Electrical and Computer Engineering Department

ENCS3340

Artificial Intelligence

Project #1

# Optimizing Job Shop Scheduling in a Manufacturing Plant Using Genetic Algorithm

Prepared by:

| | | |
|---|---|---|
| Kareem Qutob | 1211756 | section: 4 |
| Diaa  Badaha | 1210478 | section: 2 |

Date: 2024/5/20

## Table of Contents

## Problem formulation

### Problem Definition

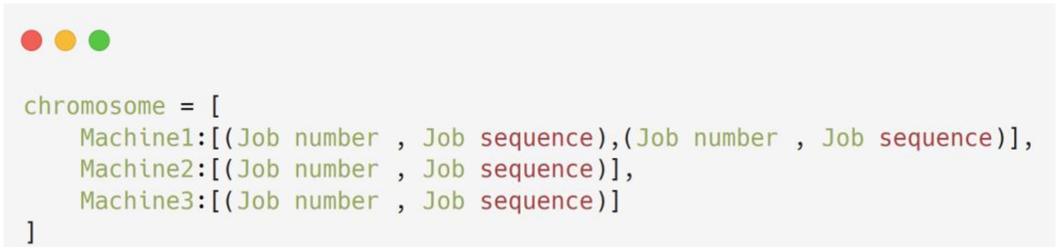
The goal of the job shop scheduling problem is to effectively distribute work among several machines in a manufacturing facility in order to reduce production time or increase throughput while taking machine capabilities and job dependencies into account.

### Objective Function

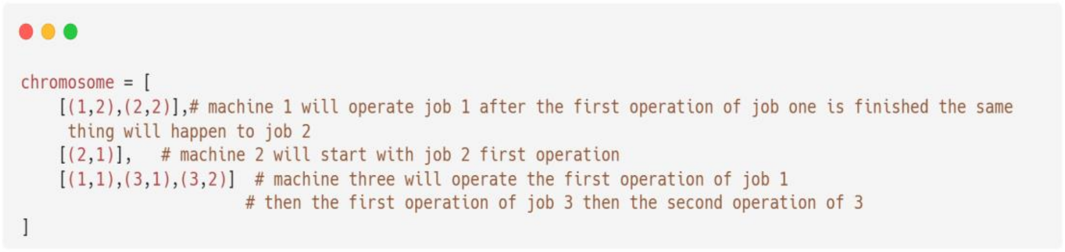
Optimize the throughput of executing a series of jobs on several machines while minimizing the execution time (makespan).

### Chromosome Encoding

To guarantee that tasks are completed in the correct order, the chromosome is encoded as a list of lists, where each list represents a machine. Each machine is represented by a list of tuples, where each tuple represents a job number and its sequence in the job.



```
chromosome = [
    Machine1:[(Job number , Job sequence),(Job number , Job sequence)],
    Machine2:[(Job number , Job sequence)],
    Machine3:[(Job number , Job sequence)]
]
```



```
chromosome = [
    [(1,2),(2,2)],# machine 1 will operate job 1 after the first operation of job one is finished the same
     thing will happen to job 2
    [(2,1)],   # machine 2 will start with job 2 first operation
    [(1,1),(3,1),(3,2)]  # machine three will operate the first operation of job 1
                           # then the first operation of job 3 then the second operation of 3
]
```

Due to the predetermined sequence for each machine, this encoding will facilitate the generation of population and help in the Gantt chart portion.

## Initial Population

To generate the population for our job shop scheduling problem, we employed two approaches based on the number of operations involved, assuming a minimum of two machines:

1. Small Problems:

For problems with fewer than 6 operations, the maximum number of permutations is 120 (5!). In this case, we generate the population by considering every possible permutation. This comprehensive approach ensures that we explore all potential solutions, thereby guaranteeing the identification of the optimal solution.

2. Large Problems:

For more complex problems with a larger number of operations, the population is generated by first creating an initial solution and then randomly shuffling this solution. Care is taken during the shuffling process to maintain the dependencies of the jobs, ensuring that the integrity of each job's sequence is preserved. This method helps in efficiently exploring a broader solution space without compromising the job dependencies.

The number of operations for large problems is defined through this equation:

$$\boldsymbol{Population\ size} = \mathbf{max}(\mathbf{50}\,,(\boldsymbol{Factor} * \boldsymbol{number\ of\ operations}))$$

The factor will be increased whenever the given solutions are not good enough

## Fitness Function

The fitness function will calculate the execution time for every chromosome by calculating the max time for the machines to finish the jobs :

$$\boldsymbol{fitnessFunction}(\boldsymbol{i}) = \boldsymbol{Max}(\boldsymbol{machine1\ time}\,,\boldsymbol{machine2\ time}\,,\ldots,\boldsymbol{machineN\ time})$$

then to calculate the fitness rate for every chromosome it will divide it by the summation of all chromosomes fitness function results as in the equation below:

$$\%\boldsymbol{Fitness\ i} = \frac{\boldsymbol{fitnessFunction(i)}}{\sum_{n=1}^{Population\ size} \boldsymbol{fitnessFunction(n)}}$$

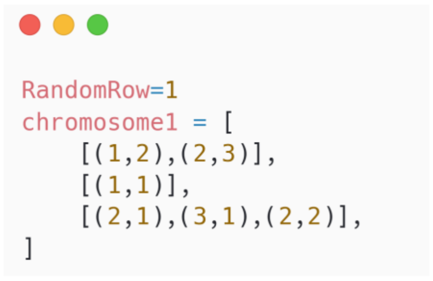
## Parents Selection

In every iteration 2 parents will be chosen randomly but proportional to the fitness score meaning that the lower the score (lower makespan time) the higher the probability to be selected

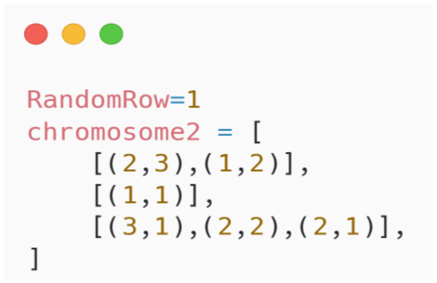
## Cross-Over

The crossover function randomly selects a row within the chromosomes and swaps every row below the randomly chosen row between the two chromosomes, the swapped rows will represent machines scheduling in each chromosome. This approach helps in diversifying the genetic material of the chromosomes and potentially introduces beneficial combinations of genes in the offspring.

For example:

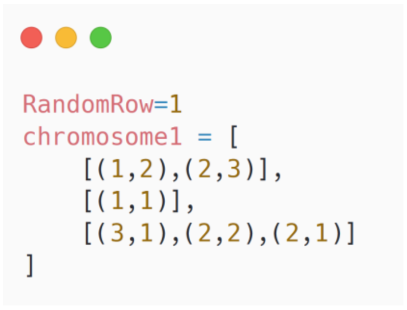


```
RandomRow=1
chromosome1 = [
    [(1,2),(2,3)],
    [(1,1)],
    [(2,1),(3,1),(2,2)],
]
```

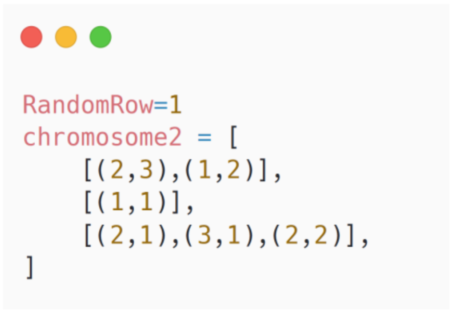


```
RandomRow=1
chromosome2 = [
    [(2,3),(1,2)],
    [(1,1)],
    [(3,1),(2,2),(2,1)],
]
```

After Cross-Over:



```
RandomRow=1
chromosome1 = [
    [(1,2),(2,3)],
    [(1,1)],
    [(3,1),(2,2),(2,1)]
]
```

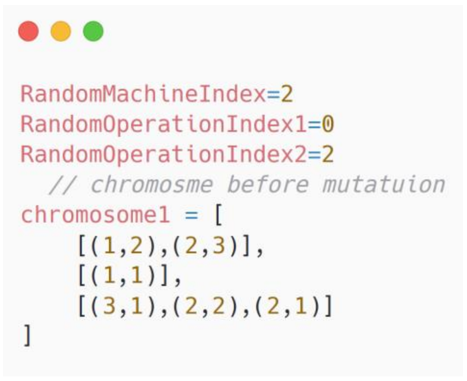


```
RandomRow=1
chromosome2 = [
    [(2,3),(1,2)],
    [(1,1)],
    [(2,1),(3,1),(2,2)],
]
```

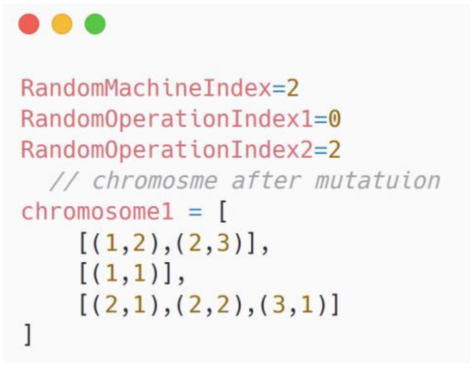
## Mutation

The mutation function randomly selects a machine from a chromosome and then chooses two random operations within it to swap, thereby altering the order of operations scheduled on this machine. mutation plays a crucial role in maintaining genetic diversity, exploring new solutions, and preventing premature convergence, ultimately leading to the discovery of better solutions in the optimization process.

For example :



```
RandomMachineIndex=2
RandomOperationIndex1=0
RandomOperationIndex2=2
  // chromosme before mutatuion
chromosome1 = [
    [(1,2),(2,3)],
    [(1,1)],
    [(3,1),(2,2),(2,1)]
]
```



```
RandomMachineIndex=2
RandomOperationIndex1=0
RandomOperationIndex2=2
  // chromosme after mutatuion
chromosome1 = [
    [(1,2),(2,3)],
    [(1,1)],
    [(2,1),(2,2),(3,1)]
]
```

## Termination Condition

The number of generations will be chosen proportional to the size of the problem the bigger the problem it will have more generations and the program will terminate after the given number of generations is satisfied (note in our project we used an incremental genetic algorithm meaning that every generation is the previous plus the new parents)

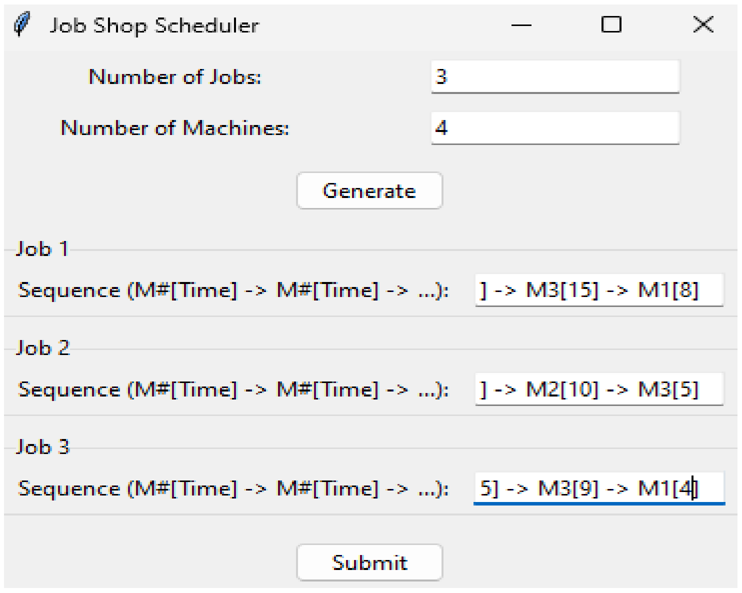
## Test Cases

Test 1:

Job 1 M2[7] -> M3[15] -> M1[8]

Job 2 M1[9] -> M3[6] -> M2[10] -> M3[5]

Job 3 M4[10] -> M2[5] -> M3[9] -> M1[4]



Job Shop Scheduler

Number of Jobs: 3

Number of Machines: 4

Job 1

Sequence (M#[Time] -> M#[Time] -> ...): ] -> M3[15] -> M1[8]

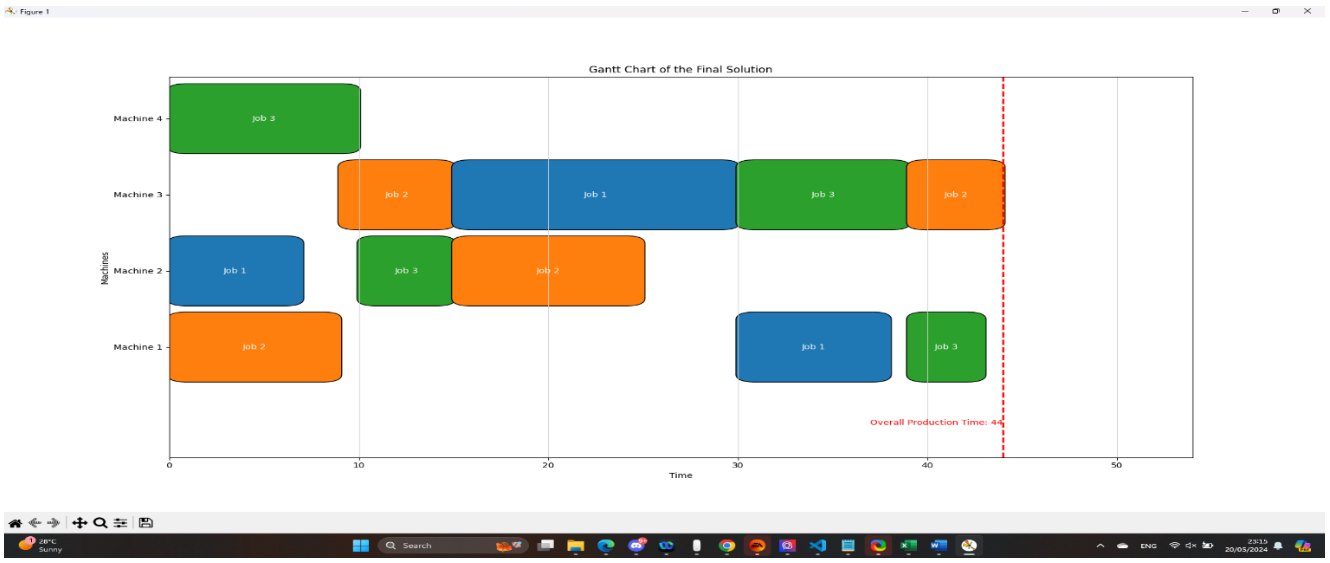
Job 2

Sequence (M#[Time] -> M#[Time] -> ...): ] -> M2[10] -> M3[5]

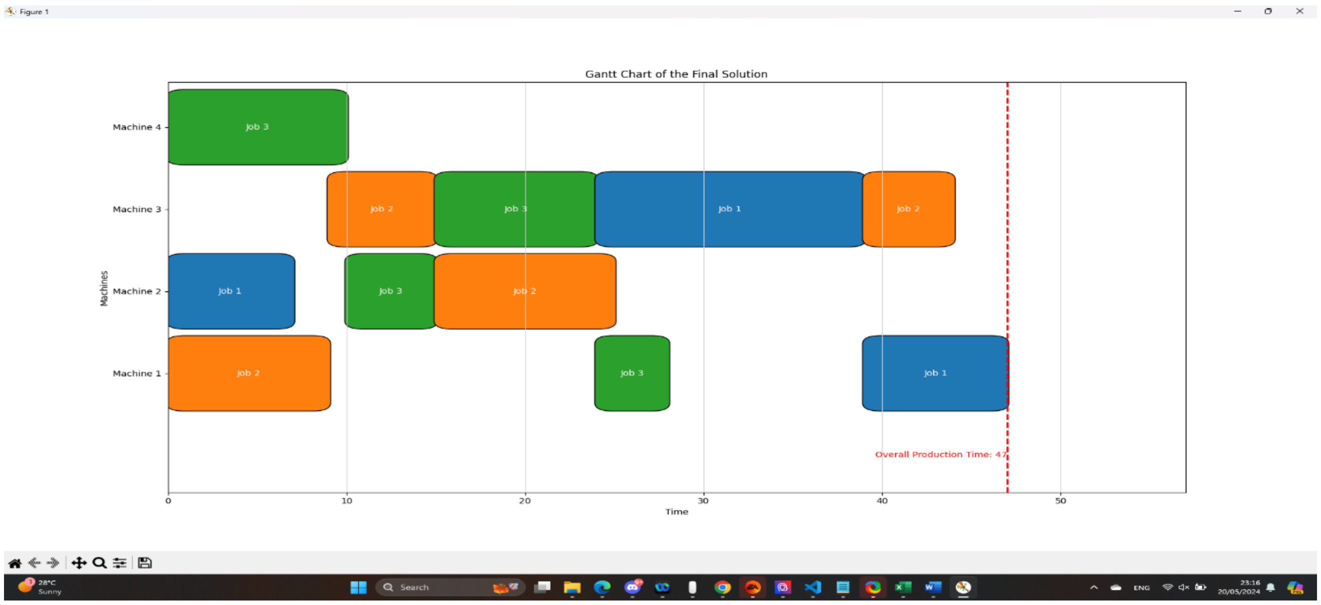
Job 3

Sequence (M#[Time] -> M#[Time] -> ...): 5] -> M3[9] -> M1[4]

Run 1 :



Run2:



Test 2:

Job 1 M2[8] -> M5[4] -> M1[6] -> M7[3] -> M3[5]

Job 2 M4[7] -> M8[3] -> M6[2] -> M1[4] -> M5[6]

Job 3 M3[5] -> M7[8] -> M2[7] -> M6[4] -> M1[6]

Job 4 M5[4] -> M2[7] -> M4[3] -> M8[8] -> M7[2]

Job 5 M1[7] -> M6[4] -> M3[3] -> M2[5] -> M5[6]

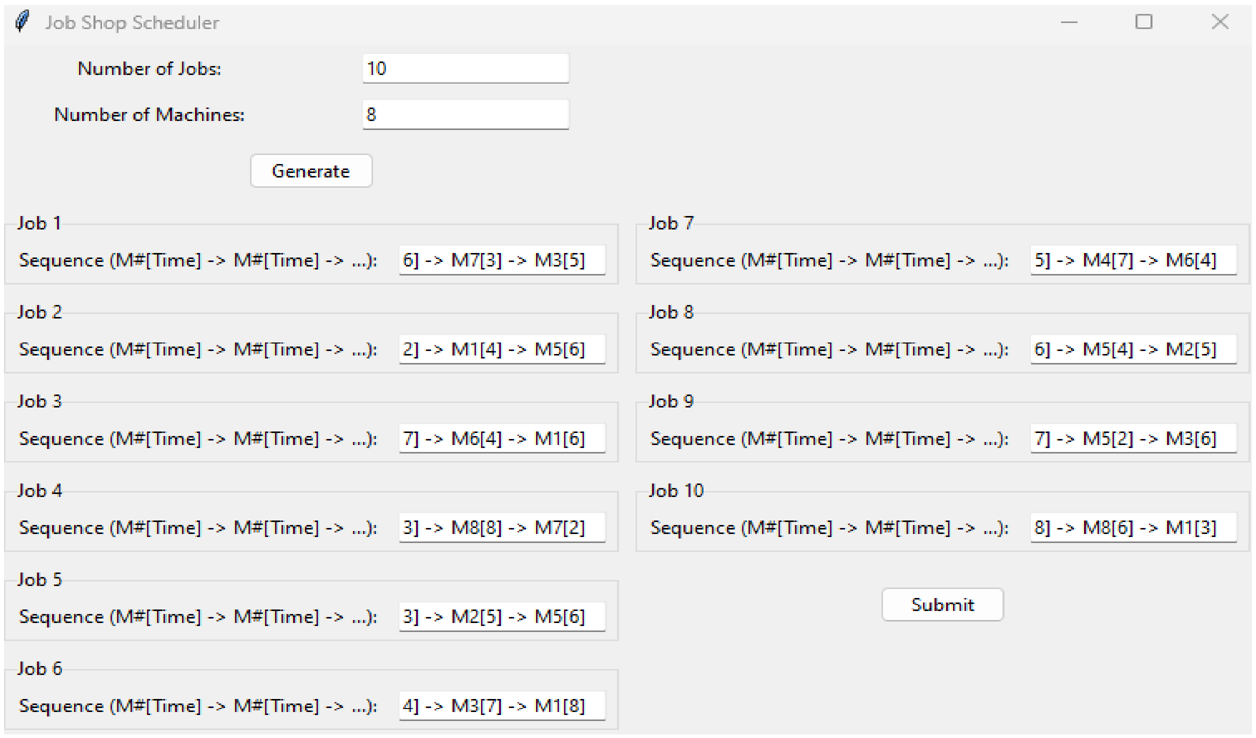
Job 6 M8[5] -> M4[6] -> M7[4] -> M3[7] -> M1[8]

Job 7 M2[6] -> M5[3] -> M8[5] -> M4[7] -> M6[4]

Job 8 M1[8] -> M3[7] -> M7[6] -> M5[4] -> M2[5]

Job  9 M8[4] -> M6[3] -> M1[7] -> M5[2] -> M3[6]

Job  10 M2[5] -> M7[4] -> M4[8] -> M8[6] -> M1[3]



Job Shop Scheduler

Number of Jobs: 10

Number of Machines: 8

Generate

Job 1
Sequence (M#[Time] -> M#[Time] -> ...): 6] -> M7[3] -> M3[5]

Job 2
Sequence (M#[Time] -> M#[Time] -> ...): 2] -> M1[4] -> M5[6]

Job 3
Sequence (M#[Time] -> M#[Time] -> ...): 7] -> M6[4] -> M1[6]

Job 4
Sequence (M#[Time] -> M#[Time] -> ...): 3] -> M8[8] -> M7[2]

Job 5
Sequence (M#[Time] -> M#[Time] -> ...): 3] -> M2[5] -> M5[6]

Job 6
Sequence (M#[Time] -> M#[Time] -> ...): 4] -> M3[7] -> M1[8]

Job 7
Sequence (M#[Time] -> M#[Time] -> ...): 5] -> M4[7] -> M6[4]

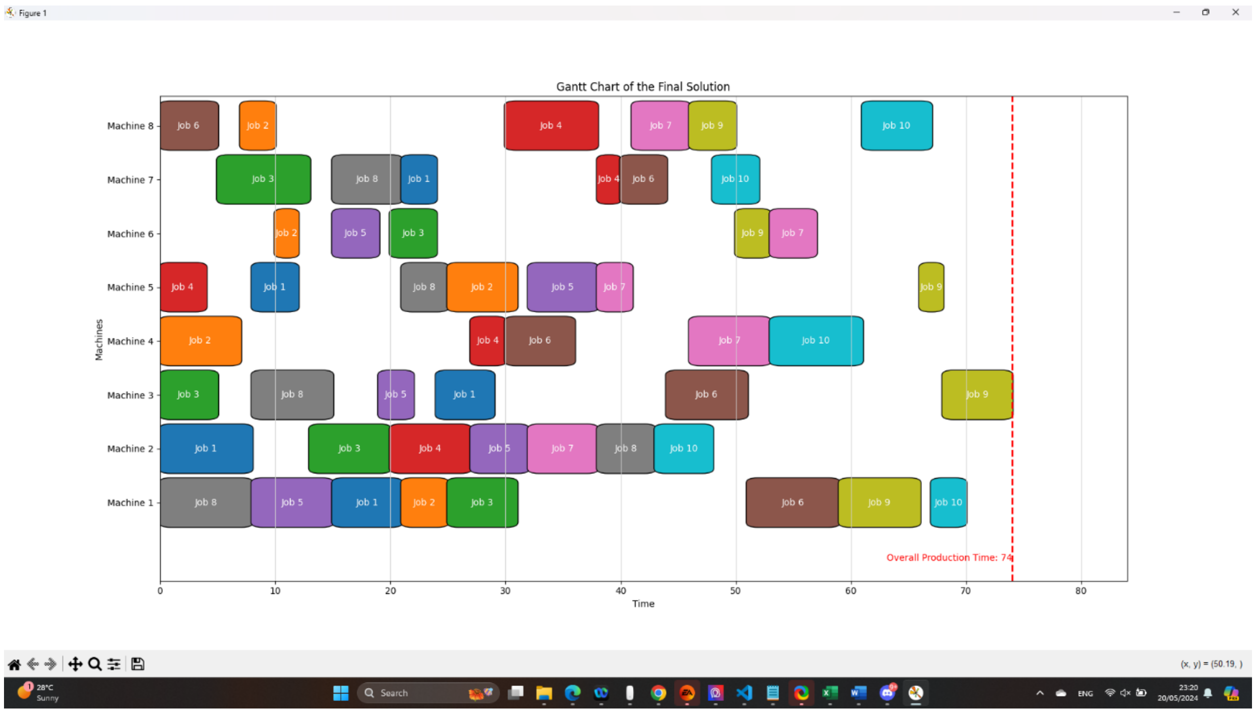
Job 8
Sequence (M#[Time] -> M#[Time] -> ...): 6] -> M5[4] -> M2[5]

Job 9
Sequence (M#[Time] -> M#[Time] -> ...): 7] -> M5[2] -> M3[6]

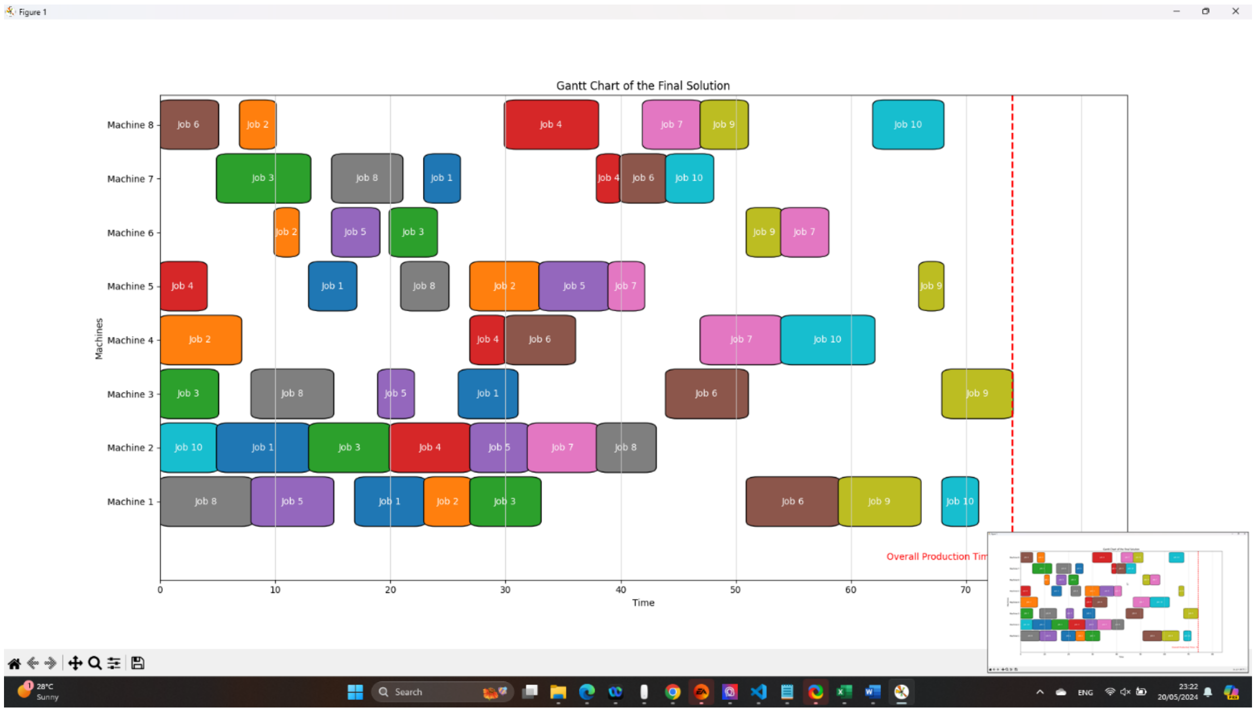
Job 10
Sequence (M#[Time] -> M#[Time] -> ...): 8] -> M8[6] -> M1[3]

Submit

Run 1:



Run2 :



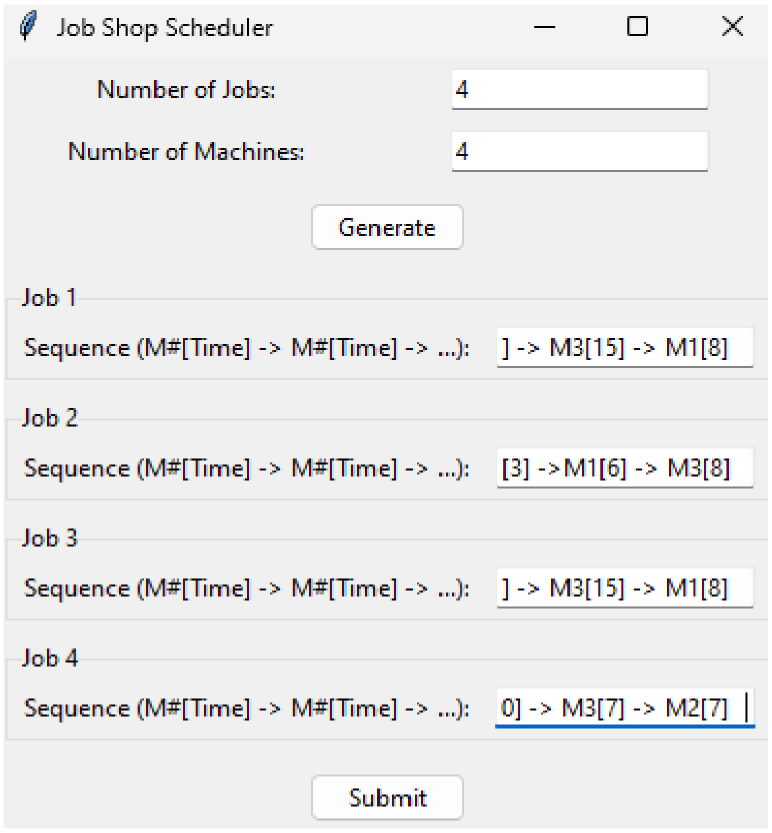
Test 3:

Job 1 M1[9] -> M3[6] -> M2[10] -> M3[5] -> M2[7] -> M3[15] -> M1[8]

Job 2 M1[5] -> M2[10] -> M4[7] -> M3[7] -> M2[15] -> M1[8] -> M4[3] ->M1[6] -> M3[8]

Job 3 M2[3] -> M4[6] -> M1[6] -> M3[8] -> M1[8] -> M3[12] -> M2[5] -> M3[7] -> M2[7] ->
job 4 M3[15] -> M1[8]

Job 5 M1[8] -> M3[12] -> M2[5] -> M1[5] -> M2[10] -> M3[7] -> M2[7]



Job Shop Scheduler

Number of Jobs: 4

Number of Machines: 4

Generate

Job 1

Sequence (M#[Time] -> M#[Time] -> ...): ] -> M3[15] -> M1[8]

Job 2

Sequence (M#[Time] -> M#[Time] -> ...): [3] ->M1[6] -> M3[8]

Job 3

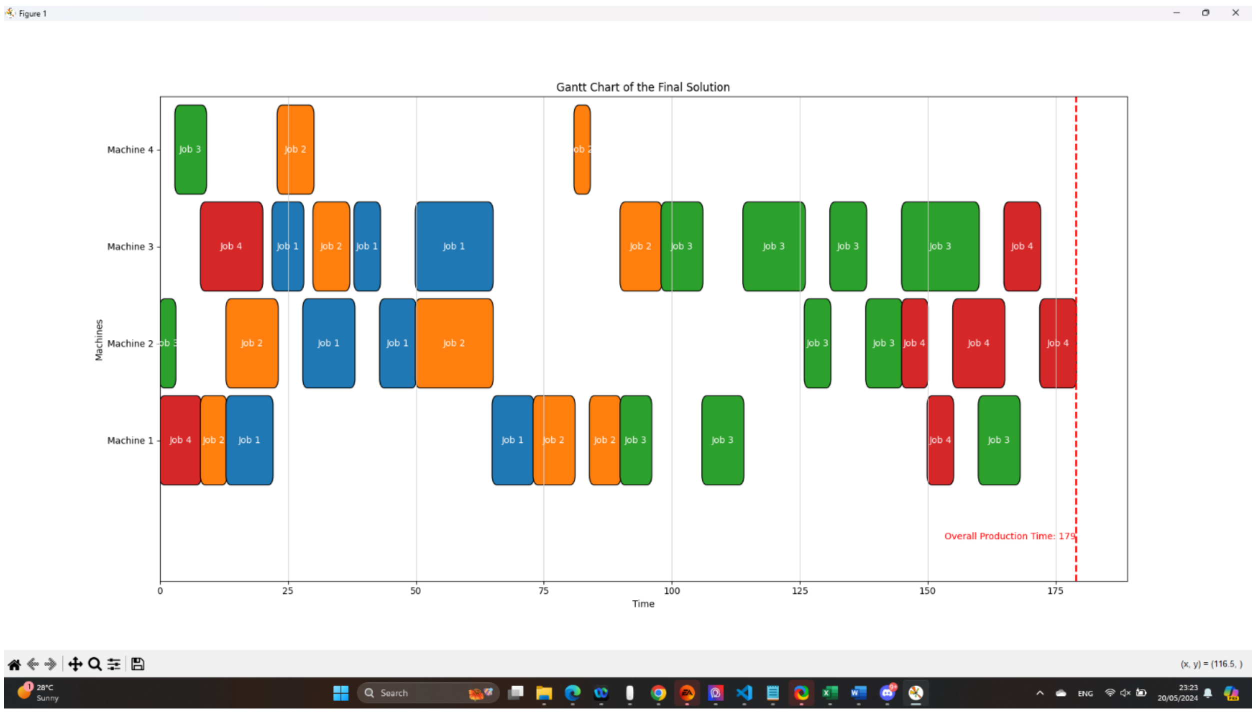
Sequence (M#[Time] -> M#[Time] -> ...): ] -> M3[15] -> M1[8]

Job 4

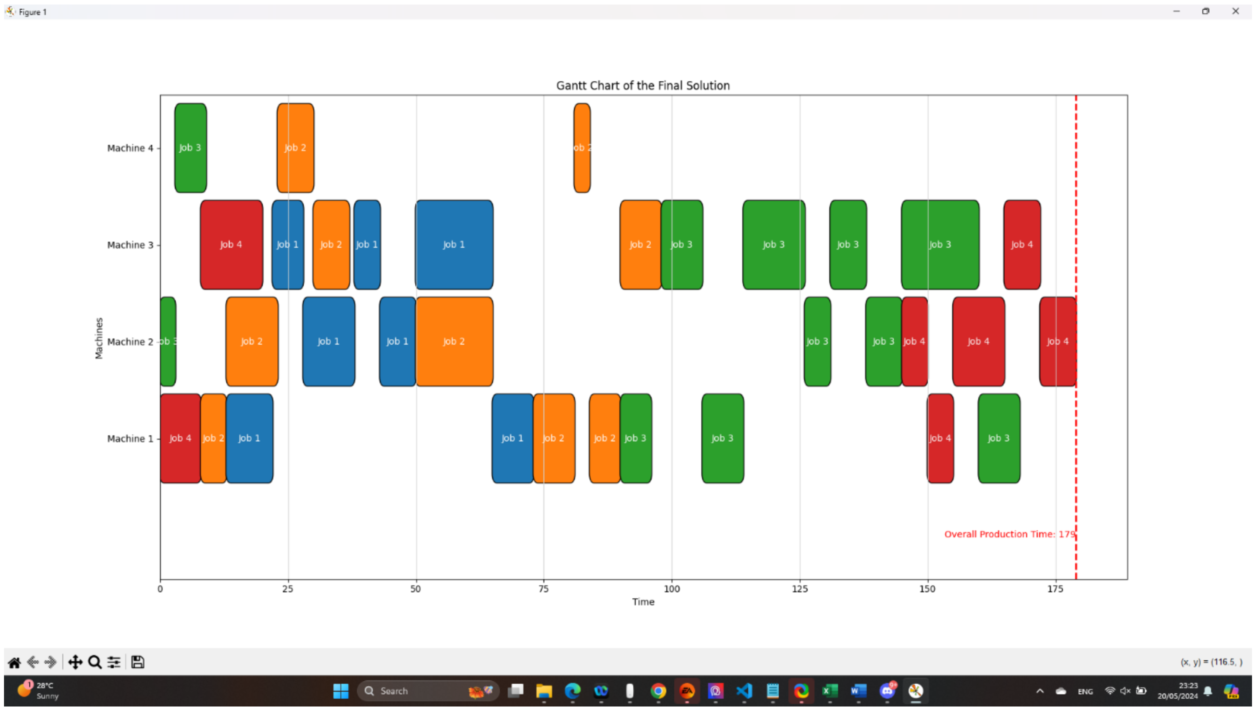
Sequence (M#[Time] -> M#[Time] -> ...): 0] -> M3[7] -> M2[7]

Submit

Run1 :



Run 2 :



Test 4:

M1[8] -> M2[4] -> M3[6] -> M1[3]

M2[7] -> M1[3] -> M2[2] -> M3[4]

M2[5] -> M3[8] -> M2[7] -> M3[4]

M1[4] -> M2[7] -> M3[3] -> M2[8]

M3[6] -> M1[5] -> M2[6] -> M3[2]

Run1:

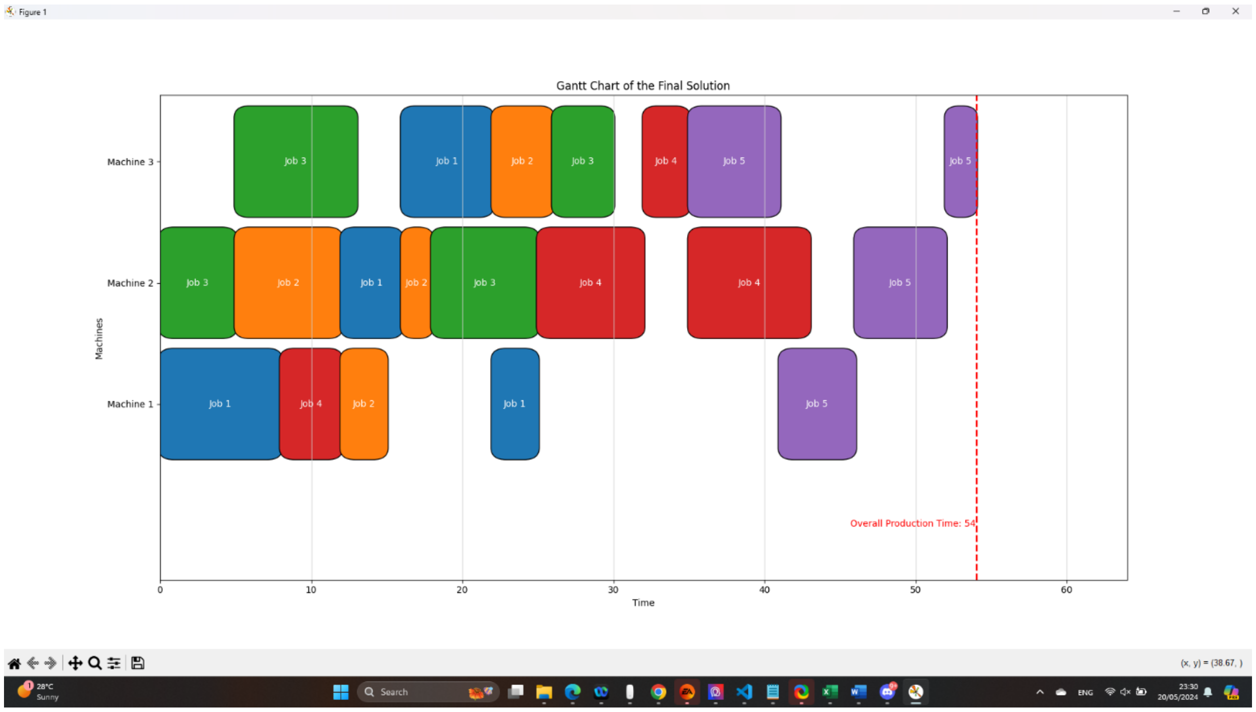


Figure 1
Gantt Chart of the Final Solution
Machine 3
Job 3
Job 1
Job 2
Job 3
Job 4
Job 5
Job 5
Machine 2
Job 3
Job 2
Job 1
Job 2
Job 3
Job 4
Job 4
Job 5
Machine 1
Job 1
Job 4
Job 2
Job 1
Job 5
Machines
Overall Production Time: 54
0
10
20
30
40
50
60
Time
(x, y) = (38.67, )

Run2:

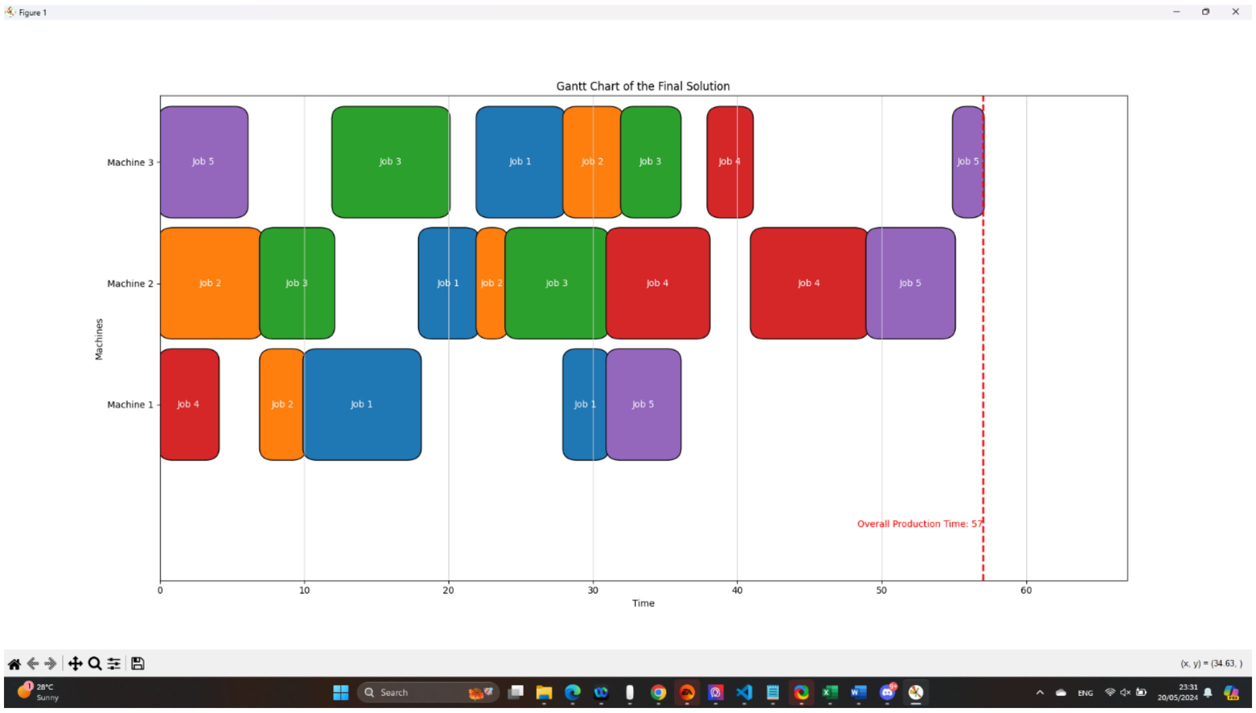


Figure 1
Gantt Chart of the Final Solution
Machine 3
Job 5
Job 3
Job 1
Job 2
Job 3
Job 4
Job 5
Machine 2
Job 2
Job 3
Job 1
Job 2
Job 3
Job 4
Job 4
Job 5
Machine 1
Job 4
Job 2
Job 1
Job 1
Job 5
Machines
Overall Production Time: 57
0
10
20
30
40
50
60
Time
(x, y) = (34.63, )