Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 3340 Artificial Intelligence

Optimizing Job Shop Scheduling in a Manufacturing Plant using Genetic Algorithm

Project # 1 Report

Student: Sarah Hassouneh **ID:** 1210068

Section: 1

Instructor: Dr. Yazan Abu Farha

Date: 18 May 2024

Contents

1 Introduction

In this project we aim to discover genetic algorithm and how we can implement it to create a Job Shop Scheduling System. Genetic Algorithm involve a heuristic process based on the concept of natural selection. This algorithm does not give the optimal solution but can identify good approximations of the solution given space and time limitations. In our project we aim to find a good approximation for the solution of scheduling given the many constraints and limitations of the problem. This report provide a full detail of the problem formulation, parameters, methods used and the general ideas of the code and methods that were used.

2 Problem Formulation

2.1 Chromosome Representation

A chromosome in genetic algorithm represents a valid solution. In our problem we define a job as a set of processes. Each process is uniquely defined by the job number, machine number to be executed on and the sequence with respect to the job. Job-shop scheduling can be viewed as defining the ordering between all processes that must be processed on the same machine. For that case, the data structure used to represent chromosome in our program is a dictionary that maps keys to values. Each key is the machine number and the value is the ordered list of processes to be executed on that machine.

It is important to note that all chromosomes have the same number of processes, and all chromosomes have the same number of processes to be executed on each machine. The difference is in the ordering of the lateral on the machine.

For example , ...

Normally, a chromosome corresponds to a unique solution x in the solution space. We know that the final solution is the final schedule we can build from the chromosome. In this representation each schedule can be drawn from a unique chromosome/ representation. This one-to-one mapping is essential and is present in this representation.

2.2 Constraints

What makes this problem challenging is the multiple constraints that are present, these include :

1. Process P_{jrs} requires the exclusive use of M_r for an uninterrupted duration p_{jr}, for its full processing time.

2. Process P_{jrs} can not start executing until P_{jrs-1} has already finished.

3. If M_r has a the following list of P_{j1rs}, P_{j2rs} , then process P_{j2rs} can not start executing on M_r until P_{j1rs} has already finished.

These constraints of ordering makes the problem extra hard, but also requires extra care when doing crossover or mutation. A chromosome that does not meat the constraints or has an illegal ordering that could lead to a deadlock when building the schedule will be refereed to as a defected chromosome. Defected chromosome are elimanted from the population.

Note : P_{jrs}: is the process in Job (j) on Machine (r) with sequence (s)

2.3 Initial Population

In genetic algorithm, the problem is composed of a group of possible solutions, known as a population. The initial population is a random valid solutions that are improved in each generation.

The initial solution that was generated in our solution are valid chromosomes with random ordering of processes on each machine. The idea is we first read the jobs list that contains the processes. We can then iterate and choose a random number of processes to add to the chromosome from each job. We repeat until all processes have been added to the chromosome and the chromosome becomes full. This approach can create as a group of different random solutions to start with.

2.4 Parameters

Multiple parameters need to be set before beginning with executing the actual algorithm:

Population Size: The population size defines the number of candidate solutions (individuals) present in each generation. A larger population size increases the diversity of the population,but it also leads to increased computation time.

In our program, after multiple testing it was concluded that a good population size ranges from ($numberofprocess$- 2*$numberofprocess$). For a larger population size we get a better solution. If no restrictions are put, we decided to set the population size to 2*$numberofprocess$. However, the program works for a lot less than that and it does give a good enough solution.

Crossover Rate: The crossover rate defines the probability of applying crossover to create new offspring from selected parent individuals. In this program, a crossover rate of 80% or (0.8) means that 80% of the new population will be as a result of a crossover and the rest of the population (20%) are brought from the current population. Those (20%) are taken to be the best (20%) in the current population.

After multiple testings a good crossover rate was decided to be 0.8, so 80% are from crossover and the rest 20% are the elite of the current population. This helps combine beneficial traits from different parents and is crucial for exploration and maintaining diversity in the population.

Mutation Rate: The mutation rate determines the probability of introducing random changes to the genetic information of individual chromosomes. In our program a good mutation rate was found to be (0.3), that means 30% of the newly generated population will undergo a process of mutation.

Termination Criteria: The termination criteria determine when the genetic algorithm stops running. To make this program more consistent the termination criteria was taken based on the number of iterations and improvement. The program is set to run indefinitely and stops if for 30 iterations there was no improvement on the current max.

2.5 Fitness Function

The fitness function assesses the quality of each individual in the population, assessing how well each potential solution solves the problem. In our code we refer to the fitness function as evaluation, because it gives an evaluation for the chromosome. To evaluate any chromosome we start by building a schedule from the representation and then based on the finishing times we can decide which solution is better.

In our program we tested different evaluation functions, the best function that evaluated the problem is **the sum of the finish times of each machine.** This helped in paralleling the processes, increased throughput and decreased overall finish time.

However we know that the convention is the higher the better, but if we build the sum, logically the lower the better, to solve this we defined a total duration. Total duration is the sum of all durations of all processes on all machines, multiplied by the number of machine. This evaluation guides the search process by giving higher scores to better solutions. This evaluation guides the search process by giving higher scores to better solutions.

The final evaluation is : total duration - the sum of the finish times of all machine.

2.6 Crossover

In our program since each chromosome has the list of ordered processes on each machine. Crossover defined as choosing a random machine and swapping the list of processes on that machine with the same machine on another chromosome.

After crossover, the two offsprings that are produced are evaluated and if they are defecated they are given an evaluation of -1 and are not considered as part of the next population.

2.7 Selection

Selection here refers to selecting parents in the crossover process. The selection method that was used in this program is called **Roulette Wheel Selection** represented in Figure 1. In the selection process, individuals with higher fitness values have a higher probability of being chosen for reproduction.

We start by evaluating each individual chromosome, then we calculate total fitness and find the probability for each chromosome. We then create a wheel (array) and we repeat each chromosome by the amount of its probability.

When selecting we select a random index from that wheel (array) and get a value, and this means that better chromosomes are more likely to be selected.

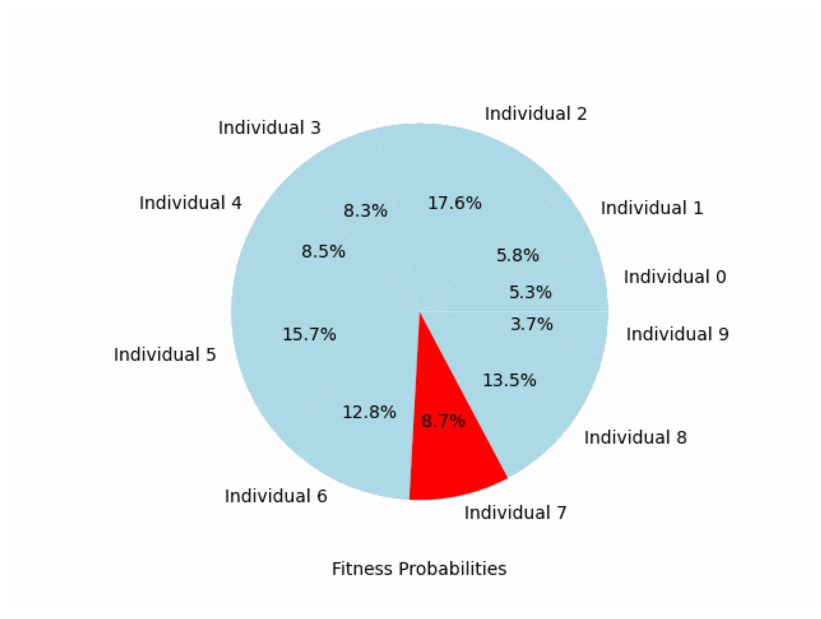

Figure 1: Roulette Wheel Selection

2.8 Mutation

In our program since each chromosome has the list of ordered processes on each machine. Mutation is defined by choosing a random machine and a random index on that machine(except the last) and then switching it with the following process on the same machine. So the order of processes on the same machine is changed.

After mutation, the chromosome is evaluated and if it is defecated it is given an evaluation of -1.

3 Test Cases

3.1 Test Case 1

For this input:

```
J0, M0-2, M4-5, M2-4,
J1, M2-6, M1-1, M3-2, M3-1
J2, M3-1, M1-4, M0-4,
J3, M0-1, M1-1,
```

- Population is set to $2 \times 12 = 24$.
- Termination: no improvement until 30 iterations.
- Crossover rate: 0.8.
- Mutation rate: 0.3.

The result graph solution is shown in Figure 2.

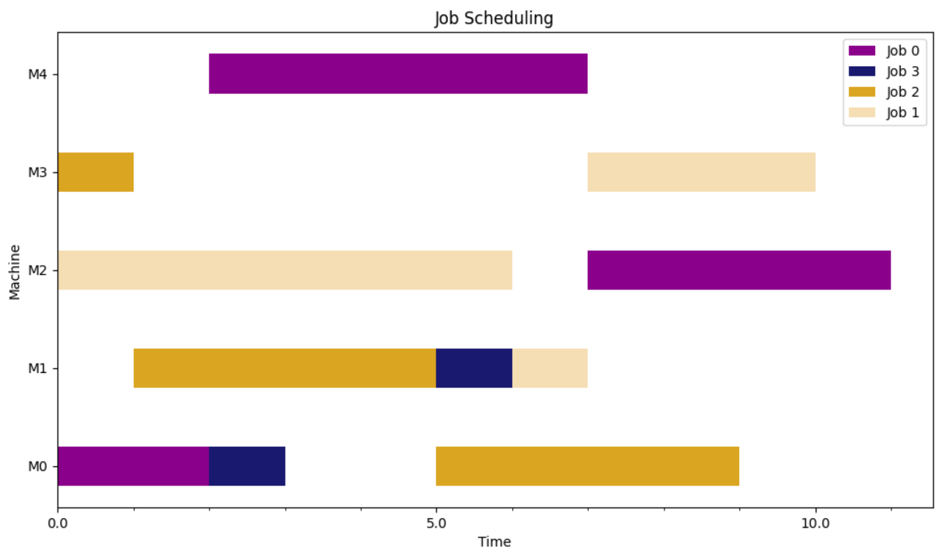

Figure 2: Test case 1 Graph

3.2 Test Case 2

For this input:

```
J0, M0-10, M1-5, M3-12
J1, M1-7, M2-15, M0-8
J2, M0-7, M3-15, M1-8
```

- Population is set to $2 \times 9 = 18$.
- Termination: no improvement until 30 iterations.
- Crossover rate: 0.8.
- Mutation rate: 0.3.

The result graph solution is shown in Figure 3.

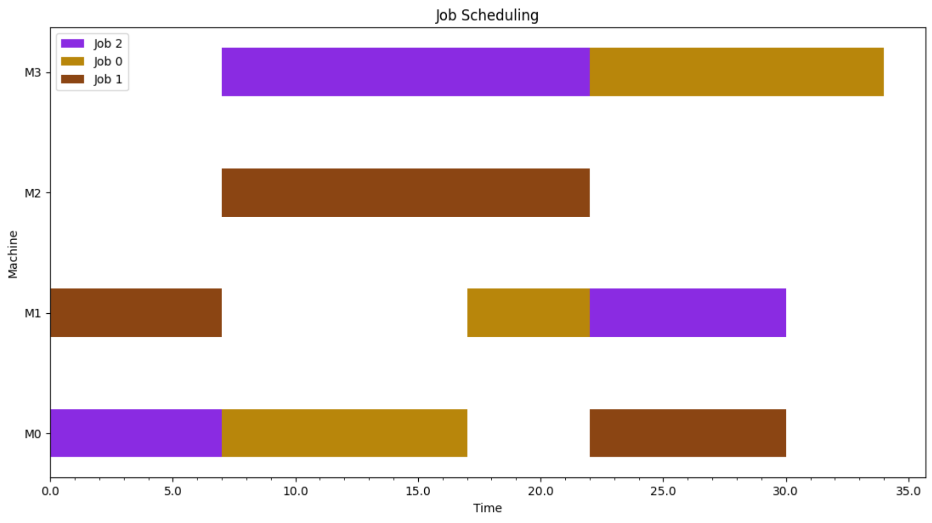

Figure 3: Test case 2 Graph

3.3 Test Case 3

For this input:

```
J0, M1-2, M2-4, M0-1, M3-3
J1, M5-3, M5-3, M6-2
J2, M3-3, M4-2, M7-1, M5-4
J3, M7-2, M4-4, M1-1
J4, M2-1, M6-5, M3-2
J5, M1-3, M7-6, M6-2, M7-3
J6, M3-1, M0-4, M2-5
J7, M3-2, M0-5, M4-3
```

- Population is set to $2 \times 27 = 54$.
- Termination: no improvement until 30 iterations.
- Crossover rate: 0.8.
- Mutation rate: 0.3.

The result graph solution is shown in Figure 4.

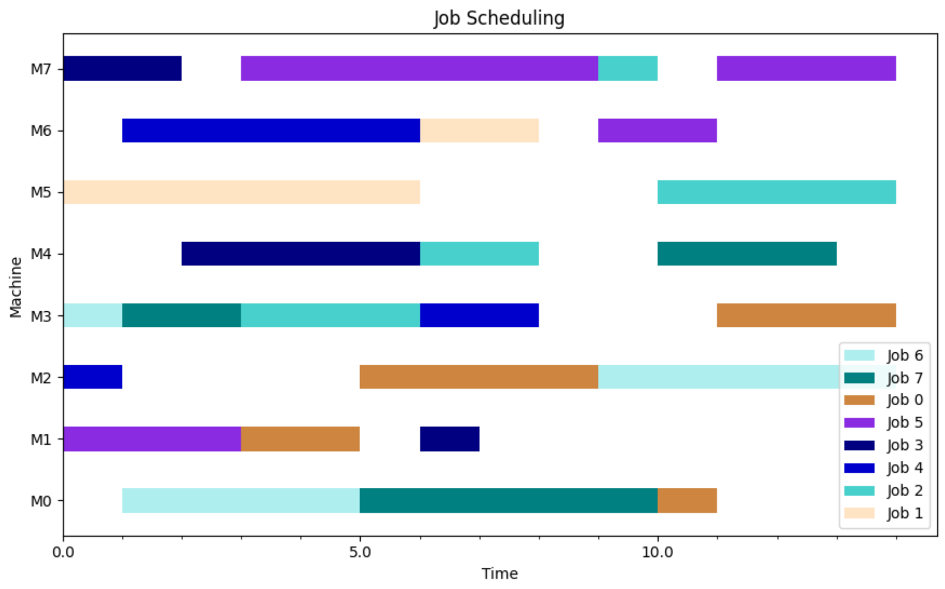

Figure 4: Test case 3 Graph