AN INNOVATIVE GENETIC ALGORITHM-BASED MASTER SCHEDULE TO OPTIMIZE JOB SHOP SCHEDULING PROBLEM

Thi Phuong Quyen Nguyen*, Thi Hoa Thuong Le

The University of Danang - University of Science and Technology, Vietnam

*Corresponding author: ntpquyen@dut.udn.vn

(Received: September 27, 2024; Revised: October 29, 2024; Accepted: November 01, 2024)

DOI: 10.31130/ud-jst.2024.422E

Abstract - The rapid development of manufacturing sectors in the contemporary Industry 4.0 era has prompted enterprises to adopt increasingly systematic, streamlined, and efficient organizational structures. In this scenario, production scheduling has arisen as an essential function that contributes to the effective allocation of organizational resources in the production process, while shortening production time and satisfying customer deadlines. Consequently, the objective of this study is to develop an advanced method to optimize the production scheduling problem. Accordingly, this study proposes an innovative genetic algorithm-based master schedule (GA-MS) focused on minimizing makespan. The proposed method is implemented in various benchmark datasets on job shop scheduling problems. Compared with some benchmark methods such as priority rules-based approaches, branch and bound algorithm (B&B), and shifting bottleneck algorithm (SB), the proposed GA-MS shows an outstanding performance concerning the tested datasets. Its application in practical manufacturing factories is highly recommended.

Key words – Job shop scheduling problem; genetic algorithm; master schedule.

1. Introduction

In today's competitive environment, production scheduling plays a very important role in the sustainability of enterprises within the marketplace. To fulfill customer demands, such as timely delivery, organizations must develop a precise strategy, adeptly allocate, and utilize available resources effectively and reasonably. If a firm underestimates the importance of production scheduling, it may face many difficulties in its operations, resulting in resource inefficiency, reduced productivity, and substantial cost increases [1].

There are many types of production scheduling models such as single machine scheduling model, flow shop, job shop scheduling model, open shop, and so on. However, the job shop scheduling problem (JSSP) is the most popular one in practical issues. JSSP is a production or service model in which each job has a separate route through different machines or equipment. In the JSSP model, there is always a conflict over resources such as human resources and equipment. Thus, JSSP is a difficult problem among all types of scheduling problems [2].

The JSSP in manufacturing industries is characterized by multifunctional machinery and equipment that produces multiple products with a wide range of features processed on a variety of machines, each of which can process a large number of details [3]. For example, printed circuit boards in the semiconductor industry are often produced in the form of job shops; orders are usually a certain batch of products, carried out through a given route with specific execution times.

The JSSP is classified as the NP-hard (Non-Polynominal-hard) problem and cannot be solved like normal linear programming problems [4]. Various meta-heuristics approaches have been proposed to solve JSSP, such as genetic algorithms (GA) [5, 6], evolutionary algorithms [7, 8], ant colony optimization [9], and particle swarm optimization (PSO) [10]. Among these meta-heuristics approaches, GA is the most popular one for solving optimization problems [11].

Thus, this study proposes an innovative genetic algorithm-based master schedule (denoted as GA-MS) to optimize the JSSP. A master schedule (MS) is a matrix that shows the sequence of jobs processed in each machine. Each MS is coded as a chromosome. An innovative method is developed to generate numerous MS for the initialization of GA procedure in which each MS can avoid infeasible solutions due to the job's sequence constraint. Thereafter, the genetic operation is implemented to determine the optimal solution to minimize makespan for JSSP.

2. Related works

2.1. Overview of JSSP

Table 1. *Notations of JSSP*

n	Number of jobs
m	Number of machines
i	Index of machines, i=1, 2,…, m
j	Index of jobs, j=1, 2,…, n
p_{ij}	Processing time of job j on machine i
x_{ij}	Starting time of job j on machine i
r_j	Release date/ Ready date of job j
d_j	Due date of job j
C_j	Completion time of job j
C_{max}	Make-span C_{max}= max{C_j: j = 1, 2, …, n}
F_j	Flow time job j
L_j	Lateness ($L_j = C_j – d_j$)
T_j	Tardiness T_j = max (0, L_j).

The classical JSSP is identified in manufacturing in which n different jobs/products are to be scheduled on m different machines, subject to two main sets of constraints which are the precedence constraints and the conflict constraints. Each job has a different sequential operation. The processing time of each job on machines is known, consistent, and independent of the schedules on machines.

The objective of JSP is to sequence jobs on machines and specify the starting time and ending time of each job to minimize certain performance. In this case, the objective function is to minimize makespan. Some notations of JSSP are listed in Table 1.

The problem can be modeled as follows:

Objective function

$$\text{Min } C_{max} \tag{1}$$

S.t.

$$\sum_{\substack{j'=1 \\ j' \neq j}}^{n} Yijj' = 1 \tag{2}$$

$$\sum_{\substack{j=1 \\ j \neq j'}}^{n} Yijj' = 1 \tag{3}$$

$$x_{ij} + p_{ij} \leq x_{i'j} \tag{4}$$

$$x_{ij} + p_{ij} \leq x_{ij}' + M(1 - Y_{ijj'}) \tag{5}$$

$$x_{ij'} + p_{ij'} \leq x_{ij} + MY_{ijj}' \tag{6}$$

$$x_{ij} \geq r_j \geq 0 \tag{7}$$

$$C_j = x_{lj} + p_{lj} \tag{8}$$

$$Y_{iii'} = \begin{cases} 1 & \text{if job } j \text{ is before job } j' \text{ on machine } i \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Eq. (1) is used for the objective function that is minimizing make-span C_{max}= max{C_j: j = 1, 2, …, n}. Eq. (2) indicates that on each machine i, when a job completes its processing, only one job in the set of available jobs is selected for processing. Eq. (3) means an operation of a job should follow only one processor. Eq. (4) is used for the precedence constraints. The precedence constraints ensure that each operation of a certain job is processed sequentially. Eq. (5) and (6) present the conflict constraints where M is a constant that is assumed to be a big number. The conflict constraints guarantee that each machine processes only one job at a time. Eq. (7) is to make certain that any job cannot start before its ready time. Eq. (8) is a formulation to identify the completion time of each job on each machine. Variable "l" in Eq. (8) is the last machine that product j is processed on. Eq. (9) explains the binary variable $Y_{ijj'}$. $Y_{ijj'} = 1$ if job j is produced before job j' on machine i. Unless, $Y_{ijj'}$ will get a value of 0.

2.2. Priority rules-based methods for scheduling problems

This section introduces some methods to solve the scheduling problems based on the priority rules. A priority rule is a type of policy that determines a specific sequencing choice for each time a machine idles. A priority rule is a type of policy that determines a specific sequencing choice for each time a machine idles [12]. Some priority rule-based methods can be listed as follows:

(1) *Earliest Due Date (EDD):* The job with the earliest due date is selected to be executed first in which the goal is to minimize maximum lateness.

(2) *Longest Processing Time (LPT): the* sequence of jobs depends on its processing time. The job with the largest processing time will be scheduled first. This rule is usually applied in parallel machine models to balance the workload across machines.

(3) *Weight Shortest Processing Time (WSPT):* The job with the largest ratio (w_j/p_j) is done first. This rule aims to minimize the total weighted completion time $\sum wjCj$. When all weights are equal, the WSPT rule becomes the SPT rule.

(4) *Minimum Slack (MS):* When a machine is idle at time t, the remaining slack time of each job at that time t is defined as:

$$Slack = (d_j - p_j - t) \tag{10}$$

Slack will be negative for late jobs, zero for on-time jobs, and positive for early jobs.

(5) *Shortest Set-up Time (SST):* This rule selects the job that has the smallest set-up time to implement first.

(6) *Least Flexible Job (LFJ):* This rule selects the job with the least flexibility to process (the least flexible job can only be executed on a certain number of machines or processed by the fewest number of machines). This rule is suitable for the heterogeneous parallel machine model.

(7) C*ritical Path (CP):* Select the job on the critical path to perform first, suitable for jobs with precedence constraints.

(8) *Largest Number of Successors (LNS):* Choose the job with the longest list of following jobs to do first.

2.3. Heuristic methods for JSSP

Branch and Bound (B&B): The B&B [13] constitutes an enumeration technique whereby all conceivable job sequences are cataloged within a hierarchical structure. Subsequently, the array of scheduling tables is sequentially eliminated by demonstrating that the target values associated with all schedules exceed a predetermined lower bound. This lower bound is defined as being greater than or equal to the target value of a previously attained schedule. The B&B is extensively employed to ascertain an optimal solution to the scheduling problem. Nonetheless, it is characterized by significant time consumption, as the number of nodes is often very large.

Genetic Algorithm (GA): GA [5, 6] aims to develop solutions that outperform their parents by utilizing diverse methodologies. GA begins with the initialization step, which simultaneously creates a population with many solutions. The best solutions are then selected from the generated alternatives. Thereafter, genetic operators, including crossover and mutation processes, are used for these selected individuals to generate offspring under the principle of an evolutionary process, i.e., the next generation is always better than the previous generation. This iterative procedure persists until an optimal solution is attained.

Particle Swarm Optimization Algorithm (PSO): PSO [10] is an algorithm designed to address optimization problems on an intelligent population model or intelligent swarm. PSO is a straightforward but efficient technique for optimizing continuous nonlinear objective functions. It has been effectively applied to solve numerous function extremum problems as well as multi-objective problems.

This study uses GA-based MS to optimize the JSSP. GA is selected among other heuristic methods due to its effectiveness has been proven in many optimization problems [5, 6].

3. Proposed GA-MS for JSSP

The idea for the proposed GA-MS to solve the JSSP is developed based on the MS. Each MS is a solution for the JSSP, i.e., a sequence of jobs to be processed on machines. The procedure of the proposed GA-MS begins with the initialization process based on designing and generating the MS. Then, the proposed GA-MS will follow the general procedure of genetic operation until obtain optimal solution. The details will be presented in the following subsections.

3.1. Master Schedule

MS is a combination of schedules for each job/product on each machine in the production process. Note that the processing time is known, consistent, and independent of the schedules on machines. Therefore, the result of the JSSP is to know the schedules for jobs/products on all machines, which is called the MS. However, each product has to follow its processing sequence which is known as precedence constraint. Thus, it is necessary to consider how one machine's schedule will affect the schedules of the remaining machines when generating a feasible MS.

An example is given to illustrate how to generate an MS. Given a JSSP with 3 jobs and 2 machines, the sequence of performing each job on the machines is shown in Table 2. The number in a row in Table 1 represents the order of machines that process jobs. For instance, Job 1 (J1) is processed on M1 first, then goes to M2. The sequence of Job 2 (J2) is on M2 first then moves to M1.

Table 2. *Sequence matrix in JSSP*

Sequence		Machine	
		M1	M2
Job	J_1	1	2
	J_2	2	1
	J_3	1	2

The processing time of a job on one machine is shown in Table 3. It is identified based on the job row and machine column. For instance, the processing time of J1 on M1 is 14; the processing time of J2 on M1 is 6.

Table 3. *Sequence matrix in JSSP*

Processing Time		Machine	
		M1	M2
Job	J1	14	16
	J2	6	4
	J3	2	8

Table 4 illustrates an MS. In the MS matrix, the schedule on one machine is represented in an equivalent column of the machine. For instance, the sequence of jobs processed on machine M1 is J2-J1-J3, and on machine M2 is J1-J2-J3.

Table 4. *An illustration of an MS*

Master Schedule		Machine	
		M1	M2
Job	J1	2	1
	J2	1	2
	J3	3	3

Referencing the MS in Table 3, the solution is infeasible because the schedule conflicts with the processing sequence in Table 1. For instance, M1 cannot process J2 first since it has to wait for the completion of J2 on M2. However, according to the MS, J2 cannot be processed on M2 because M2 has to process J1 first. Similarly, J1 cannot be processed on M2 due to the processing sequence and J1 needs to go to M1 first. As a result, the schedule is blocked.

Thus, this study develops a method to generate an MS that can avoid infeasible solutions. The concept to generate the feasible MS is to select one machine as the "Main Machine". Schedules on other machines are based on the schedule on the Main Machine and the sequence matrix.

For the above example, suppose that we select M1 as the Main Machine, and assign the schedule on M1 as Table 4. Then, we will generate the schedule on M2 as follows:

Step 1: Consider the column M2 of the sequence matrix, only J2 is available to be processed in M2, hence assign J2 to M2. The result is illustrated in Table 5.

Table 5. *An illustration to generate a feasible MS*

Sequence		Machine		**Master Schedule**		Machine	
		M1	M2			M1	M2
Job	J1	1	2	Job	J1	2	
	J2	2	1		J2	1	1
	J3	1	2		J3	3	

Step 2: Due to Step 1, J2 is assigned to be the first in M2; hence J2 can be processed on M2, then on M1. When M1 completes J2, it will continue with J1 without violating the J1 processing sequence. After J1 is completed, J3 is processed on M1 as scheduled on M1 is J2-J1-J3. The current schedule is presented in Table 6. In the sequence matrix, "0" means already finished jobs while "1" means available jobs to be processed.

Table 6. *An illustration to generate a feasible MS*

Sequence		Machine		**Master Schedule**		Machine	
		M1	M2			M1	M2
Job	J1	0	1	Job	J1	2	
	J2	0	0		J2	1	1
	J3	0	1		J3	3	

Step 3: Both J1 and J3 are available, we assign them in M2 based on the available time rule, which is the sooner the product is available, the higher the probability to be assigned before the others. This rule can enhance the diversity of solutions to large-scale problems. The schedule shown in Table 7 is a feasible MS.

Table 7. *An illustration to generate a feasible MS*

Sequence		Machine		**Master Schedule**		Machine	
		M1	M2			M1	M2
Job	J1	0	1	Job	J1	2	2
	J2	0	0		J2	1	1
	J3	0	1		J3	3	3

3.2. Proposed GA-based MS

3.2.1. Chromosome representation

A chromosome is coded as a feasible MS. According to Section 3.1, a feasible MS is a matrix type where the schedule on each machine is shown in the path representation. For instance, the sequence in M1 in Table 7 is $J_2 \rightarrow J_1 \rightarrow J_3$. Because of precedence constraints, each job will follow its processing sequence. It leads to the schedules in the path representations that have different length sizes. Table 8 illustrates this case.

Table 8. *An MS with different length sizes in each machine*

Machine	Job			
M1	J1	J3	J4	J2
M2	J4	J2	J3	
M3	J2	J3	J1	J4

Thus, a feasible MS should be coded in the matrix representation with the length consistently. From Table 8, an MS with the same is transformed and shown in Table 9. The number in Table 9 shows the order of jobs that are processed in each machine. For instance, J1 is processed in M1 first, thus the cell M1J1=1. J1 does not go to M2, so the cell M2J1=0. Similarly, J1 is executed in M1 in the 3rd order. Thus, the cell M3J1 gets a value of 3. This way is used to transform all jobs in Table 8 to Table 9.

Table 9. *An MS in the matrix representation*

Machine	Job			
	J1	J2	J3	J4
M1	1	4	2	3
M2	0	2	3	4
M3	3	1	2	4

3.2.2. Initialization

Each MS represents a chromosome. To generate a population-based MS, the study designs to code a number that is equivalent to a schedule on each machine. Following the process to generate an MS, after choosing a Main Machine, the sequence of jobs processed on the Main Machine is determined. The number of possible schedules also has to be defined on the Main Machine. For example, if we have a job list including *n* jobs that will be processed on the Main Machine; it means we will have *n* factorial permutation schedules of *n* jobs. Each possible schedule is equivalent to one unique value. Based on the population situation of each iteration, each individual (chromosome) is represented by one value. Converting the value of an individual into one unique schedule on the Main Machine will be followed in these steps.

Step 1

Order the job list including *n* jobs

Pick up **Value**

Calculate **Order** = roundup (Value/(n-1)!)

Select a job on the list as the first order at machine.

Step 2

Update Job List: (n-1) jobs

Order the remaining job list

Calculate **Update Value** = previous Value - (n-1)! *(previous Order-1)

Calculate **Update Order** = roundup (Update Value/((n-1)-1)!)

Select a job on the list as the second order at machine

Step 3: Repeat step 2 until the final job on the list

Table 10 shows the possible schedule of 4 jobs on the Main Machine. The schedules of these jobs in other machines are determined based on the process of generating an MS. Then, all possible MSs are used for the initial population.

Table 10. *Schedule on Main Machine with n=4*

Value	**Schedule**	**Value**	**Schedule**
1	1 – 2 – 3 – 4	13	3 – 1 – 2 – 4
2	1 – 2 – 4 – 3	14	3 – 1 – 4 – 2
3	1 – 3 – 2 – 4	15	3 – 2 – 1 – 4
4	1 – 3 – 4 – 2	16	3 – 2 – 4 – 1
5	1 – 4 – 2 – 3	17	3 – 4 – 1 – 2
6	1 – 4 – 3 – 2	18	3 – 4 – 2 – 1
7	2 – 1 – 3 – 4	19	4 – 1 – 2 – 3
8	2 – 1 – 4 – 3	20	4 – 1 – 3 – 2
9	2 – 3 – 1 – 4	21	4 – 2 – 1 – 3
10	2 – 3 – 4 – 1	22	4 – 2 – 3 – 1
11	2 – 4 – 1 – 3	23	4 – 3 – 1 – 2
12	2 – 4 – 3 – 1	24	4 – 3 – 2 – 1

3.2.3. Evaluation of Fitness Value

The study considers minimizing the value of makespan. Makespan is the total time required to complete a set of jobs from the beginning of the first job to the end of the final job.

$$\text{Make-span } C_{max} = \max\{C_j: j = 1, 2, \ldots, n\} \tag{10}$$

3.2.4. Genetic operation

The genetic operation includes the selection, crossover, and mutation process. The proposed GA-MS employs the Roulette Wheel method for the selection process to choose the parent chromosomes for producing the next generation. Regarding the crossover process, the traditional one-point crossover is utilized in which a random position in two selected chromosomes is picked up to perform crossover. Similarly, the mutation process is performed on a schedule of the Main Machine by randomly selecting two jobs and swapping two jobs at selected locations.

The offspring are produced after the genetic operation. The process is repeated to evaluate the fitness values of the offspring and continue with the genetic operation to reproduce the next generation. This process is repeated until the stopping criteria are met.

Some advances are designed in the proposed GA-MS. First, parameters are adjusted to encourage diversity during early iterations, thereby enhancing the exploration of the global optimum. Additionally, to improve exploitation to the local optimum, 10% of the population is replaced by very good solutions when the total number of iterations exceeds 90% or the best solution consistently persists for 40% of the total iterations. This strategy enables the population to conduct a dense search for the very best answers. To cut down on running time, the program is designed to self-terminate if the optimal solution remains unchanged after 50% of the total iterations.

4. Experimental Result

4.1. Parameter setting

To implement the proposed GA-MS, some parameters need to be set up such as the number of iterations (ItNum), population size (Popsize), probability of crossover: (Prob_CO), probability of Mutation (Prob_Mut). The values of ItNum and Popsize are chosen according to the experimental results of some previous research. The crossover rate Prob_CO is from 0.8 to 1.0. A high crossover rate is selected to lead to premature convergence. The mutation rate Prob_Mut is usually from 0.005 to 0.1. In this case, the high mutation rate is selected to enhance diversity. Finally, we experiment with multi-trials and determine the parameter for the proposed GA-MS as follows: ItNum = 200, Popsize = 100, Prob_CO = 0.8, and Prob_Mut = 0.1

4.2. Dataset

Ten tested datasets collected from OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html) are employed to implement and evaluate the performance of the proposed GA-MS. These datasets are usually used as the benchmarks for JSSP. The tested datasets can be classified into three types: 1) small-scale problems with $n, m \leq 4$; 2) medium-scale problems with $4 < n, m \leq 10$; 3) large-scale problems with $n, m > 10$. There are three small-scale datasets, four medium-scale datasets, and three large-scale datasets.

4.3. Result analysis

To evaluate the effectiveness of the proposed GA-MS, some benchmark algorithms are also used to compare its performance. This study selects the EDD rule since it is the most popular one among other priority rule-based methods. Besides, B&B and shifting bottleneck (SB) methods are also chosen as benchmark algorithms due to their simple but straightforward and efficient in solving JSSP.

Table 11. *The experimental result*

Dataset	EDD	B&B	SB	GA-MS
1	32	**28**	**28**	29
2	35	32	31	**26**
3	38	33	33	**30**
4	461	**262**	**262**	273
5	215	138	150	**131**
6	142	110	118	**104**
7	837	790	798	**736**
8	1286	1022	1022	**983**
9	1854	1682	1676	**1392**
10	2717	2523	2598	**2152**

The proposed GA-MS is implemented 20 times on each tested dataset and its average result is obtained. Table 11 shows the results of the proposed GA-MS with the three benchmark algorithms. The value in Table 11 is the makespan in which the smaller makespan is the better algorithm. The makespan was computed for each method based on Eq. (10) with its notation presented in Table 1. The result in Table 11 exhibits that the best makespan values vary based on the scale of problems. For small-scale problems, which are shown by the result of datasets 1 to 3, it is obvious that the GA-MS outperforms the other benchmarks since it achieves two best makespans on datasets 2 and 3. The B&B and SB algorithms rank the second order with the best makespan on dataset 1. EDD cannot obtain any best makespan on small-scale datasets. However, there is not a big difference between these algorithms in terms of makespan in these datasets. For example, the best makespan of dataset 2 provided by GA-MS algorithms is 26 while the worst makespan obtained by EDD is 35. The gap in makespan between GA-MS and B&B is also small with only 6.

Figure 1. *The comparison results on different datasets*

Regarding the medium-scale problems from datasets 4 to 7, the proposed GA-MS is still the best one since it obtains the best performance on 3 datasets. Especially, the difference in makespan between the proposed GA-MS and EDD method is relatively large. This illustrates the effectiveness of the proposed GA-MS when the scale of problems increases. The result is similar to the large-scale problems shown in datasets 8 to 10. It is noted that as problem sizes rise, the makespan gaps between the suggested GA-MS and B&B and GA-MS and SB also widen. Generally, heuristic methods such as EDD, B&B, and SB are ineffective for large and complicated problems. The proposed GA-MS can find the optimal solutions for these JSSPs quickly and efficiently. The computational time to implement the proposed GA-MS is also promising. It only takes from few seconds for the small-scale dataset to less than 5 minutes for the large-scale datasets. Figure 1 illustrates the experimental result for better visualization.

Figure 2 exhibits the schedule of a JSSP obtained by the proposed GA-MS for illustration. In this example, the dataset is medium-scale with 10 jobs and 6 machines. The jobs are consecutively assigned to machines. For instance, the sequence of jobs on machine 1 is J4-J10-J5-J1-J8. The makespan obtained by the proposed GA-MS is 131.

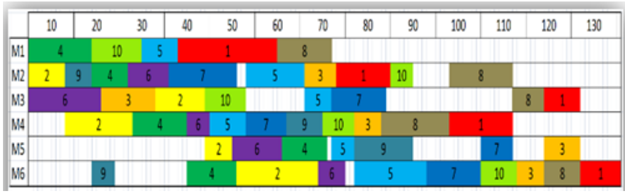

Figure 2. *A schedule of a JSSP obtained by GA-MS*

5. Conclusion

The analytical results obtained in this study demonstrate that the implementation of GA-MS in addressing the JSSP to minimize makespan has yielded more favorable outcomes in comparison to those produced by empirical algorithms such as B&B and SB. However, due to the nature of the genetic and search mechanism, the proposed GA-MS is likely to identify superior solutions but does not guarantee optimality in every instance. This algorithm is capable of addressing many problems characterized by varying scales and significant complexity, effectively tackling challenges associated with large-scale problems that are difficult to handle when using the B&B and SB algorithms.

The JSSP can be extended in future research with multi-objective functions such as minimizing maximum lateness and makespan simultaneously or considering extending the objectives functions such as minimizing tardiness or maximizing resource utilization. Besides, the comparison result can be performed on various optimization techniques such as Particle Swarm Optimization or Simulated Annealing. Moreover, there are various constraints on the expansion of the problem. For instance, it still assumes that there are always enough raw materials during the production process. It is also expected that the orders will not encounter any problems throughout this period. Thus, future research can consider extending these constraints to increase the applicability of the JSSP.

REFERENCES

[1] S. Nahmias and T. L. Olsen, *Production and operations analysis*. Waveland Press, 2015.

[2] H. Xiong, S. Shi, D. Ren, and J. Hu, "A survey of job shop scheduling problem: The types and models", *Computers & Operations Research*, vol. 142, p. 105731, 2022.

[3] S. Sršen and M. Mernik, "A JSSP solution for production planning optimization combining industrial engineering and evolutionary algorithms", *Computer Science and Information Systems*, vol. 18, no. 1, pp. 349-378, 2021.

[4] M. Abdolrazzagh-Nezhad and S. Abdullah, "Job shop scheduling: Classification, constraints and objective functions", *International Journal of Computer and Information Engineering*, vol. 11, no. 4, pp. 429-434, 2017.

[5] M. Boukedroun, D. Duvivier, A. Ait-el-Cadi, V. Poirriez, and M. Abbas, "A hybrid genetic algorithm for stochastic job-shop scheduling problems", *RAIRO-Operations Research*, vol. 57, no. 4, pp. 1617-1645, 2023.

[6] S. Tian, C. Zhang, J. Fan, X. Li, and L. Gao, "A genetic algorithm with critical path-based variable neighborhood search for distributed assembly job shop scheduling problem", *Swarm and Evolutionary Computation*, vol. 85, p. 101485, 2024.

[7] Q. Luo, Q. Deng, G. Xie, and G. Gong, "A Pareto-based two-stage evolutionary algorithm for flexible job shop scheduling problem with worker cooperation flexibility", *Robotics and Computer-Integrated Manufacturing*, vol. 82, p. 102534, 2023.

[8] Z.-Q. Zhang, F.-C. Wu, B. Qian, R. Hu, L. Wang, and H.-P. Jin, "A Q-learning-based hyper-heuristic evolutionary algorithm for the distributed flexible job-shop scheduling problem with crane transportation", *Expert Systems with Applications*, vol. 234, p. 121050, 2023.

[9] J. Zhang, X. Hu, X. Tan, J. H. Zhong, and Q. Huang, "Implementation of an ant colony optimization technique for job shop scheduling problem", *Transactions of the Institute of Measurement and Control*, vol. 28, no. 1, pp. 93-108, 2006.

[10] D. Sha and H.-H. Lin, "A multi-objective PSO for job-shop scheduling problems", *Expert Systems with Applications*, vol. 37, no. 2, pp. 1065-1070, 2010.

[11] C. K. H. Lee, "A review of applications of genetic algorithms in operations management", *Engineering Applications of Artificial Intelligence*, vol. 76, pp. 1-12, 2018.

[12] R. Haupt, "A survey of priority rule-based scheduling", *Operations-Research-Spektrum*, vol. 11, no. 1, pp. 3-16, 1989.

[13] C. Soto, B. Dorronsoro, H. Fraire, L. Cruz-Reyes, C. Gomez-Santillan, and N. Rangel, "Solving the multi-objective flexible job shop scheduling problem with a novel parallel branch and bound algorithm", *Swarm and evolutionary computation*, vol. 53, p. 100632, 2020.