Complex Engineering Problem Report

Artificial Intelligence

Third Year-Computer and Information Systems Engineering

Batch: 2022

Group Members:

Jaweria Anwer. .... (CS-22057)

Shiza Ejaz .... (CS-22060)

Fatima Mehdvi .... (CS-22065)

Submitted to: Miss Hameeza

Submission Date: 22/11/2024

Table of Contents

Abstract:

This project presents a graphical application for optimizing job scheduling using a genetic algorithm. The application facilitates task-to-machine assignments to minimize the makespan—the maximum time required to complete all tasks. Users can input task durations, the number of tasks and machines, population size, the number of generations, and mutation rates through an interactive GUI. The genetic algorithm leverages natural selection, crossover, and mutation to evolve an optimal schedule. The project demonstrates the effective application of genetic algorithms to solve real-world scheduling problems, balancing computational efficiency and practicality in scheduling tasks across multiple machine

Introduction/Problem Statement:

Efficient job scheduling is critical in various industries to minimize delays and improve resource utilization. Assigning tasks to machines with varying workloads can be challenging, particularly as the number of tasks and machines increases. Poor scheduling can lead to longer completion times and inefficiency.

This project aims to address these challenges using a genetic algorithm, a heuristic optimization technique inspired by natural evolution. By iteratively refining a population of task-to-machine schedules, the algorithm identifies an optimal schedule with minimal makespan. The user-friendly GUI allows users to explore scheduling optimization interactively, making the project accessible and practical for real-world applications.

Theoretical details of the model:

Genetic Algorithm Components

Population Initialization: Randomly generates a set of initial schedules, where each schedule assigns tasks to machines.

Fitness Function: Evaluates the schedules based on makespan, encouraging solutions with evenly distributed workloads across machines.

Selection: Uses roulette wheel selection to probabilistically choose parent schedules based on fitness.

Crossover: Combines pairs of parent schedules to create offspring, introducing new combinations of task-to-machine assignments.

Mutation: Randomly alters some task assignments in offspring schedules to maintain diversity and explore new solutions.

Implementation Details

Fitness Function: Calculates the makespan (maximum completion time across machines).

Termination Criteria: The algorithm halts after a fixed number of generations or when an optimal schedule (smallest possible makespan) is found.

Input Parameters

Number of Tasks: Total tasks to be scheduled.

Task Durations: The time required for each task.

Number of Machines: Machines available for scheduling.

Population Size: The number of schedules in each generation.

Generations: The number of iterations for algorithm evolution.

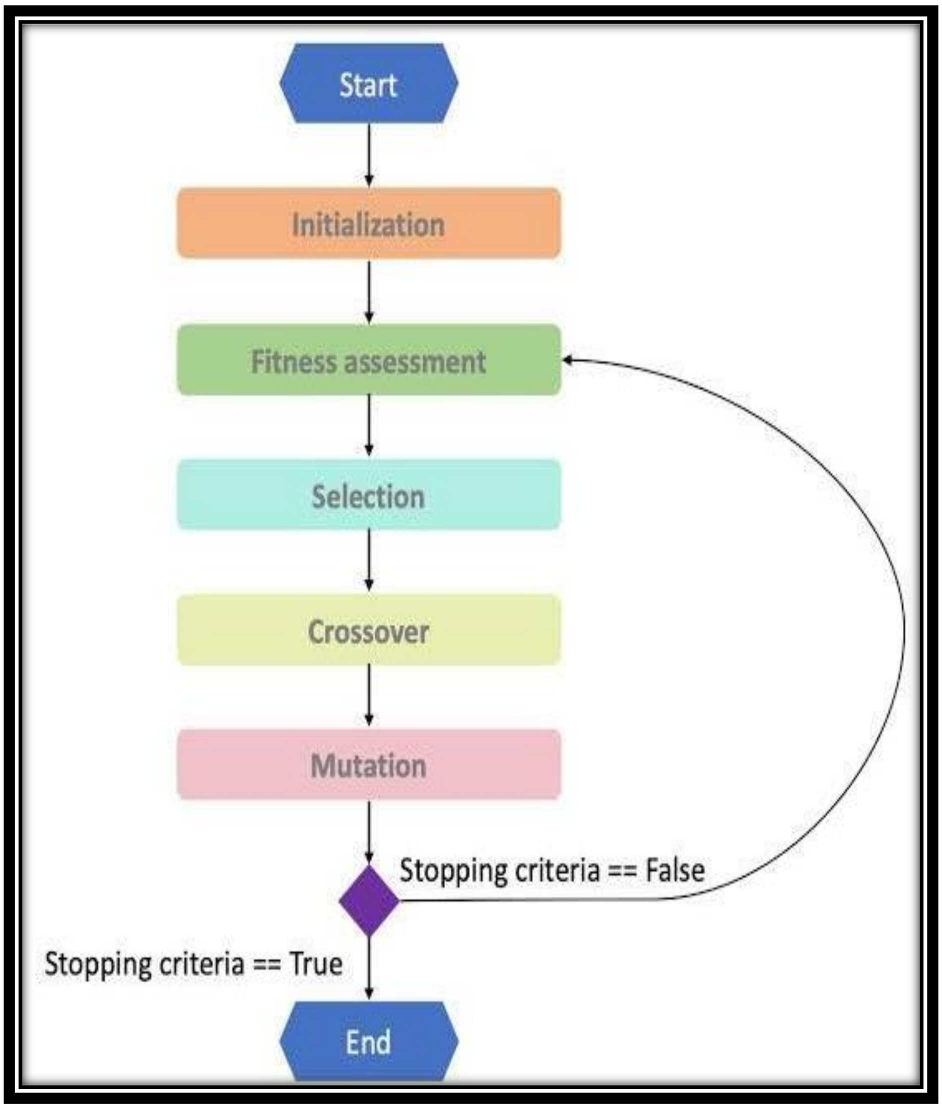
Mutation Rate: Probability of altering a task assignment in a schedule

Program for implementation:

The project integrates the genetic algorithm with a Tkinter-based GUI, enabling users to input parameters and visualize results. The application iteratively optimizes schedules and displays the final makespan and task-to-machine assignments.

Key functions include:

initialize_population: Generates initial random schedules.
fitness_function: Evaluates schedules based on makespan.
selection: Chooses parent schedules for breeding.
crossover: Produces offspring schedules.
mutate: Ensures diversity in the population.



Simulation Results:

Upon running the application, users can observe:

- The evolution of schedules over generations.
- The reduction in makespan, as the algorithm identifies optimal or near-optimal solutions.
- A detailed breakdown of task assignments to machines in the optimal schedule.

Example Results:

Input:

Tasks: 6, Machines: 3

Task Durations: [4, 7, 2, 6, 3, 8]

Population Size: 10

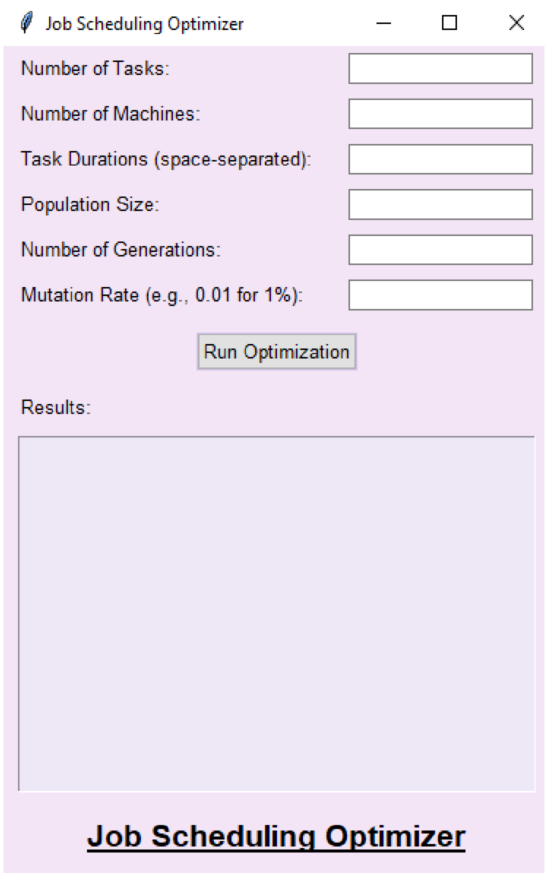
Generations: 50

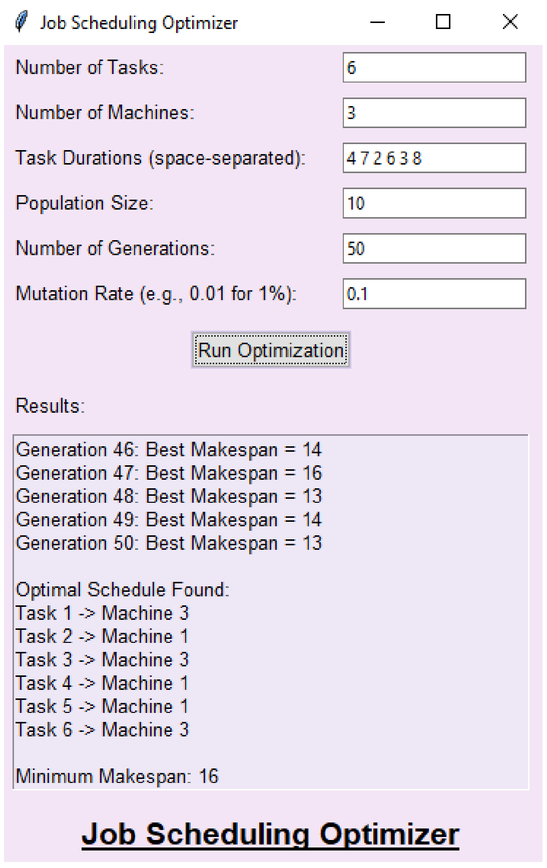
Mutation Rate: 0.1

Output:

Optimal Schedule: Task 1 -> Machine 3,Task 2 -> Machine 1,Task 3 -> Machine 3,Task 4 -> Machine 1,Task 5 -> Machine 1,Task 6 -> Machine 3

Minimum Makespan: 12





Job Scheduling Optimizer

Number of Tasks: 6

Number of Machines: 3

Task Durations (space-separated): 4 7 2 6 3 8

Population Size: 10

Number of Generations: 50

Mutation Rate (e.g., 0.01 for 1%): 0.1

Results:

```
Generation 46: Best Makespan = 14
Generation 47: Best Makespan = 16
Generation 48: Best Makespan = 13
Generation 49: Best Makespan = 14
Generation 50: Best Makespan = 13

Optimal Schedule Found:
Task 1 -> Machine 3
Task 2 -> Machine 1
Task 3 -> Machine 3
Task 4 -> Machine 1
Task 5 -> Machine 1
Task 6 -> Machine 3

Minimum Makespan: 16
```

Job Scheduling Optimizer

Conclusion:

This genetic algorithm-based application effectively minimizes makespan in job scheduling problems, demonstrating the power of heuristic optimization methods. The interactive GUI enhances usability, making the tool accessible for educational and practical purposes.

Future Enhancements:

- Incorporate constraints such as machine-specific task capabilities.
- Optimize for energy consumption or cost in addition to makespan.
- Extend to dynamic scheduling for real-time task assignments.