The Job Scheduling Problem

Bolormaa Mendbayar- x23176725

National College of Ireland

MSc in Data Analytics

Modelling Simulation and Optimization

TABA- Semester 2023/2024

Abstract— This study investigates the optimal production job schedule for a small workshop operating a number of machines, addressing the Job Scheduling Problem. Finding an optimal job schedule is crucial for saving production costs and time, as well as organizing manufacturing processes efficiently. The objective is to identify the most-effective job sequence and schedule, minimizing processing time including wait and idle time, with parameters such as the number of jobs and machines considered. Two algorithms, Greedy and Genetic, are implemented and compared to evaluate their effectiveness in finding optimal solutions. The study highlights the significance of optimizing job schedules for efficient production management and provides insights into the relative performance of the algorithms based on execution and processing times.

Keywords: Job Sequence, Production Schedule, Greedy Algorithm, Genetic Algorithm.

I. INTRODUCTION

In modern manufacturing environments, efficient job scheduling is crucial for optimizing production processes and minimizing costs. The Job-Scheduling Problem (JSP) represents a significant challenge, involving the determination of the optimal sequence of jobs across multiple machines to minimize processing time. This paper examines two contrasting approaches, the Greedy Algorithm and the Genetic Algorithm, in addressing this challenge.

The Greedy Algorithm prioritizes immediate benefits at each step, while the Genetic Algorithm evolves solutions over time, mimicking natural selection. Both methods offer practical solutions to the complexities of job scheduling, with the Greedy Algorithm being straightforward and quick, suitable for real-time decision-making, and the Genetic Algorithm more robust, capable of handling complex optimization problems albeit with increased computational effort.

According to Chen et al. [1], the iterated greedy algorithm has been effective in addressing Distributed Blocking Flowshop Problems (DBFSP) with the objective of minimizing makespan. The BFSP involves scheduling "n" jobs on "m" machines without stopping the jobs mid-process and with no waiting time between machines. The DBFSP extends this to multiple factories (F), each with identical flowshop structures, aiming to minimize the maximum completion time among all factories. The paper concludes that the proposed algorithm is effective in solving the DBFSP with makespan criterion, outperforming other algorithms, especially for large-scale instances.

The paper Zheng et al. [2] deals with the problem of coordinating different types of equipment in container terminals to optimize operations for a low-carbon economy. The challenge is to minimize ship turnaround time by managing quay cranes (QCs), yard trucks (YTs), and yard cranes (YCs) effectively.

To solve this, the researchers first tried a mixed integer linear programming (MILP) model, but it was too complex for large scenarios. So, they developed a genetic algorithm (GA) instead. The GA uses a three-dimensional representation to handle the three types of equipment and introduces new crossover and mutation methods for better solutions.

In their tests with 25 different terminal setups, they found that the GA worked better than MILP in finding good solutions quickly. Even with large-scale problems, the GA could find good solutions in reasonable time. They also compared two versions of the GA and found that one with heuristic mutations was slightly better, especially for bigger problems, even though it took a bit longer to compute.

Through this exploration, aim to compare the performance of these algorithms and provide insights into their efficacy in addressing job scheduling problems.

II. ALGORITHMS

a. Greedy Algorithm

The Greedy algorithm is a heuristic approach used to solve optimization problems by making locally optimal choices at each step. In the context of the Job-Scheduling Problem, the Greedy algorithm aims to minimize the total completion time by assigning jobs to machines based on their processing times. Hontinfinde et al. [3], had similar steps as below shown and in general the Greedy algorithm's outline as follows:

1. Problem Definition:

Define the Job-Scheduling Problem, where a set of jobs needs to be processed on a set of machines with specified processing times.

2. Objective Function:

Formulate the objective function to minimize, such as total completion time or makespan, which represents the total time taken to complete all jobs.

3. Greedy Criterion:

Determine the greedy criterion based on the objective function. For the Job-Scheduling Problem, a common greedy

criterion is to minimize the idle time of machines or prioritize jobs with shorter processing times.

4. Initialization:

Initialize an empty schedule and job sequence. Create data structures to track machine availability and job assignments.

5. Greedy Algorithm Steps:

Assign the selected job to an available machine, minimizing overall completion time. Update the schedule with the assigned job, including start and end times on the respective machine. Define a termination condition based on specific criteria, such as reaching a maximum number of iterations or achieving a satisfactory solution quality.

6. Output:

Return the final schedule and job sequence obtained from the Greedy algorithm.

Firstly, changed the random seed number to obtain the PROC matrix dataset, with N_JOBS=10 and N_MACHINES=4. After implementing the Greedy algorithm, the results are illustrated in Table 1 and Figure 1

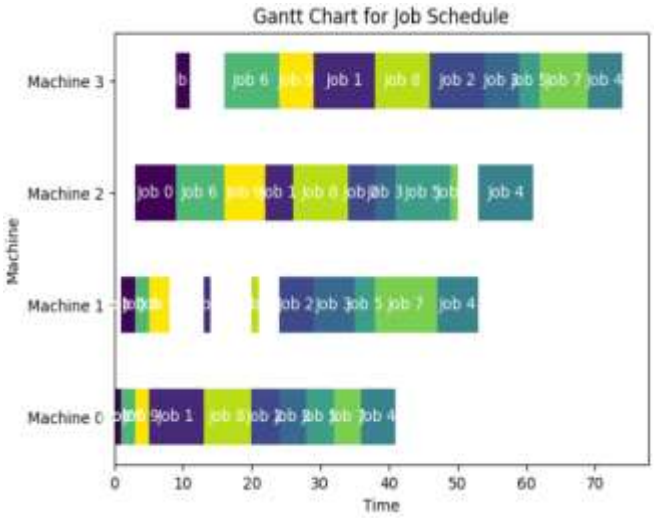

Figure 1. Gantt chart for Greedy Algorithm

below. Note that (0,1) represents the first dimension as the start time, and the second dimension as the end time.

Greedy Algorithm								
Job Sequence: [0, 6, 9, 1, 8, 2, 3, 5, 7, 4]								
Total Processing time: 74								
		Machine 0		Machine 1		Machine 2		Machine 3
		Idle 0		Idle 1		Idle 3		Idle 9
Job 0	Wait 0	(0, 1)	Wait 0	(1, 3)	Wait 0	(3, 9)	Wait 0	(9, 11)
		Idle 0		Idle 0		Idle 0		Idle 5
Job 6	Wait 1	(1, 3)	Wait 0	(3, 5)	Wait 4	(9, 16)	Wait 0	(16, 24)
		Idle 0		Idle 0		Idle 0		Idle 0
Job 9	Wait 3	(3, 5)	Wait 0	(5, 8)	Wait 8	(16, 22)	Wait 2	(24, 29)
		Idle 0		Idle 5		Idle 0		Idle 0
Job 1	Wait 5	(5, 13)	Wait 0	(13, 14)	Wait 8	(22, 26)	Wait 3	(29, 38)
		Idle 0		Idle 6		Idle 0		Idle 0
Job 8	Wait 13	(13, 20)	Wait 0	(20, 21)	Wait 5	(26, 34)	Wait 4	(38, 46)
		Idle 0		Idle 3		Idle 0		Idle 0
Job 2	Wait 20	(20, 24)	Wait 0	(24, 29)	Wait 5	(34, 38)	Wait 8	(46, 54)
		Idle 0		Idle 0		Idle 0		Idle 0
Job 3	Wait 24	(24, 28)	Wait 1	(29, 35)	Wait 3	(38, 41)	Wait 13	(54, 59)
		Idle 0		Idle 0		Idle 0		Idle 0
Job 5	Wait 28	(28, 32)	Wait 3	(35, 38)	Wait 3	(41, 49)	Wait 10	(59, 62)
		Idle 0		Idle 0		Idle 0		Idle 0
Job 7	Wait 32	(32, 36)	Wait 2	(38, 47)	Wait 2	(49, 50)	Wait 12	(62, 69)
		Idle 0		Idle 0		Idle 3		Idle 0
Job 4	Wait 36	(36, 41)	Wait 6	(47, 53)	Wait 0	(53, 61)	Wait 8	(69, 74)

Table 1. Greedy algorithm job sequence and schedule

b. Genetic Algorithm

Genetic Algorithms (GAs) are a class of optimization algorithms inspired by the process of natural selection and evolution. They are particularly well-suited for solving complex optimization problems where traditional methods may struggle to find satisfactory solutions.

At the core of Genetic Algorithms is the idea of mimicking biological evolution to find optimal or near-optimal solutions to a given problem. The process involves maintaining a population of candidate solutions, which are often represented as chromosomes.

Each chromosome represents a potential solution to the problem, encoded in a way that can be manipulated and evaluated by the algorithm. For example, in the context of the Job-Scheduling Problem, a chromosome could represent a sequence of jobs to be executed on different machines.

The main components of Genetic Algorithms as mentioned in Hontinfinde et al. [3], are as below:

1. Initialization:

A population of chromosomes is randomly generated or initialized using heuristic methods.

2. Selection:

Individuals (chromosomes) are selected from the population for reproduction based on their fitness, which is determined by how well they perform in solving the problem.

3. Crossover:

Selected individuals undergo crossover or recombination, where parts of their genetic material are exchanged to create new offspring chromosomes.

4. Mutation:

Random changes are introduced to the offspring chromosomes to maintain diversity in the population and prevent premature convergence to suboptimal solutions.

5. Evaluation:

The fitness of the new individuals (offspring) is evaluated, typically using an objective function that quantifies how well they solve the problem.

6. Replacement:

Based on their fitness, some individuals are selected to survive and form the next generation, while others may be replaced by newly generated individuals.

7. Termination:

The algorithm continues iterating through generations until a termination condition is met, such as reaching a maximum number of generations or finding a satisfactory solution.

The Genetic algorithm was implemented following the Greedy algorithm, with parameters set as follows: pop_size = 50, gen_max = 200, mutation_rate = 0.1, and crossover_rate = 0.8. Due to the variability of results produced by the Genetic Algorithm, a random seed number was used to ensure stability. The results obtained from the Genetic Algorithm are presented below.

Genetic Algorithm								
Job sequence: [6 8 3 9 1 2 5 4 0 7]								
Processing time: 71.0								
		Machine 0		Machine 1		Machine 2		Machine 3
		Idle 0		Idle 2		Idle 4		Idle 11
Job 6	Wait 0	(0, 2)	Wait 0	(2,4)	Wait 0	(4, 11)	Wait 0	(11, 19)
		Idle 0		Idle 5		Idle 0		Idle 0
Job 8	Wait 2	(2, 9)	Wait 0	(9, 10)	Wait 1	(11, 19)	Wait 0	(19, 27)
		Idle 0		Idle 3		Idle 0		Idle 0
Job 3	Wait 9	(9, 13)	Wait 0	(13, 19)	Wait 0	(19, 22)	Wait 5	(27, 32)
		Idle 0		Idle 0		Idle 0		Idle 0
Job 9	Wait 13	(13, 15)	Wait 4	(19, 22)	Wait 0	(22, 28)	Wait 4	(32, 37)
		Idle 0		Idle 1		Idle 0		Idle 0
Job 1	Wait 15	(15, 23)	Wait 0	(23, 24)	Wait 4	(28, 32)	Wait 5	(37, 46)
		Idle 0		Idle 3		Idle 0		Idle 0
Job 2	Wait 23	(23, 27)	Wait 0	(27, 32)	Wait 0	(32, 36)	Wait 10	(46, 54)
		Idle 0		Idle 0		Idle 0		Idle 0
Job 5	Wait 27	(27, 31)	Wait 1	(32, 35)	Wait 1	(36, 44)	Wait 10	(54, 57)
		Idle 0		Idle 1		Idle 0		Idle 0

Job 4	Wait 31	(31, 36)	Wait 0	(36, 42)	Wait 2	(44, 52)	Wait 5	(57, 62)
		Idle 0		Idle 0		Idle 0		Idle 0
Job 0	Wait 36	(36, 37)	Wait 5	(42, 44)	Wait 8	(52, 58)	Wait 4	(62, 64)
		Idle 0		Idle 0		Idle 0		Idle 0
Job 7	Wait 37	(37, 41)	Wait 3	(44, 53)	Wait 5	(58, 59)	Wait 5	(64, 71)

Table 2. Genetic algorithm job sequence and schedule

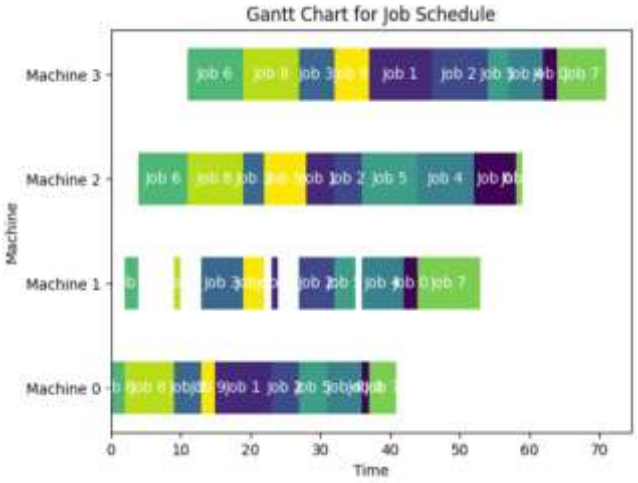

Figure 2. Gantt chart for Genetic Algorithm

To compare the Greedy and Genetic Algorithms, the job number was increased to observe differences in processing and execution times. Due to the stochastic nature of the algorithms, variability in execution times across runs was observed. To ensure reliable results, 10 multiple runs were conducted, and outcomes were averaged, but still results are not consistent, so took approximate execution time. Below illustrated in Table 3 that the Genetic Algorithm exhibits faster processing times compared to the Greedy Algorithm. However, when considering execution times, the Greedy Algorithm outperforms the Genetic Algorithm, demonstrating its efficiency in completing the scheduling task within shorter durations. This suggests that while the Genetic Algorithm may excel in finding solutions quickly, the Greedy Algorithm is more efficient in executing those solutions.

	Processing time		Average execution time	
Job number	Greedy	Genetic	Greedy	Genetic
10	74	71	0.0	0.786
20	137	129	0.0	1.333
50	304	294	0.0	3.939
100	580	573	0.0	7.332

Table 3. Comparison of Greedy and Genetic algorithms

III. EVALUATION

The comparison between the Greedy and Genetic Algorithms for the Job Scheduling Problem provided valuable insights. This findings align with previous studies [1], [2], [3], showcasing the Genetic Algorithm's effectiveness in minimizing processing time. While [1] and [2] emphasize its efficiency in solving complex scheduling problems, these results further demonstrate its superiority in this aspect. However, the Greedy Algorithm outperforms the Genetic Algorithm in execution time, validating findings by [3].

Future research could focus on refining Genetic Algorithm parameters to improve execution time without compromising its advantage in processing time. This could involve refining the selection and crossover mechanisms to better explore the solution space and converge towards optimal or near-optimal solutions more efficiently. Additionally, exploring hybrid algorithms that combine the strengths of both Greedy and Genetic approaches could lead to enhanced solutions in job scheduling. Furthermore, implementing different algorithms such as Stochastic and Ant Colony Optimizations could provide further insights and comparisons. Moreover, varying the number of machines, in addition to the job number, could offer a more comprehensive understanding of how different factors impact scheduling performance. Consistently achieving execution times for the Genetic algorithm for each number of job could be an area of focus for future work.

IV. REFERENCES

[1] S. Chen, Q.-K. Pan, X. Hu, and M. F. Tasgetiren, "An Iterated Greedy Algorithm for Distributed Blocking Flowshop Problems with Makespan Minimization," in Proceedings of the 39th Chinese Control Conference, Shenyang, China, July 27-29, 2020, pp. 1536-1540. Available: [Online]. Available: IEEE Xplore. [Accessed: May 13, 2024]. DOI: 10.1109/ChiCC51111.2020.9262049.

[2] Y. Zheng, M. Xu, Z. Wang, and Y. Xiao, "A Genetic Algorithm for Integrated Scheduling of Container Handing Systems at Container Terminals from a Low-Carbon Operations Perspective," Sustainability, vol. 15, no. 7, doi: 10.3390/su15076035

[3] R. D. Hontinfinde, A. Kamoyedji, M. T. Vitouley, and R. S. Honfo, "A comparative study of two different optimization methods applied to the static job scheduling problem in grid computing," in 2023 9th International Conference on Computer and Communications (ICCC), 8-11 Dec. 2023, pp. 2565-2571, doi: 10.1109/ICCC59590.2023.10507679.