Job Scheduling Problem Optimization Using Greedy and Stochastic Optimization Algorithms

Etinosa Eghaghe
MSc in Data Analytics
National Collegde of Ireland
x23138548@student.ncirl.ie

***Abstract*—The Job-Scheduling Problem (JSP) is a classic optimization problem with numerous real-world applications across various industries. This report investigates the problem of job scheduling in a workshop setting, where jobs need to pass through a sequence of machines. Two algorithms, namely Greedy Algorithm and a Stochastic Optimization algorithm employing Simulated Annealing, are implemented, and compared against an Integer Programming (IP) solution using Pulp. The objective is to find an optimal or near-optimal production schedule given the number of machines, jobs, and their processing times. Performance measurements including relative accuracy and runtime are provided for each algorithm concerning the IP solution. Results show that while the IP solution guarantees optimality, the Greedy Algorithm and Simulated Annealing offer competitive solutions with less computational complexity.**

Keywords*—*Job Scheduling Problem, Greedy Algorithm, Stochastic Optimization algorithm, Simulation Annealing, Optimization, Workshop.

introduction

The optimization of job scheduling in manufacturing environments is critical for enhancing productivity and resource usage. The job scheduling problem (JSP) poses a complex combinatorial optimization challenge, necessitating the exploration of heuristic and metaheuristic algorithms for efficient solution generation[1]. In a workshop scenario, jobs must be processed sequentially through a series of machines, with each machine having specific processing times for each job. The aim of this study is devising efficient scheduling strategies to minimize job completion time or makespan, and maximize resource utilization.

This report explores the performance of two heuristic algorithms, Greedy Algorithm and Simulated Annealing, in comparison to an Integer Programming solution. These algorithms are implemented and evaluated based on their accuracy and runtime efficiency. Aim to contribute to the ongoing discourse surrounding JSP optimization and its practical implications in real-world scenarios.

Literature Review

The Job Scheduling Problem (JSP) is a significant area of study in both academia and industry, aiming to optimize resource allocation and production efficiency [2]. A range of techniques has been explored to tackle its complexities.

Integer Programming (IP) offers exact solutions by formulating JSP as a mathematical model, but its computational complexity can be prohibitive for large-scale problems [3]. Heuristic methods, like the Greedy Algorithm, provide simpler and faster solutions by prioritizing local optimal decisions [4]. While effective in certain scenarios, they may struggle with more complex instances.

Metaheuristic algorithms, such as Simulated Annealing, offer a flexible approach by exploring solution spaces probabilistically. They can handle dynamic environments and avoid local optima, making them suitable for complex JSP instances [5].

Studies have compared these approaches, showing trade-offs between solution quality and computational efficiency. Exact methods ensure optimality but may be impractical for large problems. Heuristic and metaheuristic methods offer scalable solutions, making them preferable for dynamic environments and larger instances [2].

In summary, JSP optimization literature spans various approaches, from exact methods to heuristic and metaheuristic techniques. Each has its advantages and challenges, highlighting the importance of selecting the right approach based on problem characteristics and computational resources. This study contributes by evaluating the Greedy Algorithm and Simulated Annealing against Integer Programming, shedding light on their effectiveness for real-world scheduling challenges.

methodology

The methodology involves exploring two different algorithms (Greedy and simulated Annealing algorithm) to find optimal production schedule in a workshop scenario. Using parameter N-MACHINE = 4 and N-JOB = 5, N-JOB = 6, N-JOB = 7.
N-MACHINE = 4: It represents the number of machines available for processing the jobs.
N-JOB = 5, N-JOB = 6, N-JOB = 7: They represent the number of jobs.

Greedy Algorithm:

The Greedy Algorithm prioritizes the assignment of jobs to machines based on minimizing the sum of processing times [4]. It iteratively assigns each job to the machine with the least cumulative processing time among all jobs assigned to that machine.

The implemented algorithm calculates start, wait, and stop times for each job on each machine to determine the schedule. Finally, it computes the makespan, representing the total time taken to complete all jobs.

Algorithm outline:

1. Initialize an empty schedule: At the beginning of the algorithm, create an empty schedule where task-resource assignments will be stored. This schedule serves as the canvas upon which tasks will be allocated to resources.
2. Iterate over tasks in a predefined order: Next, the algorithm loops through each task in a predefined order. This order could be based on various criteria, such as task arrival time, task priority, or any other relevant scheduling rule. By iterating through tasks systematically, the algorithm ensures that each task is considered for assignment.
3. For each task, select the best available resource based on a predefined criterion (e.g., shortest processing time): For each task in the iteration, the algorithm evaluates all available resources to determine the best one for the task. The criterion for selecting the best resource depends on the specific goals of the scheduling problem. Common criteria include minimizing processing time, maximizing resource utilization, or balancing workload across resources.
4. Assign the task to the selected resource: Once the best resource is identified for a task, the algorithm assigns the task to that resource in the schedule. This assignment represents a commitment to execute the task on the selected resource according to the scheduling decision made by the algorithm.
5. Continue until all tasks are assigned: The algorithm repeats steps 2-4 until all tasks have been assigned to resources. This ensures that every task is accounted for in the schedule, leaving no tasks unallocated. Once all tasks have been assigned, the algorithm completes its execution, and the final schedule with task-resource assignments is produced as the output.

Results: The Greedy Algorithm are presented, including job assignments and makespan. Additionally, the schedule detailing idle, start, and stop times for each job on each machine is provided as see in Figure1. runtime compared to the Integer Programming Solution. This algorithm was implemented using N-MACHINE = 4 and N-JOB = 5, N-JOB = 6, N-JOB = 7 i.e. number of jobs were change in each case to compare the best. Below f show the result in different cases

For the given problem instance, the greedy algorithm yielded a makespan of 36-time units. In contrast, simulated annealing produced a makespan of 27-time units, significantly improving the solution quality.

1. Using N-MACHINE = 4 and N-JOB = 5:

Makespan (Best time) using greedy algorithm: 27.
Optimal Makespan from ILP Solution: 35
Accuracy of greedy algorithm solution: 77.14285714285715 %. This

2. Using N-MACHINE = 4 and N-JOB = 6:

Makespan (Best time) using greedy algorithm: 30
Optimal Makespan from ILP Solution: 40
Accuracy of greedy algorithm solution: 75.0 %

3. Using N-MACHINE = 4 and N-JOB = 7:

Makespan (Best time) using greedy algorithm: 36
Optimal Makespan from ILP Solution: 43
Accuracy of greedy algorithm solution: 83.72093023255815 %
Schedule:

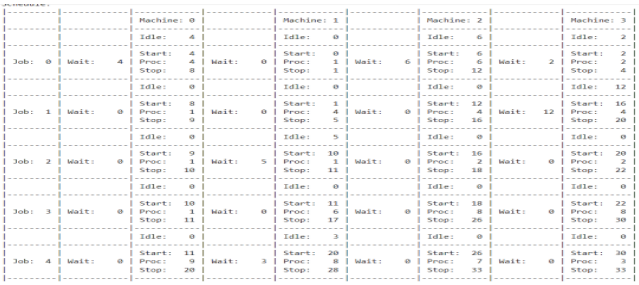

Figure 1: Schedule(N-JOB=5)

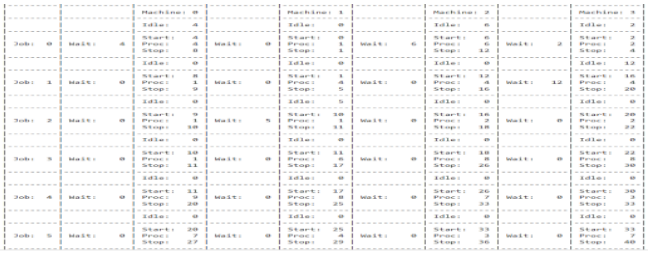

Figure 2: Schedule(N-JOB=6)

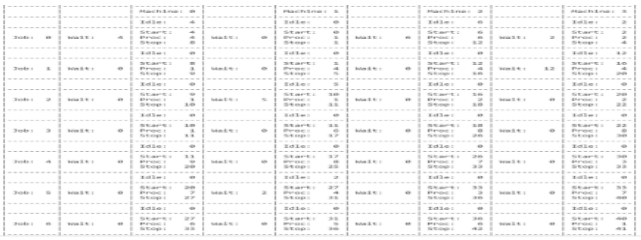

Figure 3: Schedule(N-JOB=7)

Stachastic Optimization (Simulated Annealing)

Simulated Annealing is a metaheuristic algorithm inspired by the annealing process in metallurgy [5]. It starts with an initial solution and iteratively explores neighbouring solutions, accepting, or rejecting them based on an acceptance probability function [6].

Algorithm outline:

1. Initialize a random solution as the current best solution: Begin by randomly generating an initial solution to the optimization problem. This solution serves as the starting point for the optimization process. Additionally, designate this solution as the

current best solution since it's the only solution available at the outset.

2. Initialize the temperature parameter and cooling schedule: Set up the initial temperature parameter and define the cooling schedule. The temperature parameter controls the probability of accepting worse solutions during the search process, while the cooling schedule determines how the temperature decreases over time.
3. Repeat until stopping criterion is met: The optimization process iterates until a stopping criterion is satisfied, indicating that the search should terminate. This criterion could be a maximum number of iterations, reaching a certain target objective value, or running for a specified amount of time.
4. Generate a neighbouring solution by applying a random perturbation to the current solution: At each iteration, generate a neighbouring solution by making a small perturbation to the current solution. This perturbation could involve swapping elements, modifying parameters, or other local modifications that produce a nearby solution in the solution space.
5. Calculate the change in objective function value between the current and neighbouring solutions: After generating the neighbouring solution, evaluate the change in the objective function value (e.g., completion time) between the current solution and the neighbouring solution. This change quantifies how much the new solution improves or worsens the objective compared to the current solution.
6. Accept the neighbouring solution with a probability determined by the change in objective function value and the current temperature: Use a probabilistic criterion, such as the Metropolis criterion, to decide whether to accept the neighbouring solution. The probability of acceptance depends on both the magnitude of the change in the objective function value and the current temperature according to a predefined schedule.
7. Update the temperature according to the cooling schedule: Adjust the temperature according to the cooling schedule. Typically, the temperature decreases over time to reduce the likelihood of accepting worse solutions as the search progresses. Common cooling schedules include linear cooling, exponential cooling, or adaptive schemes.
8. Update the current solution if the neighbouring solution is accepted and improves upon the current best solution: If the neighbouring solution is accepted based on the acceptance probability and it improves upon the current best solution found so far, update the current best solution to be the accepted neighbouring solution. This ensures that the search progresses towards better solutions over time.

Results: The Simulated Annealing algorithm's accuracy and runtime are assessed and compared to the Integer Programming Solution.

1. Using N-MACHINE = 4 and N-JOB = 5:

Makespan (Best time) using Simulated Annealing: 27
Optimal Makespan from ILP Solution: 35
Accuracy of Simulated Annealing Solution: 77.14285714 285715 %

2. Using N-MACHINE = 4 and N-JOB = 6:

Makespan (Best time) using Simulated Annealing: 27
Optimal Makespan from ILP Solution: 40
Accuracy of Simulated Annealing Solution: 67.5 %

3. Using using N-MACHINE = 4 and N-JOB = 7:

Makespan (Best time) using Simulated Annealing: 27
Optimal Makespan from ILP Solution: 43
Accuracy of Simulated Annealing Solution: 62.79069767 44186 %

Schedule:

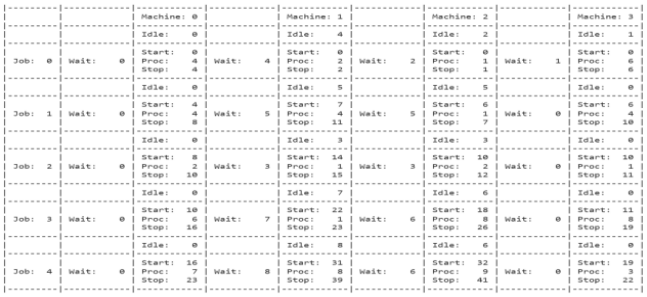

Figure 4: Schedule(N-JOB=5)

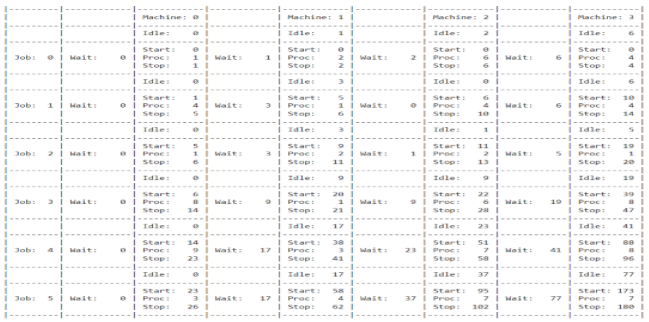

Figure 5: Schedule(N-JOB=6)

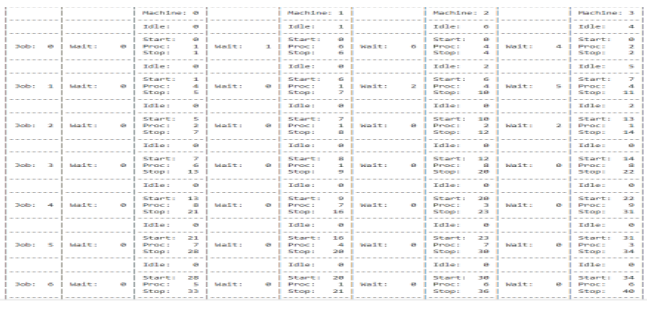

Figure :6 Schedule(N-JOB=7)

Evaluation

The performance of each method is evaluated based on the makespan, which represents the total time required to complete all jobs. The ILP approach provides an optimal solution, serving as the benchmark for evaluating the

effectiveness of heuristic methods Greedy algorithm and metaheuristic methods Simulated Annealing. The evaluation includes comparing the makespan and computational efficiency of each method.

Performance measurements for both algorithms, including relative accuracy and runtime, are compared against the IP solution provided in the Jupyter Notebook file. Below table show the result.

N-JOB	N-JOB=5	N-JOB=6	N-JOB=7
N-MACHINE	4	4	4
Optimal Makespan(IP)	35	40	43
Greedy Makespan	27	30	36
Simulated Annealing Makespan	27	27	27

Evaluation Metrics Result Table from the three instances.

conclution

In this study, we investigated the Job Scheduling Problem (JSP) optimization in a workshop setting using Greedy and Stochastic Optimization algorithms. The objective was to minimize the makespan while maximizing resource utilization. The performance of these algorithms was compared against an Integer Programming (IP) solution.

Results showed that while the IP solution guarantees optimality, both the Greedy Algorithm and Simulated Annealing provided competitive solutions with less computational complexity. The Greedy Algorithm demonstrated effectiveness in quickly generating near-optimal solutions, especially for smaller problem instances, albeit with slightly lower accuracy compared to Simulated Annealing. Simulated Annealing, on the other hand, offered consistent and improved performance across different problem sizes, showcasing its robustness in handling larger instances of the JSP.

Future Work

While this study provides valuable insights into heuristic and metaheuristic approaches for the JSP, there are several avenues for future exploration. Firstly, incorporating additional optimization techniques such as Genetic Algorithms or Tabu Search could further enhance solution quality and diversity. Additionally, exploring hybrid approaches that combine the strengths of different algorithms may yield even better performance. Moreover, extending the study to consider dynamic job arrival times, machine breakdowns, and other real-world constraints would make the results more applicable to practical manufacturing scenarios. Finally, investigating the scalability of these algorithms to handle larger problem instances and evaluating their performance in distributed computing environments could provide valuable insights for industrial implementations.

reference

[1] W. Li, Y. Wu, and M. Goh, *Planning and Scheduling for Maritime Container Yards: Supporting and Facilitating the Global Supply Network*. Springer International Publishing, 2015.

[2] L. Gui, X. Li, Q. Zhang, and L. Gao, "Domain Knowledge Used in Meta-Heuristic Algorithms for the Job-Shop Scheduling Problem: Review and Analysis," *Tsinghua Science and Technology*, vol. 29, no. 5, pp. 1368-1389, 2024, doi: 10.26599/TST.2023.9010140.

[3] W.-Y. Ku and J. C. Beck, "Mixed Integer Programming models for job shop scheduling: A computational analysis," *Comput. Oper. Res.*, vol. 73, pp. 165-173, 2016.

[4] G. Lan, G. W. DePuy, and G. E. Whitehouse, "An effective and simple heuristic for the set covering problem," *European Journal of Operational Research*, vol. 176, no. 3, pp. 1387-1403, 2007/02/01/ 2007, doi: https://doi.org/10.1016/j.ejor.2005.09.028

[5] Kirkpatrick, Scott, Charles D. Gelatt and Mario P. Vecchi. *Optimization by Simulated Annealing*. Science 220 (1983): 671 - 680.

[6] P. J. M. v. Laarhoven, E. H. L. Aarts, and J. K. Lenstra, "Job Shop Scheduling by Simulated Annealing," *Oper. Res.*, vol. 40, pp. 113-125, 1992.