algorithms

MDPI

Article

Evolutionary Optimization for Job Shop Scheduling with Blocking: A Genetic Algorithm Approach

John Valencia [1,*] and Elkin Rodríguez-Velásquez [2,*]

[1] Facultad de Ingenierías, Corporación Universitaria Remington, Medellin 050010, Colombia
[2] Facultad de Minas, Universidad Nacional de Colombia, Medellin 050034, Colombia
* Correspondence: john.valencia@uniremington.edu.co (J.V.); erodrigu@unal.edu.co (E.R.-V.)

Abstract

The Blocking Job Shop Scheduling Problem (BJSSP) is a variant of the classical Job Shop Scheduling Problem in which a job completed on one machine cannot be transferred to the next machine until the latter becomes available, causing the current machine to remain blocked. Numerous real-world applications have been modeled as the BJSSP, which is classified as a strongly NP-hard problem. Previous studies indicate that several proposed approaches fail to guarantee the generation of feasible solutions during the search process, thereby requiring a solution reconstruction. In this study, we propose a Genetic Algorithm (GA) designed to operate strictly within the feasible solution space of the BJSSP, where the objective function is the minimization of the makespan. Experimental results show that no specific factor levels significantly influenced the solution quality obtained by the GA across all problem sets. On the other hand, incorporating an assignment operator into the solution representation enhanced the diversity of the population. The proposed GA yields solutions that outperform some of the best-known makespan values for the Lawrence benchmark problems. The runtime of the GA ranged from 20 s for instances with 10 jobs and five machines to 600 s for instances with 30 jobs and 10 machines.

Keywords: blocking job shop scheduling; genetic algorithm; no buffer job shop scheduling

1. Introduction

The classical job shop problem is recognized as a fundamental scheduling problem and has been extensively studied over the past several decades [1]. However, many real-world industrial settings cannot be accurately represented as a classical job shop problem as they often involve additional features and constraints. As a result, researchers have increasingly focused on application-inspired constraints, such as setup times, limited buffer capacities, and machine flexibility, over the recent decades [2].

The blocking constraints indicate the complete lack of storage capacity between two consecutive machines. In this scenario, a job releases a machine only when the next machine it requires is ready for its operation. If not, the job remains on the current machine, preventing its use even after processing is finished [3]. There are two variations in the blocking-constrained job shop scheduling problem (BJSSP), depending on whether swaps among conflicting jobs are permitted or not.

Applications of the blocking job shop problem include scheduling for electronic manufacturing [4], pharmaceutical industries [5], steelmaking processes [6], and chemical batch production [7]. The problem also has applications in various fields beyond manufacturing,

Check for updates

Academic Editor: Frank Werner

Received: 31 December 2025
Revised: 23 January 2026
Accepted: 29 January 2026
Published: 1 February 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

particularly in logistics planning scenarios, where intermediate buffers are not permitted, such as container handling at ports or operations in railway networks [8].

Heuristics and metaheuristics have been proposed to solve the BJSSP. Ref. [9] defines a new scheduling problem using a combination of four buffering constraints (no buffer, no wait, limited buffer, and infinite buffer) and proposes a fast best insertion heuristic algorithm. Ref. [10] proposes a heuristic with Petri nets to schedule jobs in the context of manufacturing systems with blocking such as classical job shop, flexible job shop, and flexible manufacturing environments. Ref. [11] proposes a hybrid metaheuristic technique using Tabu Search and Simulated Annealing to solve a job shop scheduling problem with parallel machines in stages, such as those used in trains for transporting sugarcane between crops and mills in Australia.

Ref. [12] uses a genetic algorithm to define the number of machines to be used in a flexible job shop configuration with blocking and auto-guided vehicles. Then, they define different scenarios, determine the completion time of each job, and compare the results obtained. Ref. [13] defines a procedure based on dynamic programming for a two-stage supershop configuration with no-wait and blocking constraints. In this configuration, some jobs have flowshop characteristics, others follow a job shop pattern, and the remaining ones exhibit open shop features.

This paper presents a Genetic Algorithm (AG) to solve the Blocking Job Shop Scheduling Problem when job swapping is allowed, with the objective of minimizing the makespan (Cmax).

The contributions of this study are as follows:

1. A GA is proposed to solve the BJSSP, where the makespan is the objective.
2. The adopted representation of solutions facilitates its incorporation into diverse metaheuristic frameworks.
3. The individuals obtained after the application of genetic operators are always feasible, without requiring a subsequent solution repair process.
4. The proposed GA yielded solutions that outperform some of the best-known Cmax values in the Lawrence benchmark instances for the BJSSP.

The structure of the paper is as follows: Section 2 presents the literature review about the problem; Section 3 provides the problem description and mathematical model; Section 3.1 describes the GA methodology; Section 4 presents the results from the design of experiments and the GA; Section 5 discusses the problem and the proposed methodology; and Section 6 concludes the paper and outlines future research directions.

2. Literature Review

Early attempts to model the BJSSP relied on graphical representations and exact mathematical formulations. The use of disjunctive and alternative graph models enabled an explicit representation of blocking and no-wait constraints [14,15]. Moreover, pioneer studies addressed practical variants in industries such as continuous steel casting [6,16] and multipurpose batch-processing plants [7], extending the scope of the BJSSP beyond discrete manufacturing.

Mixed-Integer Linear Programming (MILP) formulations and branch-and-bound algorithms provided exact solutions for small-scale instances [17]. However, their limited scalability motivated the development of heuristic approaches based on priority rules [18] and local improvement strategies such as feasible insertions [19] or heuristics that focus on disjunctive graphs [20].

The core of recent research has focused on metaheuristic algorithms. Approaches explored include Tabu Search [21], parallel genetic algorithms [17], simulated annealing, and nature-inspired methods such as Coral Reef Optimization [22] and improved

Harmony Search [23]. Notably, ref. [24] proposed efficient primal heuristic updates for blocking instances.

Memetic algorithms [25] and hybrid variants of Differential Evolution [26] have also demonstrated significant progress by balancing exploration and exploitation. More recent studies incorporate machine learning and reinforcement learning to guide heuristics, including the use of Graph Neural Networks for dispatching rule generation [27] and learning-based rolling-horizon algorithms [28].

The BJSSP has been addressed in the context of several relevant variants: 1. Flexible and distributed environments, with transport by cranes or automated guided vehicles [4,29]. 2. Scheduling with multiple and flexible resources [9,30]. 3. Applications in railway traffic, modeling blocking constraints, and efficient deadlock detection [31,32]. 4. Multi-objective optimization, incorporating energy and sustainability criteria in assembly job shop scheduling [33].

This study presents a Genetic Algorithm (GA) that consistently generates feasible solutions for the BJSSP, representing a clear methodological advancement over previous approaches. In contrast, studies such as [20,34] rely on Tabu Search and local search heuristics for neighborhood generation, followed by feasibility recovery procedures. The approach proposed in [31] employs a constructive method based on disjunctive graphs, in which solution feasibility is explicitly evaluated and infeasible solutions are discarded.

Similarly, ref. [21] introduces a heuristic repair mechanism for infeasible solutions using disjunctive graph representations, while ref. [2] proposes a feasibility validation heuristic followed by subsequent recovery. By contrast, the GA proposed in this work operates entirely within the feasible solution space, thereby eliminating the need for feasibility checks or repair mechanisms during the search process

The recent literature converges into three main directions: 1. Integration of Artificial Intelligence: methods combining traditional metaheuristics with reinforcement learning and distribution-based approaches [29,35]. 2. Hybrid and Parallel Metaheuristics: memetic and swarm-based algorithms implemented in parallel computing environments to solve large-scale instances [9,17]. 3. Extensions to Real and Dynamic Systems: applications in complex industries (steel, energy, transportation) and resilient systems under uncertainty [36].

3. Materials and Methods

The Blocking Job Shop Scheduling Problem (BJSSP) is a variant of the classical Job Shop Scheduling Problem in which n jobs must be processed on m machines. Each job is composed of a sequence of operations assigned to specific machines, and the sequence may differ across jobs. Let O represent the set of all operations. For any operation $\in O$, let $\alpha(o)$ denote its job predecessor operation and $\gamma(o)$ denote its job successor operation. The processing time of each operation is denoted p(o) and assumed to be constant and known in advance. Once an operation starts on a machine, it must be completed without interruption, and each machine can handle only one job at a time. Unlike the conventional job shop problem, the BJSSP does not allow intermediate buffering between machines; consequently, a job remains on its current machine until the subsequent machine becomes available. If we let t_0 denote the start time of o and $c_{j,m}$ denote the completion time of job j on machine m, then the minimization formulation for the BJSSP is shown below

$$Min\ \mathrm{C_{max}} = \max\{c_{j,m}\}, j \in \{1, \ldots, n\} \tag{1}$$

Subject to

$$t_{\gamma(o)} \geq t_o + \mathrm{p(o)} \forall o \in \mathrm{O} \tag{2}$$

$$\left(t_u \geq t_{\gamma(v)}\right) \bigvee \left(t_v \geq t_{\gamma(u)}\right)$$

$$\forall u, v \in \mathrm{O} \text{ such that } M(u) = M(v) \text{ and } u \neq v \tag{3}$$

Equation (1) represents the objective function. Constraint (2) indicates that each job can only be processed on one machine at a time. Finally, constraint (3) defines the blocking restriction with which a job remains on a machine until the next machine becomes available.

3.1. Genetic Algorithm

This section presents a GA for solving the BJSSP. Inspired by biological evolution, GA uses natural selection principles—survival of the fittest—to iteratively generate a population of new individuals (chromosomes), each representing a solution for a scheduling problem. In the proposed GA, all generated schedules are feasible and do not need an additional reconstruction phase. The original idea was developed by Holland [37]. The initialization of the Genetic Algorithm (GA) involves both the creation of the initial population and the specification of its parameters.

3.1.1. Initial Population and Parameters

To create the initial population, all individuals are generated randomly according to the encoding scheme introduced in the next section. The parameters of the GA include the population size (N), the crossover rate (Cr), the selection rate (Sr), the mutation rate (Mr), and the stopping criteria.

3.1.2. Encoding Scheme

Encoding defines how a problem solution is represented within the Genetic Algorithm. The encoding of individuals plays a critical role in the GA, as it directly influences the search process and the quality of the solutions obtained. In the proposed structure, a feasible solution is represented by two lists for each machine. The first list contains a permutation of the jobs that start processing on that machine, while the second list consists of a binary sequence of length n in which a value of 0 indicates that the first job operation in the permutation is not yet scheduled, whereas a value of 1 indicates that it is scheduled. In the representation, the binary sequence contains at least as many entries equal to 1 as the number of jobs assigned to start on that machine.

For example, consider the problem of scheduling 6 jobs on 3 machines with the routes and processing times shown in Table 1.

Table 1. Routes and processing times for 6 jobs.

Job	Route	Processing Times
1	3-1-2	4-1-6
2	1-2-3	7-1-4
3	2-1-3	3-1-4
4	2-3-1	5-1-2
5	1-3-2	7-2-5
6	2-1-3	1-3-5

The representation of a solution encoding for this problem is shown in Figure 1.

Thus, jobs 2 and 5 have their first operation on machine 1, jobs 3, 4, and 6 start on machine 2, and job 1 begins on machine 3.

The job permutation for each machine is followed by the binary sequence introduced before Table 1. Thus, when machine 1 becomes available, the first position of the job permutation and the corresponding position in the binary sequence are examined. If the binary value is equal to 1, the first operation of job 5 is scheduled, the job is removed from the permutation, and the position in the binary sequence is updated accordingly. If the

binary value is 0, the machine remains idle, and the binary sequence index is advanced to the next position on the right.

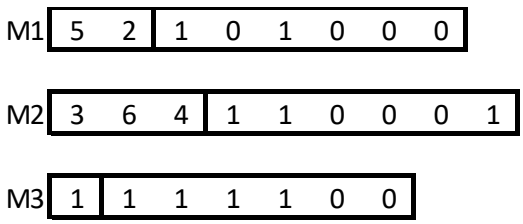


Figure 1. Representation of a solution.

3.1.3. Decoding Scheme

The procedure by which a chromosome is decoded into a production schedule for the BJJSP is described as follows:

AT_i: Maximum completion time on machine i. It is updated as the schedule is built.

CP_i : Current position of the binary sequence of machine i.

$State_i$: state of machine i. Possible values: "Free," "Blocked," "Scheduled".

Active job: A job for which at least its first operation has been already scheduled, and the last one has not.

tprog: Earliest starting time of each stage.

Mtprog: Set of machine(s) i for which $AT_i = tprog$.

$Mtprog^C$: Complement set of *Mtprog*.

AT: Set of completion times AT_i of all machines.

CP: Set of current positions CP_i of all machines.

JSM_i: Permutation of jobs whose initial operation is processed on machine i.

$VectorFP_i$: Binary sequence of length n for machine i.

Inputs:

Jobs J{1, 2, . . .}, Machines M{1, 2, . . .}, Operations O{$o_{1,1}, o_{1,2}$, . . .}, where $o_{j,k}$, is the k-th operation of job j,

Routes R{$m_{1,1}, m_{2,2}$, . . .}, where $m_{j,k}$, is the machine on which the operation k of job j is performed,

Processing time t{$t_{1,1}, t_{2,2}$, . . .}, where $t_{j,k}$ is the processing time of the k-th operation of job j,

Chromosome $C\{\{JSM_1, VectorFP_1\}, \{JSM_2, VectorFP_2\}, \ldots\}$, where each pair $\{\{JSM_i, VectorFP_i\}$ corresponds to machine i.

Output:

Production schedule $S\{(o_{j,k}, m_{j,k}, t_ini, t_fin), \ldots\}$ where t_ini and t_fin are the starting time and the completion time of each operation k, respectively. $o_{j,k}$.

3.1.4. Pseudo-Code

Table 2 presents the general pseudo-code of the decoding scheme:

Table 2. Main steps of the decoding scheme.

1	Initialize variables
2	Make the first scheduling decision for each machine and update variables
3	While unscheduled operations exist, do
4	Process machines with completion time distinct from *tprog*
5	Process machines with completion time equal to *tprog*

In line 1, all scheduling variables and sets are initialized. During line 2, it is decided for each machine i if the first job of JSM_i is scheduled or not according to the binary sequence $VectorFP_i$, that is called a Scheduling decision. Then all variables and sets are updated. Next, while there are unscheduled operations, lines 4 and 5 are performed.

Line 4 handles the set of machines where the maximum completion time differs from *tprog*. If there is a machine idle and ready to process an active job (a job whose processing has already started) at *tprog*, the next operation of that job is scheduled, and that machine is set free. Otherwise, if there are jobs that have not yet begun processing, a Scheduling decision is made.

Line 5 addresses those machines having a maximum completion time equal to *tprog*. If a machine i is neither blocked nor scheduled at *tprog*, and it has schedulable active jobs, the job with the lowest index is scheduled, and the machine is set free. If there are non-schedulable jobs on i, the machine is set free. If $|\text{Mtprog}| > 1$, that is, there is more than one machine ready to be scheduled, the algorithm checks if a job swap is possible, among machines in Mtprog. If so, it performs the swap and sets free the implied machines; otherwise, the machines in Mtprog are blocked.

Next, Figures 2–5 show the flow diagrams corresponding to lines 1, 2, 4, and 5 of the pseudo-code, whereas Table 3 describes the subprograms used within them. The subprograms used are explained in Table 2.

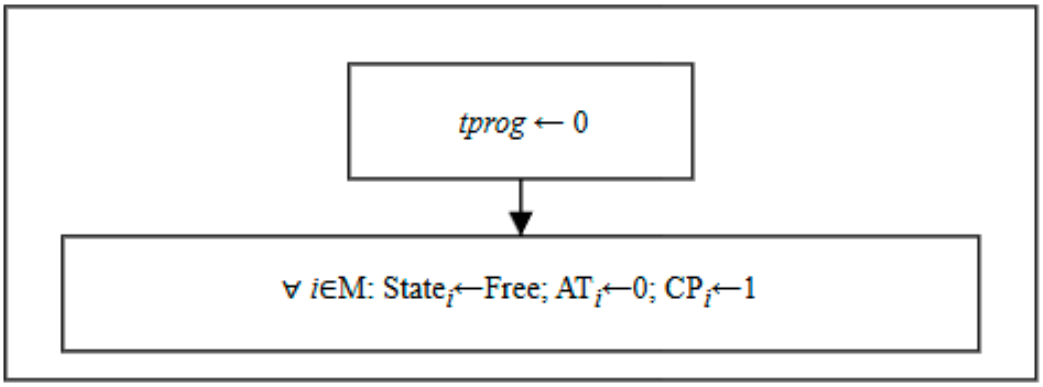


Figure 2. Initialize variables.

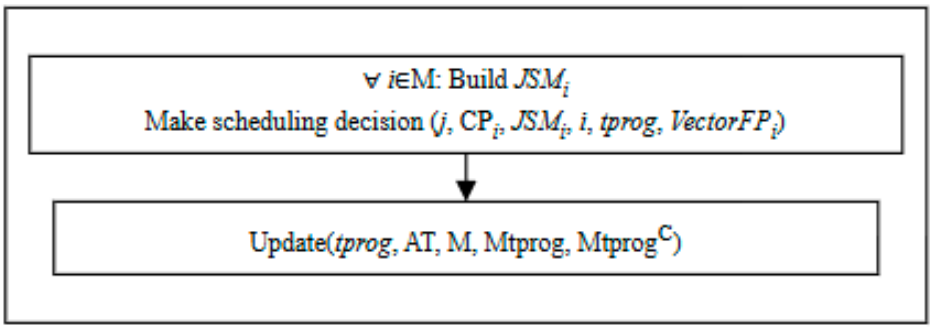


Figure 3. Make the first scheduling decision for each machine and update variables.

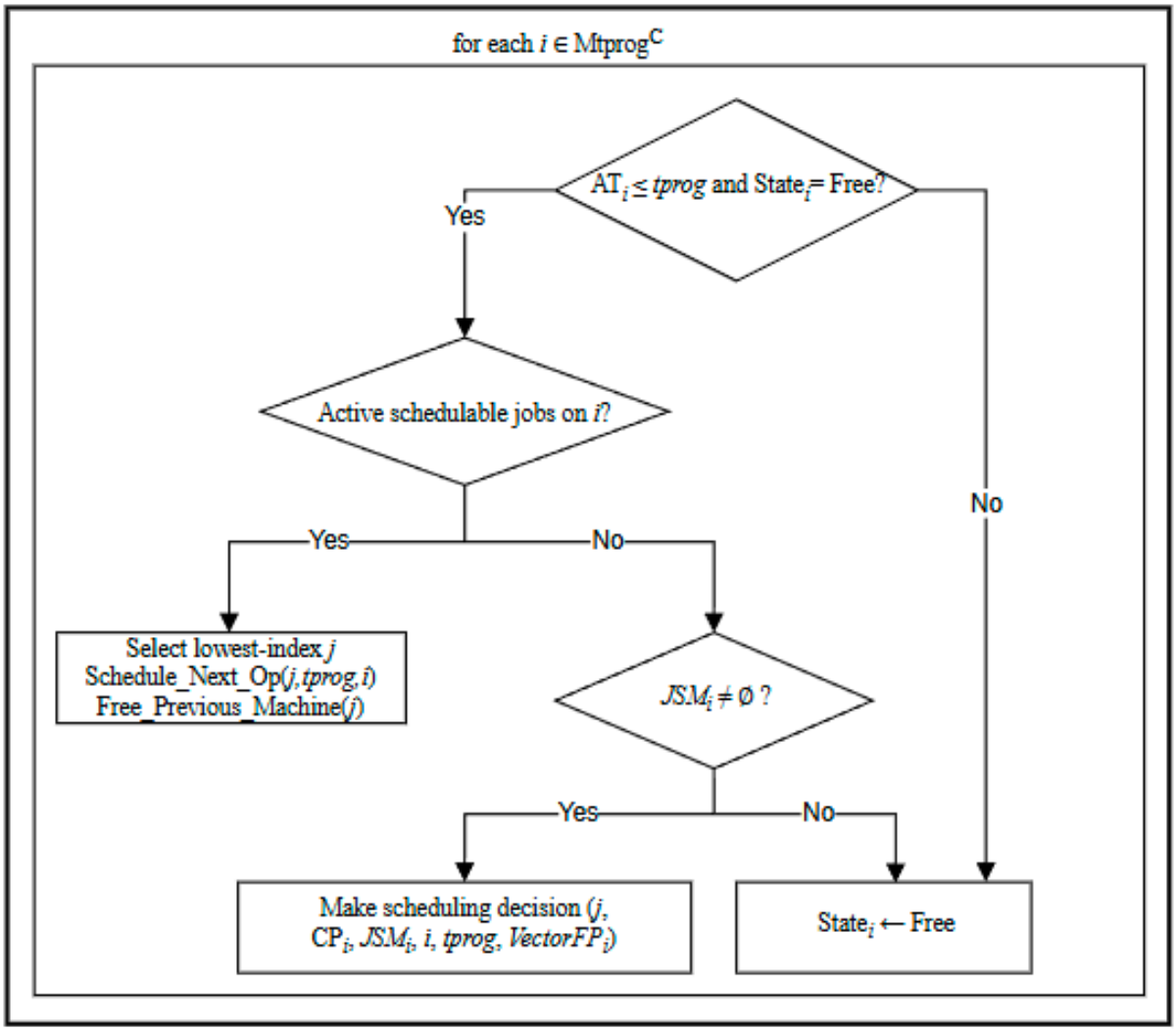


Figure 4. Process machines not ready for immediate scheduling.

Table 3. Subprograms of the pseudo-code.

Subprograms	Description
Schedule the next operation (j, $tprog$, i)	Schedules operation $o_{j,k}$ on machine $i = m_{j,k}$, starting at $tprog$ and ending at $C_j = tprog + t_{j,k}$. Updates $State_i$, AT_i, and adds to set S.
Free_Previous_Machine (j)	Sets $State_i \leftarrow$ "Free" to release the previous machine used by job j.
Make scheduling decision (j, $VectorFP_i$, JSM_i, i, $tprog$)	If $VectorFP_i$ $(CP_i) = 1$, call Schedule_Next_Op, remove j from JSM_i Else, set $State_i \leftarrow$ "Free", set $CP_i = CP_i + 1$
Update $\left(tprog, AT, M, Mtprog, Mtprog^C\right)$	Sets $tprog = \text{Min}\{AT_i\}$

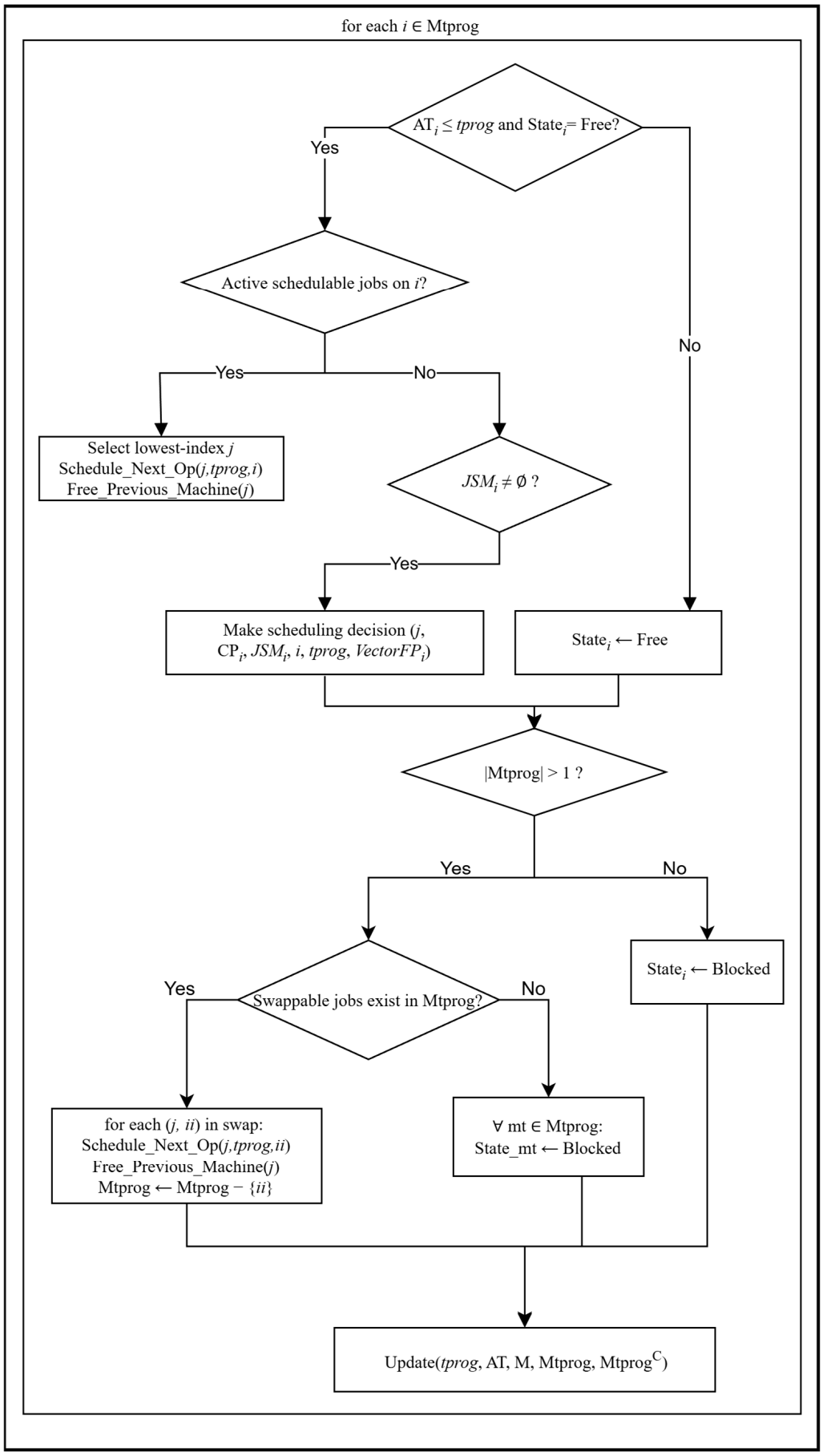


Figure 5. Process machines ready to be scheduled.

3.1.5. Selection of Individuals

Once all the solutions in a generation have been created, their performance is evaluated. Subsequently, they are sorted in ascending order, given that the objective function (Cmax) is to be minimized, and the *Sr* best solutions are selected to proceed to the next generation.

3.1.6. Crossover Operator

The crossover operator aims to combine characteristics of parent individuals to generate *Cr* new offspring by using characteristics of known good solutions to explore new promising areas of the solution space. The proposed crossover operator is defined as follows:

Step 1. Randomly select two parent solutions from the *Sr* best solutions.

Step 2. For each machine, determine the structure of the offspring by applying a binary tournament between the selected parent solutions.

3.1.7. Mutation Operator

In this study, the mutation is performed by an immigration strategy where randomly generated new solutions complete the population of individuals in each new generation, to maintain the population diversity [38]. Consequently, the population of size N of the next generation is composed of three groups: $Sr \times N$ individuals preserved through the selection operator, $Cr \times N$ individuals created by the crossover operator, and $(1 - Sr - Cr) \times N$ individuals introduced by the mutation operator.

4. Results

A 2^k factorial design with replications was proposed to evaluate the effects of the GA parameters on the makespan quality. Table 4 presents the factors considered and their corresponding levels.

Table 4. Factors and levels.

Factor-Notation	Levels
Initial population-A	5000–15,000
Selection rate-B	5–20%
Crossover rate-C	50–70%
Generations-D	30–60

The levels of factors A and D were determined based on the available computational capacity to ensure reasonable execution times, whereas the levels of factors B and C were defined according to the recommendations provided in [39,40].

The experimental runs were conducted on an instance with 20 jobs and five machines. The processing times were generated according to a uniform distribution between 1 and 99. The processing routes were generated randomly, with all jobs visiting each machine once.

Figure 6 presents the results from the experimental runs considering the response variable and the significant factors: initial population and selection rate have a significant effect on the Cmax value found by the GA.

Figure 7 shows the graphs of the main effects for each factor and its levels.

Consequently, the algorithm was parameterized with a population size of 15,000 individuals, a selection rate of 20%, a crossover rate of 70%, and 30 generations as the stopping criterion.

The proposed algorithm was implemented in Java. All experiments were conducted on a computer equipped with an AMD Ryzen 3 processor at 2.70 GHz and 16 GB of memory.

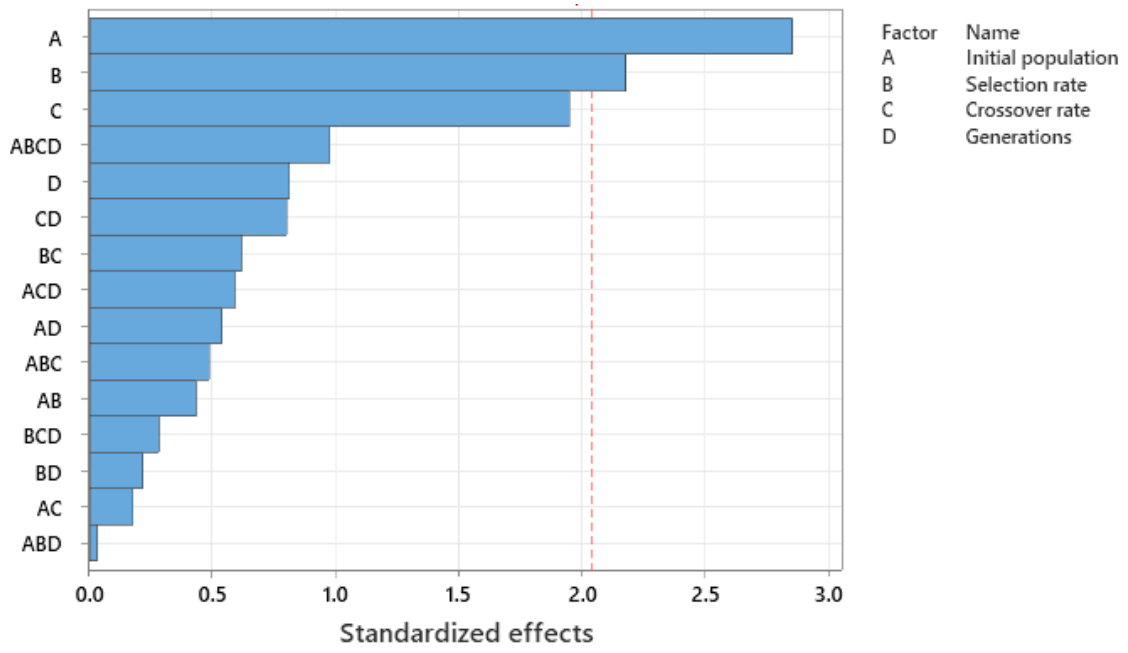


Figure 6. Pareto chart of standardized effects.

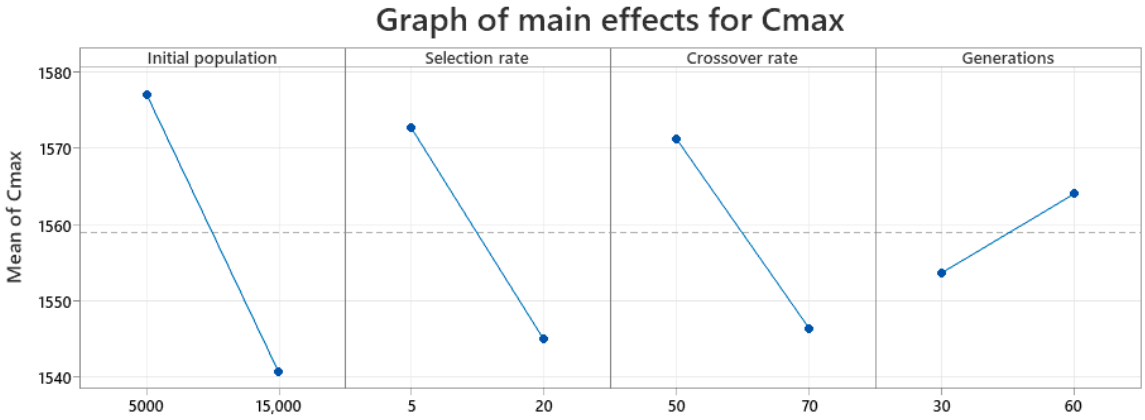


Figure 7. Graph of main effects for Cmax.

The performance of the proposed algorithm was evaluated on the job shop scheduling Lawrence instances, which include 40 problems with sizes of 5, 10, and 15 machines and 10, 15, 20, and 30 jobs. Table 5 summarizes the results obtained for each instance, including the Average Relative Percentage Deviation (ARPD), which is calculated as $\frac{Best\ GA - Best\ O}{Best\ O} \times 100\%$, compared to the best Cmax reported by [14,15,21,23,41,42]. The ARPD measures the difference between the best result in Cmax obtained by the GA (Best GA) and the best result reported by the references (Best O), divided by the reference result, with negative values indicating that the proposed GA achieved a better performance. The column header GA refers to the best Cmax obtained by the proposed algorithm in the current article.

The ARPD results presented in bold indicate instances where the proposed GA found better solutions than those reported by the authors, while results not reported by the authors were denoted as NR.

Table 5. Results in Lawrence instances.

Instance	n × m	GA	[40]	[23]	[14,41]	[42]	[15]	[21]
La01	10 × 5	852	**−5.33%**	7.44%	**−25.07%**	3.90%	7.44%	7.44%
La02	10 × 5	863	1.05%	8.83%	**−18.20%**	5.63%	5.89%	8.83%
La03	10 × 5	812	1.63%	13.57%	**−25.37%**	9.73%	2.78%	13.57%
La04	10 × 5	767	**−5.54%**	3.23%	**−21.01%**	0.39%	**−2.17%**	3.23%
La05	10 × 5	714	**−0.14%**	7.53%	**−36.02%**	7.21%	7.53%	7.53%
La06	15 × 5	1128	**−6.47%**	6.02%	**−24.45%**	**−4.41%**	**−0.27%**	4.83%
La07	15 × 5	1141	**−3.55%**	9.92%	**−34.31%**	5.26%	3.16%	10.88%
La08	15 × 5	1184	**−1.25%**	11.49%	**−24.54%**	5.24%	4.87%	11.70%
La09	15 × 5	1256	**−1.80%**	5.99%	**−35.36%**	2.70%	**−0.87%**	5.72%
La10	15 × 5	1087	**−12.90%**	**−2.07%**	**−29.09%**	**−9.64%**	**−6.93%**	**−2.07%**
La11	20 × 5	1584	**−7.04%**	8.05%	**−27.64%**	0.00%	4.21%	7.39%
La12	20 × 5	1250	**−15.94%**	**−1.73%**	**−41.09%**	**−10.14%**	**−4.43%**	**−2.04%**
La13	20 × 5	1524	**−5.93%**	4.03%	**−33.62%**	**−1.10%**	**−0.26%**	4.67%
La14	20 × 5	1538	**−9.37%**	2.12%	**−36.52%**	**−5.06%**	2.12%	4.48%
La15	20 × 5	1583	**−11.02%**	4.35%	**−33.23%**	**−2.88%**	0.76%	6.24%
La16	10 × 10	1389	13.48%	31.04%	25.25%	21.63%	20.78%	31.04%
La17	10 × 10	1269	19.60%	36.60%	**−26.18%**	29.89%	27.41%	36.45%
La18	10 × 10	1341	15.90%	30.83%	-	24.40%	18.15%	30.83%
La19	10 × 10	1579	32.24%	51.39%	**−11.59%**	44.46%	42.51%	49.95%
La20	10 × 10	1461	21.35%	37.83%	**−10.04%**	26.60%	30.56%	37.83%
La21	15 × 10	1807	**−0.22%**	18.80%	**−33.10%**	16.96%	14.44%	23.18%
La22	15 × 10	1676	1.95%	21.54%	**−34.68%**	14.95%	21.54%	24.42%
La23	15 × 10	1612	**−12.01%**	7.68%	**−47.04%**	2.68%	7.68%	11.79%
La24	15 × 10	1807	4.57%	20.63%	**−46.06%**	16.88%	18.65%	29.26%
La25	15 × 10	1774	4.23%	24.58%	**−19.76%**	18.35%	13.65%	29.21%
La26	20 × 10	2213	**−3.82%**	8.75%	**−45.00%**	4.14%	8.75%	14.72%
La27	20 × 10	2801	11.46%	33.13%	-	28.78%	29.98%	42.91%
La28	20 × 10	2379	**−0.83%**	17.37%	**−22.51%**	14.87%	15.37%	26.54%
La29	20 × 10	2192	0.00%	15.49%	**−42.19%**	10.15%	15.49%	21.58%
La30	20 × 10	2281	**−8.02%**	8.88%	**−28.11%**	8.77%	6.24%	16.08%
La31	30 × 10	3498	3.06%	19.75%	-	11.51%	19.75%	28.84%
La32	30 × 10	3693	**−5.36%**	14.09%	-	11.37%	14.09%	23.64%
La33	30 × 10	3406	0.38%	19.76%	-	11.27%	19.76%	27.47%
La34	30 × 10	3474	**−5.75%**	21.98%	**−23.83%**	10.43%	21.98%	27.30%
La35	30 × 10	3556	0.17%	21.66%	-	12.14%	21.66%	28.10%
La36	15 × 15	2323	14.21%	29.56%	**−31.05%**	21.05%	19.01%	35.61%
La37	15 × 15	2501	11.45%	28.13%	**−16.88%**	23.26%	28.13%	38.79%
La38	15 × 15	2331	16.90%	36.48%	**−26.86%**	27.52%	23.99%	43.01%
La39	15 × 15	2598	20.72%	45.71%	**−31.40%**	38.04%	43.30%	53.09%
La40	15 × 15	2284	9.70%	28.53%	**−31.72%**	18.65%	18.46%	34.99%

5. Discussion

The experimental results demonstrate that the proposed Genetic Algorithm (GA) exhibits strong optimization performance in solving the BJSSP. As shown in Table 2, the GA achieves the new lowest Cmax values in two instances compared to the best-known results reported in the literature.

When compared with previous studies, the GA outperforms the results reported by [40] in 20 out of 40 benchmark instances. On the other hand, the results reported by [15,42] are outperformed in six of the 40 instances. Additionally, the results obtained by [21,23] are surpassed in two of the 40 instances.

The execution time of the proposed Genetic Algorithm (GA) for the problem sets used in the experiments ranged from 20 s for instances with 10 jobs and five machines to 600 s

for those with 30 jobs and 10 machines. These times are substantially lower than those reported in previous studies, which range from 7200 s in [23] to approximately 3 h in [43].

The processor used to run the proposed Genetic Algorithm provides higher computational power and performance compared to the Pentium IV processors reported in [23,38], the Pentium II processor used in [40], the Intel Core Duo in [39], and the AMD Phenom II X4 employed in [15]. This difference in hardware capabilities partially explains the observed discrepancies in execution times. Therefore, the reported computational times should be interpreted as indicative rather than strictly comparable, given the differences in experimental platforms and computational environments.

The ARPD values obtained for the 10×10 instances exhibit inferior performance compared to those reported in previous studies. This behavior can be attributed to the fact that the number of feasible solutions available at the initial stage of the algorithm depends on the distribution of jobs according to their first operation across machines. Specifically, when each job starts on a different machine, the size of the initial solution space is significantly reduced, limiting the algorithm's exploratory capacity, and consequently affecting the quality of the solutions obtained.

The ARPD values obtained for large-scale instances, such as 20×10 and 30×10, are higher when compared to those reported in previous studies. This behavior is mainly attributed to computational limitations that restrict the algorithm's ability to thoroughly explore large populations of individuals within competitive computational times.

The proposed approach has relevant applications in operations scheduling and decision-making across various domains, including manufacturing, health services, and warehousing operations, and scenarios involving shared resources such as automated guided vehicles (AGVs), cranes, and forklifts.

6. Conclusions

This study proposes a Genetic Algorithm for solving the Job Shop Scheduling Problem with blocking constraints, aimed at minimizing the makespan. A procedure was proposed to consistently generate feasible solutions throughout the search process without the need for a reconstruction procedure.

This article proposes a machine-based solution representation that considers the first operation of each job and employs a binary encoding to determine whether an operation should be scheduled at a given time. The correspondent decoding procedure can be applied in other metaheuristics approaches, such as Ant Colony Optimization, Particle Swarm Optimization, and Tabu Search.

The results obtained for large-scale instances indicate opportunities for improvement using higher-capacity, higher-performance processors, which would enable the exploration of a larger number of solutions within reasonable computational times.

As future work, the incorporation of local search heuristics for generating initial solutions could be considered, with the aim of enhancing the exploration and exploitation performance of the Genetic Algorithm.

The solution representation depends on the distribution of jobs according to their initial machine assignments, which reduces the size of the search space when the job permutations associated with each machine are small.

Although the upper levels of the selection and crossover rates were established according to the literature, future work will consider adjusting these limits to further evaluate their impact on the performance of the GA.

Future research will investigate intermediate levels of the GA parameters to analyze the nature of their dependence on Cmax.

On the other hand, future research could incorporate additional constraints, such as job recirculation and sequence-dependent setup times. Moreover, the proposed methodology could be extended to address alternative objectives, including the minimization of tardiness, production costs, and energy consumption.

Author Contributions: Writing, J.V. and E.R.-V.; review, J.V. and E.R.-V.; editing, J.V. and E.R.-V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding. The APC was partially funded by Corporación Universitaria Remington.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding authors.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviation

AGV	Automated Guided Vehicle
ARPD	Average Relative Percentage Deviation
BJSSP	Blocking Job Shop Scheduling Problem
Cmax	Maximum Completion Time
GA	Genetic Algorithm
NP-hard	Non-deterministic Polynomial Difficult Problem.
PSO	Particle Swarm Optimization

References

1. Louaqad, S.; Kamach, O.; Iguider, A. Scheduling for Job Shop Problems with Transportation and Blocking No-Wait Constraints. *J. Theor. Appl. Inf. Technol.* **2018**, *96*, 2782–7282.
2. Lange, J.; Werner, F. A Permutation-Based Neighborhood for the Blocking Job-Shop Problem with Total Tardiness Minimization. In *Operations Research Proceedings 2017*; Springer: Cham, Switzerland, 2018; pp. 581–586. [CrossRef]
3. Meloni, C.; Pranzo, M.; Samà, M. Evaluation of VaR and CVaR for the makespan in interval valued blocking job shops. *Int. J. Prod. Econ.* **2022**, *247*, 108455. [CrossRef]
4. Zeng, C.; Tang, J.; Yan, C. Scheduling of no buffer job shop cells with blocking constraints and automated guided vehicles. *Appl. Soft Comput. J.* **2014**, *24*, 1033–1046. [CrossRef]
5. Hall, N.G.; Sriskandarajah, C. A survey of machine scheduling problems with blocking and no-wait in process. *Oper. Res.* **1996**, *44*, 510–525. [CrossRef]
6. Meloni, C.; Pacciarelli, D.; Pranzo, M. A Rollout Metaheuristic for Job Shop Scheduling Problems. *Ann. Oper. Res.* **2004**, *131*, 215–235. [CrossRef]
7. Romero, J.; Puigjaner, L.; Holczinger, T.; Friedler, F. Scheduling Intermediate Storage Multipurpose Batch Plants Using the S-Graph. *AIChE J.* **2004**, *50*, 403–417. [CrossRef]
8. Wójcik, R.; Pempera, J.; Bocewicz, G.; Banaszak, Z. The no-wait cyclic scheduling of delivery traffic in the grid distribution network. *ISA Trans.* **2022**, *131*, 282–298. [CrossRef] [PubMed]
9. Liu, S.Q.; Kozan, E. A hybrid metaheuristic algorithm to optimise a real-world robotic cell. *Comput. Oper. Res.* **2017**, *84*, 188–194. [CrossRef]
10. Mejía, G.; Montoya, C. Scheduling manufacturing systems with blocking: A Petri net approach. *Int. J. Prod. Res.* **2009**, *47*, 6261–6277. [CrossRef]
11. Masoud, M.; Kozan, E.; Kent, G. Hybrid metaheuristic techniques for optimising sugarcane rail operations. *Int. J. Prod. Res.* **2015**, *53*, 2569–2589. [CrossRef]
12. Mati, Y.; Lahlou, C.; Dauzère-Pérès, S. Modelling and solving a practical flexible job-shop scheduling problem with blocking constraints. *Int. J. Prod. Res.* **2011**, *49*, 2169–2182. [CrossRef]

13. Koulamas, C.; Panwalkar, S.S. The two-stage no-wait/blocking proportionate super shop scheduling problem. *Int. J. Prod. Res.* **2019**, *57*, 2956–2965. [CrossRef]
14. Mascis, A.; Pacciarelli, D. Discrete Optimization Job-shop scheduling with blocking and no-wait constraints. *Eur. J. Oper. Res.* **2002**, *143*, 498–517. [CrossRef]
15. Oddi, A.; Rasconi, R.; Cesta, A.; Smith, S.F. Iterative Improvement Algorithms for the Blocking Job Shop. In *Proceedings of the International Conference on Automated Planning and Scheduling*; Association for the Advancement of Artificial Intelligence: Washington, DC, USA, 2012.
16. De Moerloose, P.; Maenhout, B. A two-stage local search heuristic for solving the steelmaking continuous casting scheduling problem with dual shared-resource and blocking constraints. *Oper. Res.* **2023**, *23*, 2. [CrossRef]
17. Aitzai, A.; Benmedjdoub, B.; Boudhar, M. A branch and bound and parallel genetic algorithm for the job shop scheduling problem with blocking. *Int. J. Oper. Res.* **2012**, *14*, 343–365. [CrossRef]
18. Lakehal, M.; Aissani, A.; Bouibede, K. New algorithm of scheduling based on prioriry rules on machines to solve bi-criteria blocking job shop scheduling problem. In Proceedings of the 12th International Symposium on Programming and Systems, ISPS 2015, Algiers, Algeria, 28–30 April 2015; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2015; pp. 149–156. [CrossRef]
19. Gröflin, H.; Klinkert, A. Feasible insertions in job shop scheduling, short cycles and stable sets. *Eur. J. Oper. Res.* **2006**, *177*, 763–785. [CrossRef]
20. Brucker, P.; Kampmeyer, T. Cyclic job shop scheduling problems with blocking. *Ann. Oper. Res.* **2008**, *159*, 161–181. [CrossRef]
21. Dabah, A.; Bendjoudi, A.; Aitzai, A. An efficient tabu search neighborhood based on reconstruction strategy to solve the blocking job shop scheduling problem. *J. Ind. Manag. Optim.* **2017**, *13*, 2015–2031. [CrossRef]
22. Shieh, C.S.; Nguyen, T.T.; Lin, W.W.; Nguyen, D.C.; Horng, M.F. Modified Coral Reef Optimization Methods for Job Shop Scheduling Problems. *Appl. Sci.* **2022**, *12*, 9867. [CrossRef]
23. Hou, P.; Wang, D.; Li, X. An improved harmony search algorithm for blocking job shop to minimize makespan. In Proceedings of the 16th International Conference on Computer Supported Cooperative Work in Design, Wuhan, China, 23–25 May 2012.
24. Mogali, J.K.; Barbulescu, L.; Smith, S.F. Efficient primal heuristic updates for the blocking job shop problem. *Eur. J. Oper. Res.* **2021**, *295*, 82–101. [CrossRef]
25. Freud, D.; Elalouf, A. A Memetic Algorithm Approach for the Job-Shop Scheduling Problem with Variable Machine Efficiency and Maintenance Activities. *Appl. Sci.* **2025**, *15*, 1431. [CrossRef]
26. Chen, X.; Du, W.; Qian, F. Multi-objective differential evolution with ranking-based mutation operator and its application in chemical process optimization. *Chemom. Intell. Lab. Syst.* **2014**, *136*, 85–96. [CrossRef]
27. Wong, V.W.H.; Kim, S.H.; Park, J.; Park, J.; Law, K.H. Generating Dispatching Rules for the Interrupting Swap-Allowed Blocking Job Shop Problem Using Graph Neural Network and Reinforcement Learning. *ASME J. Manuf. Sci. Eng.* **2024**, *146*, 011009. [CrossRef]
28. Zeiträg, Y.; Figueira, J.R. A novel machine-learning rolling horizon heuristic for dynamic lot-sizing and job shop scheduling problems. *Int. J. Prod. Res.* **2025**, *63*, 4563–4589. [CrossRef]
29. Du, Y.; Li, J.; Li, C.; Duan, P. A Reinforcement Learning Approach for Flexible Job Shop Scheduling Problem With Crane Transportation and Setup Times. *IEEE Trans. Neural Netw. Learn. Syst.* **2024**, *35*, 5695–5709. [CrossRef]
30. Mati, Y.; Xie, X. Multiresource shop scheduling with resource flexibility and blocking. *IEEE Trans. Autom. Sci. Eng.* **2011**, *8*, 175–189. [CrossRef]
31. Liu, S.Q.; Kozan, E. Scheduling trains as a blocking parallel-machine job shop scheduling problem. *Comput. Oper. Res.* **2009**, *36*, 2840–2852. [CrossRef]
32. Doshi, H.; Tripathi, A.; Agarwal, K.; Khadilkar, H.; Kalyanakrishnan, S. Linear-Time Optimal Deadlock Detection for Efficient Scheduling in Multi-Track Railway Networks. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24), Jeju, Republic of Korea, 3–9 August 2024.
33. Dong, Y.; Liao, W.; Xu, G. Multi-Objective Optimization of Energy-Efficient Multi-Stage, Multi-Level Assembly Job Shop Scheduling. *Appl. Sci.* **2024**, *14*, 8712. [CrossRef]
34. Mascis, A.; Pacciarelli, D. *Machine Scheduling via Alternative Graphs*; Report DIA-46-2000; Dipartimento di Informatica e Automazione, Università Roma Tre: Rome, Italy, 2000.
35. Brucker, P.; Heitmann, S.; Hurink, J.; Nieberg, T. Job-shop scheduling with limited capacity buffers. *OR Spectr.* **2005**, *28*, 151–176. [CrossRef]
36. Hu, M.; Zhang, W.; Ren, X.; Qin, S.; Chen, H.; Zhang, J. A novel resilient scheduling method based on multi-agent system for flexible job shops. *Int. J. Prod. Res.* **2025**, 1–21. [CrossRef]
37. Holland, J.H. *Adaptation in Natural and Artificial Systems*; University of Michigan Press: Ann Arbor, MI, USA, 1975.
38. Huang, S.; Tsai, Y.C.; Der Chou, F. A Trajectory-Based Immigration Strategy Genetic Algorithm to Solve a Single-Machine Scheduling Problem with Job Release Times and Flexible Preventive Maintenance. *Algorithms* **2023**, *16*, 207. [CrossRef]

39. Grefenstette, J.J. Optimization of Control Parameters for Genetic Algorithms. *IEEE Trans. Syst. Man Cybern.* **2007**, *16*, 122–128. [CrossRef]
40. Hassanat, A.; Almohammadi, K.; Alkafaween, E.; Abunawas, E.; Hammouri, A.; Prasath, V.B.S. Choosing mutation and crossover ratios for genetic algorithms-a review with a new dynamic approach. *Information* **2019**, *10*, 390. [CrossRef]
41. Pranzo, M.; Pacciarelli, D. An iterated greedy metaheuristic for the blocking job shop scheduling problem. *J. Heuristics* **2016**, *22*, 587–611. [CrossRef]
42. Gröflin, H.; Klinkert, A. A new neighborhood and tabu search for the Blocking Job Shop. *Discrete Appl. Math.* **2009**, *157*, 3643–3655. [CrossRef]
43. Mati, Y.; Rezg, N.; Xie, X. Scheduling Problem of Job-Shop with Blocking: A Taboo Search Approach. In Proceedings of the 4th Metaheuristics International Conference, Porto, Portugal, 16–20 July 2001; pp. 643–648.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.