Comparative Analysis of Greedy and Genetic Algorithms for Optimal Job Scheduling in a Workshop Environment

Isaac Umukoro
Postgraduate Diploma in Science In Data Analytics
(PGDDA)
National College of Ireland
(NCIRL)
Dublin, Ireland
x23215640@student.ncirl.ie

***Abstract*—This study discusses the implementation of two job scheduling algorithms: the Greedy Algorithm and the Genetic Algorithm. The goal is to find an optimal production schedule or a schedule as near enough as possible to the optimal schedule, in this case, the integer programming solution is used as the benchmark The study employs a custom "generateData()" function, based on specific seed values, to mimic various task and machine settings, assessing the algorithm's efficiency and accuracy. The results show that while the Greedy Algorithm gives a reliable rapid approximation of the scheduling problem, the Genetic Algorithm excels at getting closer to ideal solutions, especially as the complexity of work scheduling rises. This comprehensive analysis not only illustrates each algorithm's capabilities and limits. it also emphasizes the Genetic Algorithm's supremacy in addressing larger and more complex scheduling scenarios.**

Keywords—Job scheduling, Greedy algorithm, Genetic algorithm, Integer programming.

I. INTRODUCTION

The Job Scheduling Problem presents a significant challenge in research operations and computer science [1]. It is the process of allocating jobs to resources across time. The goal is to achieve one or more objectives, such as reducing resource use. Job scheduling is critical in various industrial and service sectors, including manufacturing, cloud computing, and project management, because scheduling effectiveness directly impacts productivity and operating costs.

This study delves into two algorithms: The Greedy Algorithm and the Genetic Algorithm to solve the Job Scheduling Problem in a simple workshop.

The Greedy Algorithm [1] was chosen because of its simple approach and computational efficiency, making it excellent for situations requiring quick and straightforward answers. The Greedy Algorithm iteratively creates solutions, selecting locally optimal options at each stage in the hopes of achieving a global optimum. Greedy algorithms have been extensively studied and applied to a variety of scheduling problems. For example, Jackson's rule is a well-known greedy method for workshop scheduling that selects the next job based on the earliest due date to minimize lateness.

Genetic algorithms (GAs) [1] are selected for their robustness and ability to effectively navigate large and complex search spaces, based on natural selection principles. They utilize techniques such as selection, crossover, and mutation to guide a population of candidate solutions to optimality. John Holland created genetic algorithms in the 1970s, and they have subsequently been successfully used for a variety of optimization problems, including job scheduling. Numerous studies have shown that GAs are adaptable and successful in addressing job scheduling difficulties.

The Greedy approach for job scheduling is based on classical scheduling theory, involving foundational studies such as Johnson's optimal solution for two-machine flow shop scheduling. Genetic algorithms [1], on the other hand, have been widely employed to tackle a range of optimization problems, including job scheduling, due to their versatility and ability to discover near-optimal solutions to NP-hard problems. This project will assess and compare the performance of these two algorithms in solving the Job Scheduling Problem for a workshop with several machines. The comparison will be based on how accurate and efficient each algorithm's solutions are.

II. ALGORITHMS AND PERFORMANCE ANALYSIS

A. The greedy Algorithm

The Greedy Algorithm used in this analysis for solving the Job Scheduling problem is derived from the general form of the Greedy Algorithm as introduced in the example in class contained in the TSP Greed 2 Jupyter Notebook [1]. The application of the Greedy Algorithm to the Job Scheduling problem involves selecting the next job to schedule based on the minimum start time for any given job on any machine. This implementation ensures that the job schedule is designed in such a way as to minimize idle time and maximize machine utilization. In Travelling Salesman problem example 2 in class, once a point is added to the path, it is removed from the list of remaining points, whereas, in job scheduling, once a job is scheduled, it is removed from the list of remaining jobs, and the machine availability is updated. The greedy approach is discussed in "Introduction to Algorithms" by Cormen, Leiserson, Rivest, and Stein [1], which describes how local decisions are made at each step to find an approximate solution.

Greedy Algorithm Application; The application of the Greedy algorithm to solve the job scheduling problem involves the following steps outlined below

- Initialization: This involves setting up the Greedy Algorithm data to track start times, stop times, idle times, and wait times for each job on each machine. it initialize the time available for each machine to zero, the last job on each machine to -1, indicating that no job has yet to be processed, and finally initialize an empty job sequence list. They include 'start', 'stop', 'idle', and 'wait'.
- Selection of Job: This involves selecting the next job iteratively that can be started the earliest on any machine. The determining factor for this is the minimum additional time required to start each job on each machine. It then adds the selected job to the job sequence list and removes the selected job from the list of remaining jobs.
- Update of Schedule: This involves updating the start and stop times for each selected job on each machine. It calculates the idle time for each machine as the difference between the start time of the current job and the stop time of the last job processed on the machine, then updates the available time for each machine to the stop time of the current job. It also calculates the wait time for the current job and updates the last completed job on each machine to the current job.
- Repeat: It involves the continuation of the process until all jobs are scheduled
- Calculation of Total Completion Time: This is the total time required to complete all jobs across all machines.

Derivation of the Greedy Algorithm as Introduced in the Travelling Salesman Problem (TSP): The application of the Greedy Algorithm in this context is because it selects the next job at the earliest possible time on any machine which is similar to how the Greedy Algorithm employs the nearest neighbor approach to select the closest city in the Traveling Salesman Problem (TSP). This implementation meets the specific constraint and requirement of the Job scheduling problem while applying the Greedy method.

Result and Performance Measurement: The relative accuracy calculation is the performance of the Greedy Algorithm is compared to the optimal solution obtained using the integer programmer. For N_JOBS = 7, the optimal solution of the Integer programming using Pulp is 55. The Greedy Algorithm produces an objective value of 60.

$$\text{Relative Accuracy} = (55 - 60/55) \times 100 = -9.09\% \quad \text{(I)}$$

Therefore, the relative accuracy of the Greedy Algorithm is -909%.

Runtime result: The Greedy Algorithm completed its processing immediately with a run time of 0.0000 seconds which is the same as the run time for the Pulp integer programming as shown in Fig. 1 as seen in cell 28, Jupyter Notebook file. This is expected because the Greedy Algorithm is constructed to make quick results with complex computation.

Algorithm	Objective Value	Runtime
Greedy Algorithm	60	0.00 secs
Integer Programming	55	0.00 secs

Fig. 1. Runtime Table

B. The Genetic Algorithm

A genetic algorithm is a type of evolutionary algorithm that simulates the process of natural selection. It is widely used for optimization problems. This has been discussed in various books, including Dan Simon's "Evolutionary Optimization Algorithms".

The Genetic Algorithm utilized in this analysis is derived from the literature titled "Evolutionary Optimization Algorithms" by Dan Simon and the DEAP Library Documentation.
In the Job Scheduling Problem, the implementation of the Genetic Algorithm evolves a population of job sequences across several generations, to obtain a near-optimal schedule. This method analyzes the solution space and improves solution quality through the use of selection, crossover, and mutation operators.

Application of the Genetic Algorithm: Below is the process involved in the application of the Genetic Algorithm as it applies to the job scheduling problem.

- Initialization: This involves the creation of an Initial population of the job sequence, and also creating a fitness function for the evaluation of the objective value (makespan) for each solution.
- Evaluation: This involves the calculation of the fitness of each individual in the population.
- Selection: This involves the selection of individuals based on their fitness to reproduce. Better solutions are more likely to be selected and the common methods used include the tournament selection and the roulette wheel selection.
- Crossover: Combine pairs of selected individuals to produce offspring. This is done using crossover operations like one-point crossover or ordered crossover.
- This involves the combination of pairs of selected individuals to produce offspring. This is carried out by using crossover operations like the one-point crossover or ordered crossover.
- Mutation: this is done by applying random changes to offspring to maintain genetic diversity. Mutation

operations might include shuffling parts of the job sequence or swapping positions of the job.

- Replacement: This is done by forming a new population by selecting the best individuals from the current population, and offspring
- Repeat: The process is repeated until the specified criteria are met.
- Return Best Solution: The best individual selected from the final population is used to represent the optimal or near-optimal Job sequence.

Derivation of Genetic Algorithm from its Original form:

In the context of job scheduling, the GA is designed to function with permutations of jobs. The fitness function is intended to reduce the makespan, and suitable genetic operators (such as ordered crossover and shuffle mutation) are utilized to keep valid job sequences [1].

This implementation of the Genetic Algorithm used to solve the Job scheduling problem in this analysis, is based on the pseudo-code and explanations provided in "Evolutionary Optimization Algorithms" by Dan Simon and DEAP documentation.

Result and Performance Measurement: The performance of the Genetic Algorithm is compared to the optimal solution obtained using the integer programmer. For N_JOBS = 7, the optimal solution of the Integer programming using Pulp is 55. The Genetic Algorithm produces an objective value of 55.

$$\text{Relative Accuracy} = (55 - 55/55) \times 100 = 0\% \qquad \text{(II)}$$

Runtime result: The genetic algorithm completed its operation with a runtime of 0.1285 seconds as seen in the table in Fig. 2 available in cell 29, number 3, Jupyter Notebook file. This is beyond the runtime of Integer programming and Greedy algorithms. The high runtime is due to the iterative nature of the genetic program, involving multiple generations of selection, crossover, and mutation.

Algorithm	Objective Value	Runtime
Genetic Algorithm	55	0.00 secs
Integer Programming	55	0.1285secs

Fig. 2. Runtime

III. EVALUATION

1. The Greedy Algorithm: The performance of the Greedy Algorithm was evaluated for different numbers of jobs (N_JOBS = 5,6,7,8,9,10). The table below shows the Job sequence and objective value for each of the N_JOBS carried out by the Greedy Algorithm.

N_JOBS	Job Sequence	Objective Value	Runtime in Seconds
5	3,2,1,0,4	49	0.0000
6	3,2,5,1,0,4	54	0.0000
7	3,2,5,6,1,0,4	60	0.0000
8	3,7,2,5,6,1,0,4	61	0.0000
9	3,7,8,2,5,6,1,0,4	66	0.0000
10	3,7,8,9,2,5,6,1,0,4	74	0.0000

Fig. 3 Performance table

The results show that the Greedy Algorithm is fast by achieving a runtime of 0.0000 across the different numbers of jobs. Also, when comparing the objective value of the Greedy Algorithm with the optimal solution in Integer programming, in this case, N_JOBS = 5, The objective value of the Greedy Algorithm was 60 which is slightly higher than that of the Integer programming which is 55. This indicates that, while the Greedy Algorithm is fast, it does sacrifice the optimal result for speed.

2. The Genetic Algorithm: The Performance of the Genetic Algorithm was also evaluated for different numbers of Jobs (N_JOBS = 5,6,7,8,9,10). The table below in Fig. 4 shows the job sequence and the objective value for each of the N_JOBS carried out by the Genetic Algorithm as found in the Job Scheduling Solution cell 29, number 2, Jupyter Notebook file.

N_JOBS	Job Sequence	Objective Value	Runtime in Seconds
5	2,3,1,0,4	49	0.1196
6	2,1,3,0,4,5	52	0.1188
7	2,1,3,0,4,5,6	55	0.1285
8	7,1,2,0,4,3,6,5	57	0.01316
9	7,1,8,0,2,6,5,4,3	63	0.1468
10	9,8,1,2,0,6,4,5,7,3	69	0.1478

Fig. 4 Performance Table for Genetic Algorithm

The above results show that the objective function of the Genetic Algorithm is the same as that of the Integer programming. In this case of N_JOBS = 5, The objective value of the Genetic Algorithm was 55, which is the same as the integer programming but with a higher runtime of 0.1285 seconds. This indicates the efficiency of the Genetic Algorithm and its ability to find the near-optimal value but at the expense of high computational time.

3. Potential Improvement Future research may examine hybrid strategies that combine the genetic algorithm's accuracy and the greedy algorithm's speed. Further enhancing the Genetic Algorithm's performance could involve adjusting its parameters (such as population size and mutation rate). Furthermore, by reducing its runtime through parallelization, the Genetic Algorithm may become more useful for larger problem situations.

References

[1] J. R. Jackson, Scheduling a production line to minimize maximum tardiness, Califonia: Office of Technical Services, 1955.

[2] J. H. Holland, Adaptation in Natural and Artificial Systems, The MIT Press, 1992.

[3] R. C. Mitsuo Gen, Genetic Algorithms and Engineering Optimization, John Wiley & Sons, 2007.

[4] J. N. Dimitris Bertsimas, Introduction to Linear Optimization, Athena Scientific, 1997.

[5] C. E. L. R. L. R. C. S. Thomas H. Cormen, Introduction to Algorithms, Cambridge, Massachusetts: MIT Press, 2002.

[6] M. L. Pinedo, Scheduling: Theory, Algorithms, and Systems, Springer, 2022.