random Selection**: For each tournament, select `k` individuals randomly from the population. The value of `k` can be adjusted based on how selective you want the process to be. A larger `k` increases competition and might favor stronger individuals more heavily, while a smaller `k` keeps selection pressure lower and maintains more diversity.
    - **Determine Winner**: Evaluate the fitness of the `k` selected individuals. The one with the **best fitness (e.g., the lowest total time or the best balance)** wins the tournament and is selected for breeding. In robot task optimization, this step effectively selects task assignments that are closer to the optimization objectives for propagation.
    - **Reproduction Preparation**: The winners of the tournaments are then used in the crossover and mutation steps to generate new individuals (new task assignments), which will constitute the next generation in the population.
2. **Advantages in Robot Task Optimization**:
    - **Diversity**: Tournament selection helps maintain genetic diversity within the population by giving a chance to a variety of individuals to be selected.

This is crucial in dynamic environments where task conditions or robot capabilities might change.
- **Efficiency**: This method can be more efficient computationally than some alternatives, as it does not require sorting the entire population by fitness or calculating cumulative probabilities, making it well-suited for large-scale or complex optimization problems.
- **Flexibility**: By adjusting the tournament size `k`, you can easily control the selection pressure, making the algorithm adaptable to various optimization landscapes and goals in robot task assignment.

Crossover (Recombination)

Crossover is a genetic operation used to combine the genetic information of two parents to generate new offspring. It is pivotal for introducing new genetic combinations into the population.

1. **Single-Point Crossover**:
   - Randomly choose a crossover point on the parent individuals' genomes.
   - Create offspring by swapping all genes (task assignments in our context) after this point between the two parents.
   - **Example**: If our **crossover point is 3** and we have two parents `[A, B, C, D, E]` and `[V, W, X, Y, Z]`, the offspring would be `[A, B, C, Y, Z]` and `[V, W, X, D, E]`.

Mutation

Mutation introduces random genetic variations, providing new genetic structures for exploration and helping the algorithm escape local optima.

1. **Task Swapping**:
   - Randomly select two tasks within an individual's assignment list and swap their assigned robots.
   - **Implementation Note**: Ensure that the mutation rate is kept relatively low to prevent excessive randomization of the individuals, which could disrupt the convergence of the GA.

Implementation Tips

- **Parameter Tuning**: The effectiveness of the GA heavily depends on the careful tuning of parameters such as the size of the tournament in tournament selection, the mutation rate, and the method used for crossover. Experiment with different settings to find the optimal configuration for your specific problem.

- **Balancing Exploration and Exploitation**: Ensure your selection, crossover, and mutation methods maintain a healthy balance between exploring new areas of the solution space (exploration) and refining the best solutions found so far (exploitation). This balance is key to avoiding premature convergence on suboptimal solutions and ensuring the GA can navigate towards the global optimum over time.

- **Check the code example notebook.**