International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Computational Study on Job Flow Shop Scheduling Using MiniMax Optimizing Algorithm

Margia Yesmin[1], Md. Abdul Alim[2]

[1]marzia.yesmin@ms.butex.edu.bd
[2]maalim@math.buet.ac.bd

Abstract
Shop scheduling is a critical part of manufacturing systems. The goal is to find an efficient schedule that optimizes system performance. Finding a feasible schedule becomes more difficult as the number of jobs and machines increases. The present research aims to study the computational study of different optimization methods to solve job flow shop scheduling problems. This paper reviews computational research of varying optimization methods to solve job flow shop scheduling problems. We discuss Johnson's algorithm, Campbell, Dudek, and Smith's (CDS) Approach, and Gantt chart and propose a Minimax optimization. This new method has the least processing time in the machine in the current situation. We present substantial computational results using MiniMax optimization techniques. From the result, we can conclude that the proposed algorithm gives less processing time and more frequently an optimal schedule than the other studied methods.

Keywords: Job flow shop scheduling, Johnson's algorithm, CDS Approach, Gantt chart, MiniMax Optimizing algorithm

1. Introduction

The job-shop scheduling problem involves scheduling n jobs to be processed by m dedicated machines. Each job must go through all the machines in a specific order. The time required for each job on each machine is predetermined and cannot be changed. Jobs cannot overlap on machines and no job can be processed simultaneously by multiple machines. It is not allowed to interrupt a job once it has started. The primary objective is to schedule the jobs in a way that minimizes the makespan, which is the maximum time taken by any job to complete. Despite being a highly researched optimization problem, solving the job-shop scheduling problem optimally remains a considerable challenge.

Some simplified versions of the job-shop scheduling problem are still classified as NP-Hard. For instance, the problem of scheduling three machines and three jobs with an arbitrary number of operations per job (where a job may have to visit a machine more than once) is NP-Hard [1]. Similarly, scheduling three machines and unitary processing times or three machines and no more than two operations per job is also NP-Hard [1, 4]. However, there are some particular cases of the job-shop scheduling problem that can be solved in polynomial time. For example, the problem of scheduling two machines and no more than two operations per job can be

solved in polynomial time [3], as well as the problem of scheduling two machines and unitary processing times [2].

According to the authors' understanding, all the proposed algorithms for solving the job-shop scheduling problem are branch-and-bound procedures. Although not an exhaustive list, some notable works include those of Applegate and Cook [10], Brucker et al. [5], Carlier and Pinson [6], Lageweg et al. [7], and Martin and Shmoys [8].

A schedule is a plan that allocates a specific time interval for each operation. However, a more helpful way of representing this is through the disjunctive graph model developed by Roy and Sussmann [9]. In this model, the disjunctive graph is represented as G = (0, A, E), where O is the set of vertices that corresponds to the operation set, A is the set of arcs that corresponds to the job precedence constraints, and E is the set of edges that corresponds to the machine capacity constraints.

Several branch-and-bound methods have been developed for solving the job-shop scheduling problem to optimality; the more recent ones were proposed by Carlier and Pinson [7,11], Applegate and Cook [10], and Brucker, Jurish, and Sievers [5]. These methods require a large amount of computation time and therefore it is not practical to apply them to many instances of even modest size. Carlier and Pinson [7] solved the famous 10-job 10-machine instance of the job-shop scheduling problem proposed by Fisher and Thompson [12], which remained unsolved for a long period and largely motivated the development of algorithms for this scheduling problem. Applegate and Cook presented 7 problems they could not solve optimally, where the smallest one has 10 machines and 15 jobs. Recently, Louren [13] proposed a new lower bound for the job-shop scheduling problem, which is a strengthening of one frequently used, and is based on solving a one-machine scheduling problem with additional constraints corresponding to minimal lags of time between the process of some pairs of jobs.

This paper proposes a new framework of job scheduling algorithm to decrease job completion time, improve the load balance, and satisfy users' priority demands. According to the result, the algorithms proposed in this paper outperform the Minimum processing time MPT optimizing algorithm in terms of makespan, load balancing, and user-priority awareness.

2. Combinatorial Optimization Problems

Optimization problems involve finding the best possible solution from multiple available solutions. Combinatorial optimization is an optimization problem that aims to discover the optimal solution from a finite set of solutions. It involves identifying an objective function's maximum or minimum value in a discontinuous domain with a vast configuration space. The traveling salesman problem, job-shop scheduling, and Boolean satisfiability are examples of combinatorial problems.

A combinatorial optimization problem is defined formally as a quadruple I, m, n, f where I is a set of instances, $m(x)$ contains all possible solutions, given an instance $x \in I, x$ is an instance and y is a viable solution, $x, h(x, y)$ is the measure of y which is usually a positive real number. f is the goal function either minimum or maximum.

The goal is to find an optimal solution y for some instance x with $h(x, y) = f\{h(x, y)|\ y' \in m(x)\}$.

Solution Techniques for Combinatorial Optimization Problems

- Exact Methods (Branch and Bound, Dynamic Programming, Integer Programming)

- Heuristics (Greedy algorithms, Rule-based algorithms)
- Metaheuristics (Genetic Algorithms, Simulated Annealing, Ant Colony Optimization)

2.1 Job flow shop problems

The Job Flow Shop scheduling problem is one of the most challenging scheduling problems manufacturers face. It is unique because it caombines elements of both flow shop and job shop environments. With multiple stages and one or more machines at each stage, all jobs must pass through these machines efficiently. The task of scheduling jobs across multiple machines at each stage in an efficient manner is a daunting task that requires careful planning and execution.

In a job flow shop, each stage may contain multiple identical or distinct machines, and the job processing order can vary from stage to stage. This setup complicates the scheduling process, requiring mathematical models to optimize job sequencing and machine allocation.

One common mathematical representation for hybrid flow shop scheduling is as follows:

Let:

- n be the number of jobs to be processed.
- m be the number of stages.
- m_j be the number of machines in stage j.
- p_{ij} represent the processing time of job i at stage j.
- C_{ij}be the completion time of job i at stage j.
- S_{ij} be the start time of job i at stage j.

Decision Variables:

Binary decision variable x_{ijk} represents whether job i is processed on the machine k at stage j.

$$x_{ijk} = \begin{cases} 1, & \text{if job } i \text{ is processed on machine } k \text{ at stage } j \\ 0, & otherwise \end{cases}$$

Objective Function: The objective is typically to minimize some measure of the schedule's performance, such as makespan (the time to complete all jobs) or total flow time.

Constraints:

1. Each job must be processed exactly once at each stage:

$$\sum_{k=1}^{m_j} x_{ijk} = 1, \quad \forall \ i,j$$

2. Each machine can process at most one job at a time:

$$\sum_{i=1}^{n} x_{ijk} \leq 1, \quad \forall \ j,k$$

3. Precedence constraints to ensure that the processing order of jobs is respected between stages.
4. Non-negativity constraints: $x_{ijk} \geq 0, \forall \ i,j,k$

These constraints ensure that each job is assigned to exactly one machine at each stage, each machine processes at most one job at a time, and the processing order of jobs is maintained between stages.

Solving the job flow shop problem involves finding values for the decision variables that minimize the objective function while satisfying all constraints. Various optimization techniques such as mathematical

programming (e.g., mixed-integer linear programming), heuristic algorithms (e.g., genetic algorithms, simulated annealing), or metaheuristic approaches (e.g., tabu search, ant colony optimization) can be employed for this purpose.

2.2 Johnson's Algorithm

Johnson's algorithm is a way to find the shortest paths between all pairs of vertices in an edge-weighted directed graph. It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist. It works by using the Bellman-Ford algorithm to compute a
transformation of the input graph that omits all negative weights, allowing Dijkstra's algorithm to be used on the transformed graph. It is named after Donald B. Johnson, who first published the technique in 1977.

A similar re-weighting technique is also used in Suurballe's algorithm for finding two disjoint paths of minimum total length between the same two vertices in a graph with non-negative edge weights.

Johnson's algorithm consists of the following steps:

First, a new node p is added to the graph, connected by zero-weight edges to each of the other nodes.

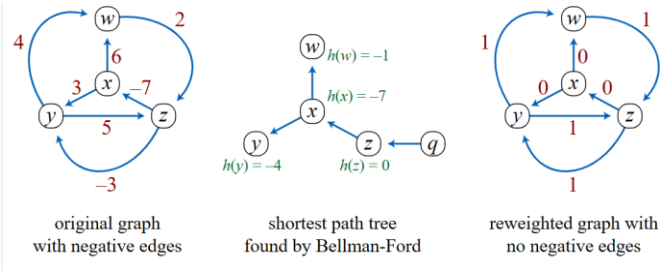
The next step involves implementing the Bellman-Ford algorithm which starts from the new vertex p. It aims to find the minimum weight g(v) of a path from p to every vertex v. However, if this step detects a negative cycle, the algorithm must be stopped.

The Bellman-Ford algorithm recomputes the shortest path from a source vertex to all other vertices but with negative edge weights allowed.

Finally, after removing the vertex p, Dijkstra's algorithm is applied to find the shortest paths from each node to all other vertices in the re-weighted graph. To compute the distance in the original graph, the value of g(v) - g(u) is added to the distance returned by Dijkstra's algorithm for each distance D(u,v).

Example:

The first three stages of Johnson's algorithm are depicted in the illustration below.



The graphic representation illustrates three graphs, each with unique characteristics. The first graph, located on the left, features negative edges but no negative cycles. The second graph portrays the shortest path tree calculated by the Bellman-Ford algorithm, with a new vertex 'p' as the starting point. The third graph is a re-weighted graph generated by substituting each edge weight with w(u,v) + g(u) - g(v). Notably, the re-weighted graph exhibits non-negative edge weights, and the shortest path between nodes is identical to that of the original graph.

To conclude, Dijkstra's algorithm is applied to the four starting nodes within the re-weighted graph. In the re-weighted graph, every path linking a pair of nodes s and t includes a uniform quantity added to it.

2.3 Campbell, Dudek, and Smith (CDS) Approach

The CDS heuristic is an extension of the two-machine scheduling approach, generalizing it to scenarios involving more than two machines.

CDS approach is a heuristic algorithm for solving flow shop scheduling problems. It works well for unrelated parallel machines. It was proposed by Campbell, Dudek, and Smith in 1985.

Here's a brief overview of the CDS approach:

1. Initialization: Begin by randomly generating an initial sequence of jobs.
2. Local Search: Use local search to improve the solution by iteratively swapping job pairs to minimize the makespan.
3. Neighborhood Structure: Neighborhood structure specifies which solutions are considered for local search. In CDS, swapping adjacent job pairs in sequence form the neighborhood.
4. Stopping Criterion**:** Define a stopping criterion for the search, such as a maximum iteration limit, a threshold for improvement, or a pre-defined computational boundary.
5. Restart Mechanism: Use a restart mechanism to explore different regions of the solution space by periodically restarting the search from different initial solutions.

CDS is a heuristic approach for flow shop scheduling, ideal for unrelated parallel machines. It finds near-optimal solutions quickly without using complex optimization methods. Though not optimal, CDS provides good-quality solutions promptly, making it suitable for real-time scheduling decisions with limited computational resources.

2.4 Gantt Chart

A generalized Activity Normalization Time Table (GANTT) chart is a type of chart in which a series of horizontal lines are present that show the amount of work done or production completed in a given period about the amount planned for those projects. It is a horizontal bar chart developed by Henry L. Gantt (American engineer and social scientist) in 1917 as a production control tool. It is simply used for the graphical representation of a schedule that helps to plan efficiently, coordinate, and track some particular tasks in a project.

The purpose of the Gantt chart is to emphasize the scope of individual tasks. Hence set of tasks is given as input to the Gantt chart. Gantt chart is also known as a timeline chart. It can be developed for the entire project or it can be designed for individual functions. In most projects, after the generation of the timeline chart, project tables are prepared. In project tables, all tasks are listed properly along with the start date and end date and information related to it.

Gantt chart represents the following things:

- Gantt chart is a horizontal bar chart used to represent operating systems. The horizontal bars indicate the required time by corresponding particular tasks.
- When occurring of multiple horizontal bars take place at the same time on the calendar. The diamonds indicate milestones

This chart is a horizontal bar chart used to represent operating systems scheduling in a graphical view that helps to plan, coordinate, and track specific CPU utilization factors like throughput, waiting time, turnaround time, etc.

3. Proposed MiniMax Optimizing Algorithm

The MiniMax optimizing algorithm is simple. It starts with a set S of all unmapped jobs J. Then the machine M which has the minimum completion time for all jobs is found. Next, the job J with the minimum size is selected and assigned to the corresponding machine R. Last, the job J is removed from set S and the same procedure is repeated until all tasks are assigned (i.e., set S is empty). The pseudo-code of MiniMax optimizing algorithm is represented assuming we have a set of n jobs (J1, J2, J3 … Jn) that need to be scheduled onto m available machines (M1, M2, M3 … Mm). We denote the Expected Completion Time for job k (1≤k≤n) on machines i (1≤i≤m) as CJ_{ik} that is calculated, where rJ_k represents the ready time of the machine M_i and Et_{ik} represents the execution time of jobs J_k on machine M_i.

Input:

Vertex: machines, jobs consequently

Edges: Processing step of the corresponding job on the corresponding machine

Weight: Processing time

Output:

Minimize the make-span of FSSP

- All jobs visit all machines
- No idle time for the machine
- Machines ≥ Jobs
- Ignored machine set-up time
- Machine $M_i; i = 1,2, \ldots . . m \ and \ Job \ J_{k;} \ k = 1,2, \ldots, n$

Algorithm:

For all submitted jobs in the set J_k;

For all machines M_i;

$CJ_{ik} = EJ_{ik} + rJ_k$;

End;

End;

Do while jobs set is not empty

Find job J_k that cost least processing time.

Assign J_k to machine M_i which gives least expected time

Remove J_k from the jobs set

Update ready timerJ_k for select M_i

Update C_{ik} for all J_k

End **Do**

4. Computational Results

Example 1**.** Solve 2 machines 6 jobs scheduling problems by (i) Jonhson's Algorithm (ii) Gantt Chart (iii) MiniMax Optimizing Algorithm

Job / Machine	J1	J2	J3	J4	J5	J6	Total
M1	1	3	8	5	6	3	26
M2	5	6	3	2	2	10	28

(i) Johnson's Algorithm:

If SPT (shortest processing time) is for 1st machine, do that job first and
If SPT is for 2nd machine, do that job last.
According to this algorithm, the job sequence will be

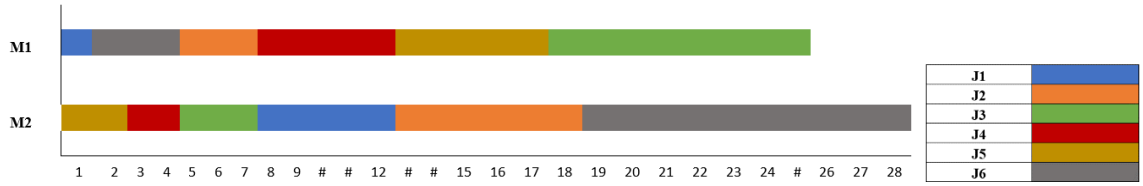
J1	J6	J2	J3	J4	J5

The In-Out table will be:

Machine / Job	M1		M2		Idle time M1	Idle time M2
	In	Out	In	Out		
J1	0	1	1	6	-	1
J6	1	4	6	16	-	-
J2	4	7	16	22	-	-
J3	7	15	22	25	-	-
J4	15	20	25	27	-	-
J5	20	**26**	27	**29**	3	-
				total	3	1

Total Elapsed time = 29
Idle time for M1= 3 = 29-26
Idle time for M2= 1 = 29-28

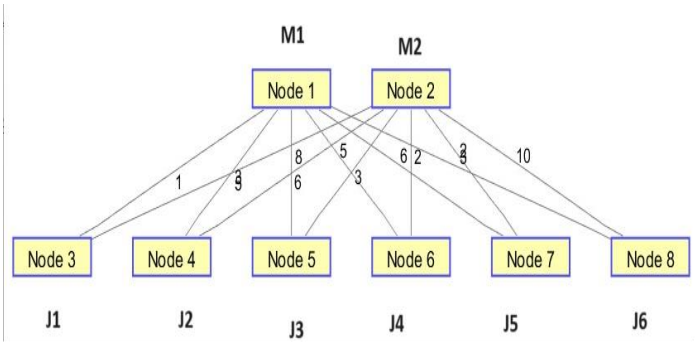
(ii) Gantt Chart:



According to the Gantt Chart
Total Elapsed time = 28
Idle time for M1= 2 = 28-26
Idle time for M2= 0

(iii) MiniMax Optimizing algorithm

We implement the new algorithm and then compare.



According to the proposed MiniMax optimizing algorithm, the table becomes:

Job Sequence↓	M1			M2		
	In	Out	Delay	In	Out	Delay
S1	J1	1	-	J4	2	-
S2	J2	1+3=4	-	J5	2+2=4	-
S3	J6	7	-	J3	7	-
S4	J4	12	-	J1	12	-
S5	J5	18	-	J2	18	-
S6	J3	**26**	2	J6	28	-

According to the MiniMax optimizing algorithm
Total Elapsed time = 28
Idle time for M1= 2 = 28-26
Idle time for M2= 0

Example-2:

Solve 3 machine 7 jobs scheduling problems by (i) Campbell, Dudek, and Smith (CDS) Approach (ii) Gantt Chart (iii) MiniMax Optimizing Algorithm

Jobs / Machine	J1	J2	J3	J4	J5	J6	J7	Total
M1	3	8	7	4	9	8	7	46
M2	4	3	2	5	1	4	3	22
M3	6	7	5	11	5	6	12	52

(i) Campbell, Dudek, and Smith (CDS) Approach:

Convert 3 machines into 2 machines:

$\min(t_{1j}) \geq \max(t_{ij})$ *or* $\min(t_{kj}) \geq \max(t_{ij})$; $i = 2,3,\dots,k-1$, $k = no.\ of\ machines$

Since the 2nd condition is satisfied, so according to this algorithm, we create two fictitious machines from 3, say MI and MII. Now

MI =$t_{1j} + t_{2j}$

MII =$t_{3j} + t_{2j}$

The problem is converted into 2 machine Problems:

job / machine	J1	J2	J3	J4	J5	J6	J7
M1	7	11	9	9	10	12	10
M2	10	10	7	16	6	10	15

According to Johnson's algorithm, the job sequence will be

J1	J4	J7	J2	J6	J3	J5

The In-Out table will be:

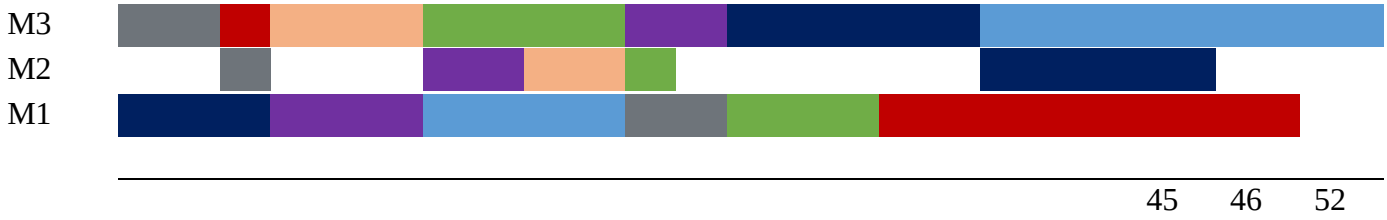
	M1		M2		M3		Idle time For		
Sequence	In	Out	In	Out	In	Out	M1	M2	M3
J1	0	3	3	7	7	13	-	3	7
J4	3	7	7	12	13	24	-	-	-
J7	7	14	14	17	24	36	-	2	-
J2	14	22	22	25	36	43	-	5	-
J6	22	30	30	34	43	49	-	5	-
J3	30	37	37	39	49	54	-	3	-
J5	37	46	46	47	54	59	13	19	-
				Total	delay	time	13	37	7

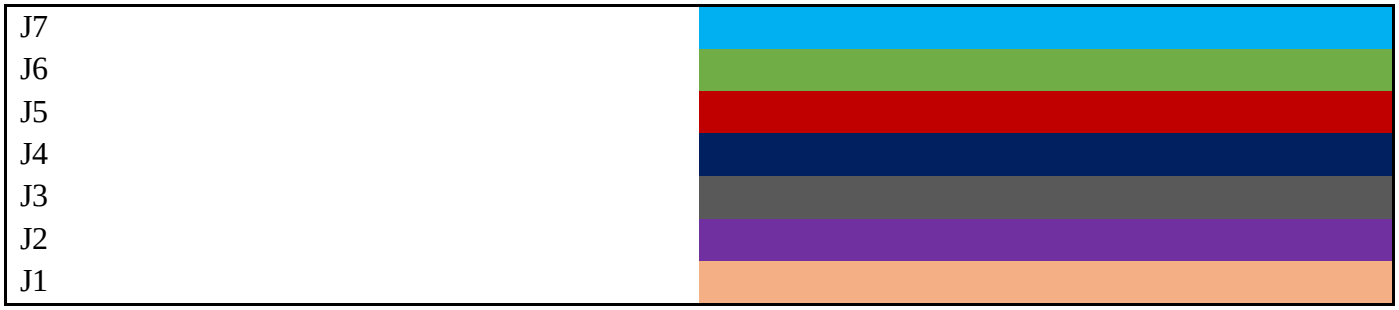
According to the CDS approach,

Total Elapsed time = 59

Idle time for: M1= 59-46=13, M2= 59=37, M3=59-52=7

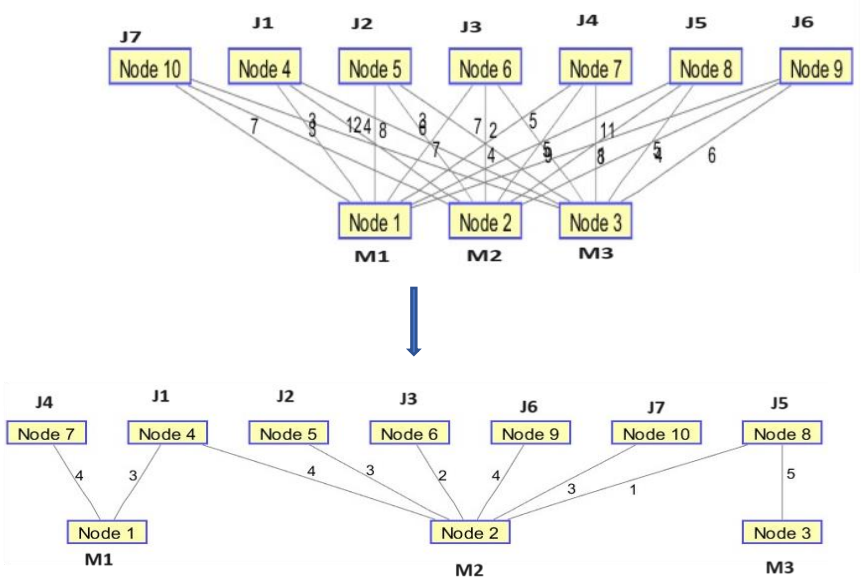
(ii) Gantt Chart:





According to the Gantt Chart
Total Elapsed time = 52
Idle time for M1= 52-46= 6
M2= 4+5+14+7= 30
M3= 52-52= 0

(iii) MiniMax Optimizing Algorithm:



The In-Out table will be:

Job sequence↓	M1			M2			M3		
	In	Out	Delay	In	Out	Delay	In	Out	Delay
S1	J1	3	-	J5	1	-	J3	5	-
S2	J4	7	-	J3	7	4	J5	10	-
S3	J3	14	-	J7	10	-	J1	16	-
S4	J7	21	-	J2	13	-	J6	22	-

S5	J2	29	-	J6	26	9	J4	33	-
S6	J6	37	-	J1	30	-	J2	40	-
S7	J5	46	6	J4	35	17	J7	52	-
	Total	46	6		35	30		52	0

According to the MiniMax optimizing algorithm
Total Elapsed time = 52
Idle time for: M1= 6, M2= 30, M3= 0

5. Comparison and Analysis

1. Comparison tables for 2 machines n jobs and 3 machine n jobs:

Table-1

	Total elapsed time	Idle time for M1	Idle time for M2
Johnson's algorithm	29	3	1
Gantt chart	28	2	0
MiniMax optimizing algorithm	28	2	0

2. machines n jobs comparison

Table-2

	Total elapsed time	Idle time for M1	Idle time for M2	Idle time for M3
CDS Approach	59	13	37	7
Gantt chart	52	6	30	0
MiniMax optimizing algorithm	52	6	30	0

3. machines n jobs comparison

This comparative analysis aimed to evaluate the efficiency and cost of a production factory. Efficiency was measured based on processing time and cost evaluations. The results indicated that the Gantt chart and MiniMax optimizing algorithm were less time-consuming compared to other algorithms. Additionally, both the Gantt chart and MiniMax optimizing algorithm produced the same result, but the MiniMax optimizing technique was easier to calculate than the Gantt chart.

This study discovered that the MiniMax optimizing algorithm is the most efficient and cost-effective production factory solution, providing reduced processing time and superior long-term efficiency. It's perfect for those who prioritize long-term operational sustainability and user engagement.

6. Conclusion

This paper focuses solely on the MiniMax algorithm for job scheduling emphasizing makespan and user-priority. Other scheduling algorithms, such as Round Robin, Max-Min, and Genetic Algorithm (GA), could

be developed. However, there are many outstanding issues to address, such as job deadlines, the high heterogeneity of interconnection networks, the geographic location of jobs and machines, and other quality of service requirements that could be subjects of future research. Although the jobs in this paper are independent, they may have precedence relations in real-life scenarios. We plan to investigate and enhance this algorithm for such job types.

References

1. Brucker P. Scheduling algorithms. 2nd ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc.; 1998.
2. Hefetz N, Adiri I. An efficient optimal algorithm for the two-machines unit-time job shop schedule-length problem. Mathematics of Operations Research 1982;7: 354–60.
3. Jackson JR. Scheduling a production line to minimize maximum tardiness. Technical report, Management Science Research Project, University of California. Los Angeles; 1955.
4. Lenstra JK, Rinnooy Kan AHG, Brucker P. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1977;1:343–62.
5. Brucker P, Jurisch B, Sievers B. A branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics 1994;49:109–27.
6. Carlier J, Pinson E., "An algorithm for solving the job-shop problem", Management Science 1989;35:164–76.
7. Lageweg BJ, Lenstra JK, Rinnooy Kan AHG. Job-shop scheduling by implicit enumeration. Management Science 1977;24:441–50.
8. Martin P, Shmoys DB. A new approach to computing optimal schedules for the job-shop scheduling problem. In: Proceedings of the fifth international IPCO conference on integer programming and combinatorial optimization. London, UK: Springer-Verlag; 1996. p. 389–403.
9. Roy, B., and Sussmann, B., "Les probl~mes d'ordonnan- cement avec contraintes disjonctives", Notes DS no. 9 bis, SEMA.
10. Applegate, D., and Cook, W., "'A computational study of the job-shop scheduling problem", ORSA Journal on Computing 3/2 (1991) 149-156.
11. Carlier, J., and Pinson, E., "A practical use of Jackson's preemptive schedule for solving the job shop problem", Annals of Operations Research 26 (1990) 260-287.
12. Fisher, H., and Thompson, G.L., "Probabilistic learning combinations of local job-shop scheduling rules", in: J. Muth and G. Thompson (eds.) Industrial Scheduling, Prentice-Hall, Englewood Cliffs, NY, 1963, 225-251.
13. Louren~o, H.R. "A computational study of the job-shop and the flow-shop scheduling problems", Ph.D. Thesis, School of OR& IE, Cornell University, Ithaca, NY, 1993.