Faculty of Engineering & Technology

Electrical & Computer Engineering Department

ARTIFICIAL INTELLIGENC , ENCS3340

Project #1

Optimizing Job Shop Scheduling in a Manufacturing Plant using Genetic Algorithm

---

Prepared by:

Baraa Nasar 1210880

Leen Sweilem 1182728

Instructor: Dr.Aziz Qaroush , Dr. Ismail Khater

Section: 2 , 3

Date: 5/20/2024

Table of Contents

The genetic Algorithm In general:

A genetic algorithm is an adaptive heuristic search algorithm inspired by "Darwin's theory of evolution in Nature." It is used to solve optimization problems in machine learning. It is one of the important algorithms as it helps solve complex problems that would take a long time to solve [1].

The genetic algorithm Consists of:

- **Population:** Population is the subset of all possible or probable solutions, which can solve the given problem.
- **Chromosomes:** A chromosome is one of the solutions in the population for the given problem, and the collection of gene generate a chromosome.
- **Gene:** A chromosome is divided into a different gene, or it is an element of the chromosome.
- **Fitness Function:** The fitness function is used to determine the individual's fitness level in the population. It means the ability of an individual to compete with other individuals. In every iteration, individuals are evaluated based on their fitness function.
- **Genetic Operators:** In a genetic algorithm, the best individual mate to regenerate offspring better than parents. Here genetic operators play a role in changing the genetic composition of the next generation.
- **Selection:** After calculating the fitness of every existent in the population, a selection process is used to determine which of the individualities in the population will get to reproduce and produce the seed that will form the coming generation.

How Genetic Algorithm Work?

The genetic algorithm works on the evolutionary generational cycle to generate high-quality solutions. These algorithms use different operations that either enhance or replace the population to give an improved fit solution [1].

It basically involves five phases to solve the complex optimization problems, which are given as below:

- Initialization
- Fitness Assignment
- Selection
- Reproduction
- Termination

Initialization: The process of a genetic algorithm starts by generating the set of individuals, which is called population. Here each individual is the solution for the given problem. An individual contains or is characterized by a set of parameters called Genes. Genes are combined into a string and generate chromosomes, which is the solution to the problem. One of the most popular techniques for initialization is the use of random binary strings [1].

Fitness Assignment: Fitness function is used to determine how fit an individual is? It means the ability of an individual to compete with other individuals. In every iteration, individuals are evaluated based on their fitness function. The fitness function provides a fitness score to each individual. This score further determines the probability of being selected for reproduction. The high the fitness score, the more chances of getting selected for reproduction [1].

Selection: The selection phase involves the selection of individuals for the reproduction of offspring. All the selected individuals are then arranged in a pair of two to increase reproduction. Then these individuals transfer their genes to the next generation [1].

Reproduction: After the selection process, the creation of a child occurs in the reproduction step. In this step, the genetic algorithm uses two variation operators that are applied to the parent population. The two operators involved in the reproduction phase are given below [1]:

- **Crossover:** The crossover plays a most significant role in the reproduction phase of the genetic algorithm. In this process, a crossover point is selected at random within the genes. Then the crossover operator swaps genetic information of two parents from the current generation to produce a new individual representing the offspring [1].

- **Mutation:** The mutation operator inserts random genes in the offspring (new child) to maintain the diversity in the population. It can be done by flipping some bits in the chromosomes. Mutation helps in solving the issue of premature convergence and enhances diversification [1].

Termination: After the reproduction phase, a stopping criterion is applied as a base for termination. The algorithm terminates after the threshold fitness solution is reached. It will identify the final solution as the best solution in the population [1].

The problem Formalization

Scheduling several tasks, each consisting of a series of activities, on a group of machines is a complicated optimization issue known as "job shop scheduling." Only one operation may be processed by each machine at a time, and those operations have to happen in a certain order. Taking machine capabilities and task dependencies into account, the objective is to ascertain the best sequence and timing for each work in order to minimize the total production time or maximize throughput.

Input:

- A list of jobs, where each job is defined as a sequence of operations.
- The number of available machines.

Output:

- A schedule for each machine that depicts the start and end time for each process and to which job it belongs. This is visualized using a Gantt Chart.

Genetic Algorithm

Natural selection serves as the inspiration for the optimization method known as a genetic algorithm. Over several generations, it evolves a population of solutions towards better solutions through mechanisms including crossover, mutation, and selection.

Entire Chromosome Illustration

A chromosome in our implementation is a timetable. Every work is a series of processes, and every job is contained within a single chromosome.

Function of Fitness

The fitness function assesses the quality of a schedule. The make span, or maximum completion time across all machines, is a useful metric for determining a schedule's fitness. Better schedules are indicated by lower make span values.

```
def calculate_fitness(self):
    self.machine_schedules = {machine: [] for machine in self.machines}
    time = {machine: 0 for machine in self.machines}
    for job in self.jobs:
        current_time = 0
        for operation in job.operations:
            machine = operation.machine_id
            start_time = max(current_time, time[machine])
```

```
            end_time = start_time + operation.processing_time
            self.machine_schedules[machine].append((job.job_id, start_time, end_time))
            time[machine] = end_time
            current_time = end_time
    self.fitness = max(time.values())
```

Crossover

A kid schedule is created by combining portions of two parent schedules using the crossover function. Each job's operations are divided between the two parents at a randomly selected split point.

```
def crossover(self, other):
    child = Schedule(self.jobs, self.machines)
    for job in self.jobs:
        split_point = random.randint(1, len(job.operations) - 1)
        new_ops = job.operations[:split_point] +
other.jobs[self.jobs.index(job)].operations[split_point:]
        child.jobs[self.jobs.index(job)].operations = new_ops
    child.calculate_fitness()
    return child
```

Mutation

The mutation function introduces variability into the population by randomly shuffling the operations of each job with a certain probability.

```
def mutate(self):
    for job in self.jobs:
        if random.random() < 0.1:
            random.shuffle(job.operations)
    self.calculate_fitness()
```

Implementation

Our implementation uses Python and the Tkinter library for the graphical user interface. We also utilize Matplotlib to visualize the Gantt Chart.

GUI

The GUI allows the user to input machine IDs and job descriptions. Upon running the scheduler, it displays the optimized schedule in a Gantt Chart format.

Main Components

1. **Operation Class**: Represents an operation in a job.
2. **Job Class**: Represents a job consisting of multiple operations.
3. **Schedule Class**: Represents a schedule for all jobs and their operations on various machines.
4. **Genetic Algorithm Function**: Evolves the population of schedules.
5. **JobShopSchedulerApp Class**: Manages the GUI and user interactions.

Test Cases

To validate our implementation, we tested the system with various input scenarios. Below are some examples:

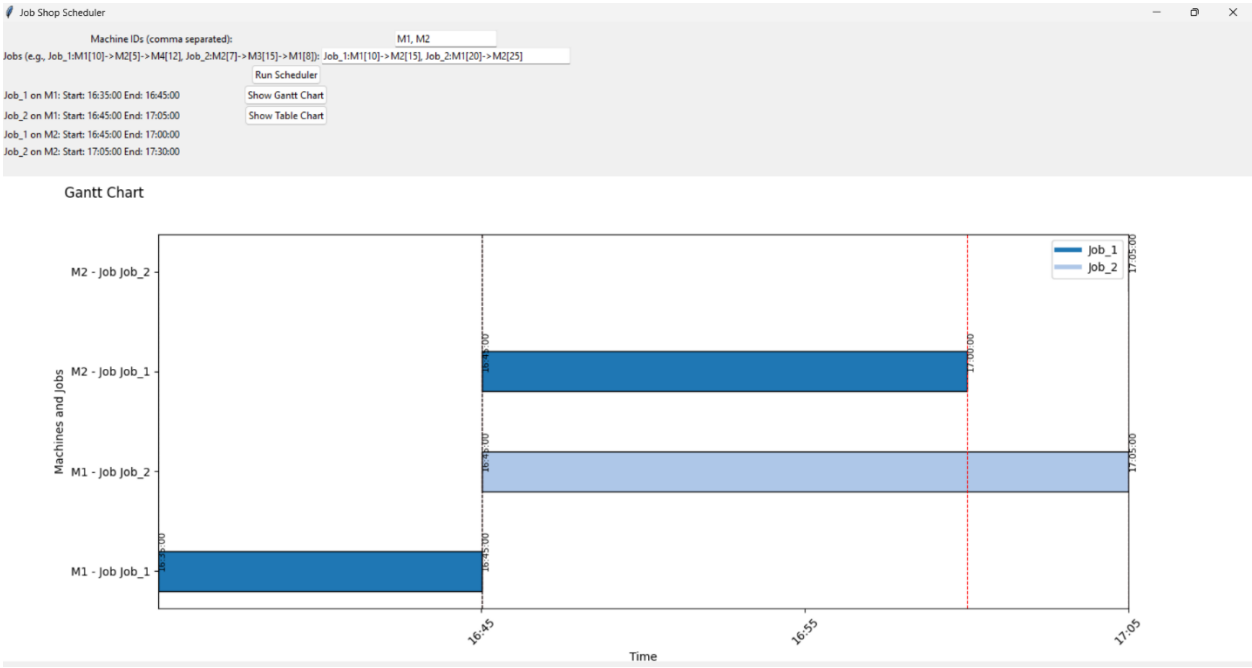
Test Case 1:

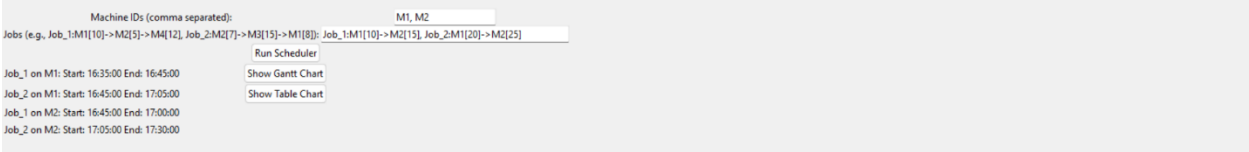
Input:

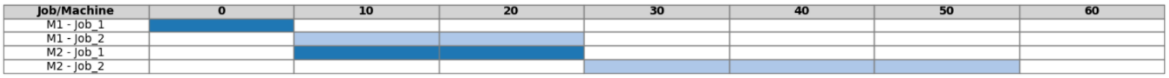
- Machine IDs: **M1, M2**
- Jobs: **Job_1:M1[10]->M2[15], Job_2:M1[20]->M2[25]**

Output:

- Gantt chart displaying the optimized schedule with minimized make span.







Job/Machine	0	10	20	30	40	50	60
M1 - job_1							
M1 - job_2							
M2 - job_1							
M2 - job_2							

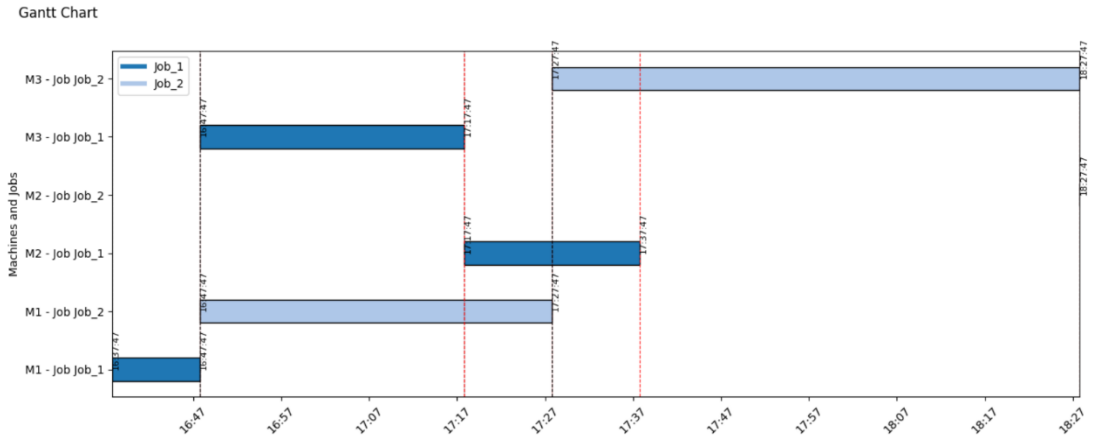
Test Case 2:

Input:

- Machine IDs: **M1, M2, M3, M4**
- Jobs: **Job_1:M1[10]->M2[5]->M4[12], Job_2:M2[7]->M3[15]->M1[8]**

Output:

- Gantt chart displaying the optimized schedule with minimized make span.



Job/Machin	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160
M1 - Job_1																	
M1 - Job_2																	
M2 - Job_1																	
M2 - Job_2																	
M3 - Job_1																	
M3 - Job_2																	

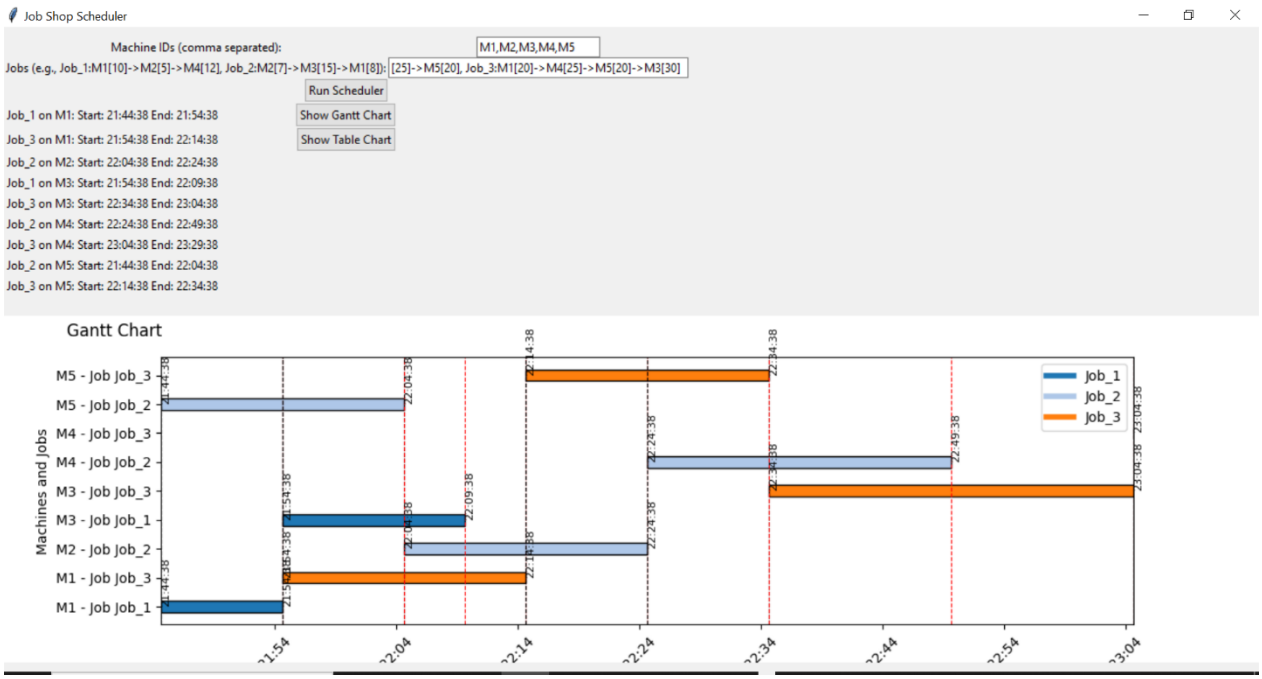
Test Case 3:

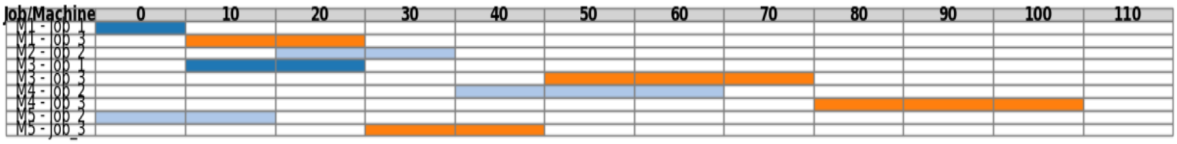
Input:

- Machine IDs: **M1, M2, M3, M4, M5**
- Jobs: **Job_1:M1[10]->M3[15], Job_2:M2[20]->M4[25]->M5[20], Job_3:M1[20] ->M4[25]->M5[20]->M3[30]**

Output:

- Gantt chart displaying the optimized schedule with minimized make span.





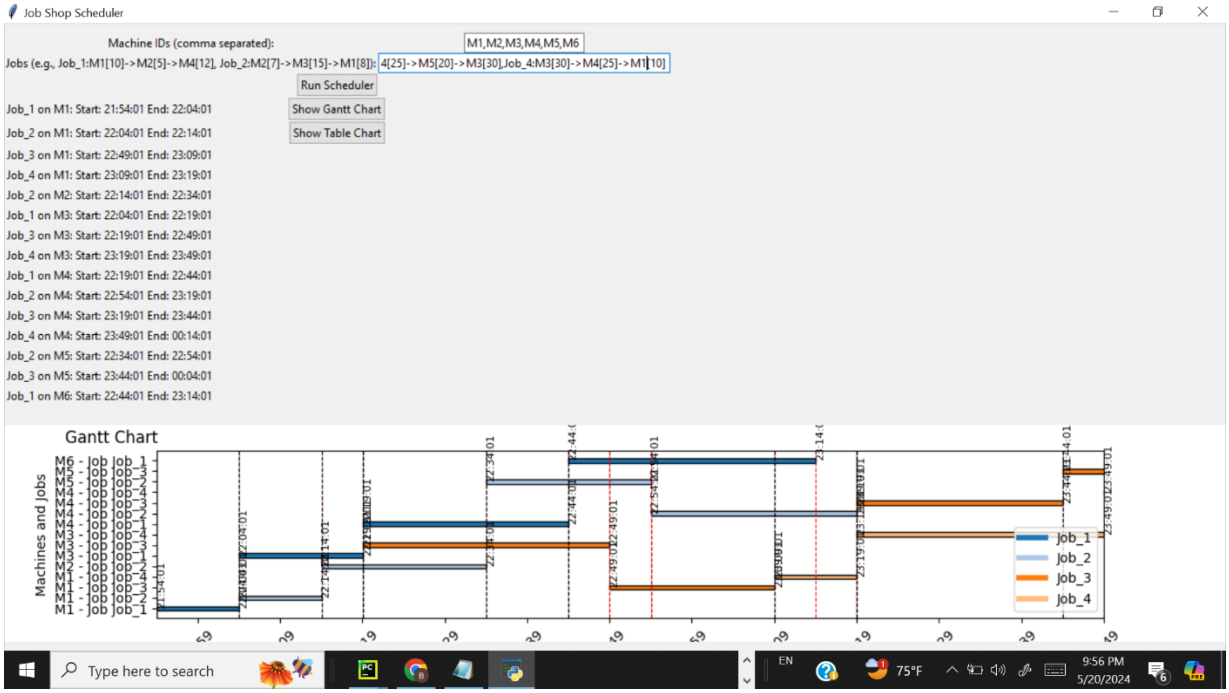
Test Case 4:

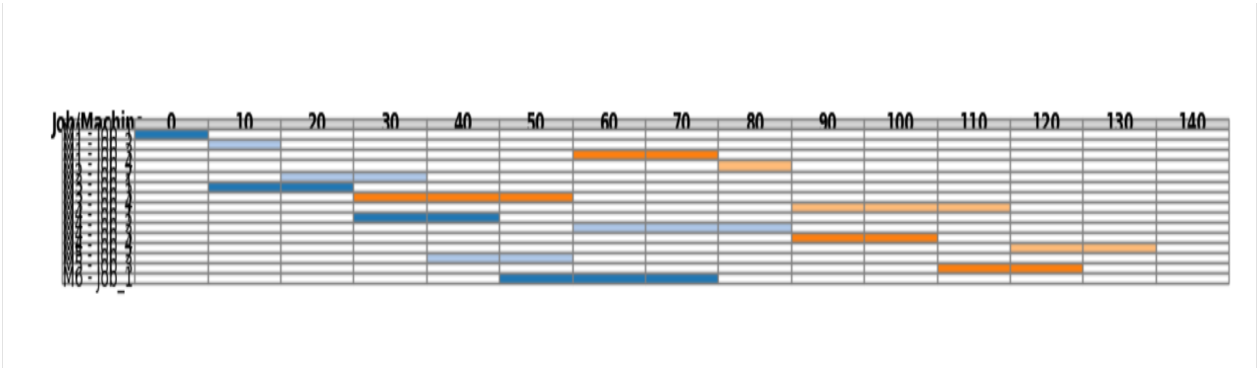
Input:

- Machine IDs: **M1,M2,M3,M4,M5,M6**
- Jobs: **Job_1:M1[10]->M3[15]->M6[30]->M4[25], Job_2:M2[20]->M4[25]->M5[20]->M1[10], Job_3:M1[20]->M4[25]->M5[20]->M3[30],Job_4:M3[30]->M4[25]->M1[10]**

Output:

- Gantt chart displaying the optimized schedule with minimized make span.





Conclusion

The research effectively optimizes job shop scheduling through the use of a genetic algorithm. Users may input machine and work data into the system, which then uses a genetic algorithm to generate the best schedule and displays the outcome on a Gantt chart. Robustness is ensured by the integrated error handling, and its graphical interface facilitates ease of use.

References

[1] Genetic algorithm in machine learning . (2024, 20 5). Retrieved from Javatpoint: https://www.javatpoint.com/genetic-algorithm-in-machine-learning .