applied sciences

MDPI

Article

Hybrid Genetic Simulated Annealing Algorithm for Improved Flow Shop Scheduling with Makespan Criterion

Hongjing Wei [1,2], Shaobo Li [3,4,*], Houmin Jiang [5], Jie Hu [6] and Jianjun Hu [3,7,*]

1 Key Laboratory of Advanced Manufacturing Technology of the Ministry of Education, Guizhou University, Guiyang 550025, China; hongjingwei@126.com
2 School of Mechanical Engineering, Guizhou Institute of Technology, Guiyang 550003, China
3 School of Mechanical Engineering, Guizhou University, Guiyang 550025, China
4 Guizhou Provincial Key Laboratory of Public Big Data (Guizhou University), Guiyang 550025, China
5 College of Computer Science and Technology, Guizhou University, Guiyang 550025, China; jhmhehe@gmail.com
6 College of Big Data Statistics, GuiZhou University of Finance and Economics, Guiyang 550025, China; jason.houu@gmail.com
7 Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208, USA
* Correspondence: lishaobo@gzu.edu.cn (S.L.); jianjunh@cse.sc.edu (J.H.)

Received: 11 October 2018; Accepted: 5 December 2018; Published: 14 December 2018

Abstract: Flow shop scheduling problems have a wide range of real-world applications in intelligent manufacturing. Since they are known to be NP-hard for more than two machines, we propose a hybrid genetic simulated annealing (HGSA) algorithm for flow shop scheduling problems. In the HGSA algorithm, in order to obtain high-quality initial solutions, an MME algorithm, combined with the MinMax (MM) and Nawaz–Enscore–Ham (NEH) algorithms, was used to generate the initial population. Meanwhile, a hormone regulation mechanism for a simulated annealing (SA) schedule was introduced as a cooling scheme. Using MME initialization, random crossover and mutation, and the cooling scheme, we improved the algorithm's quality and performance. Extensive experiments have been carried out to verify the effectiveness of the combination approach of MME initialization, random crossover and mutation, and the cooling scheme for SA. The result on the Taillard benchmark showed that our HGSA algorithm achieved better performance relative to the best-known upper bounds on the makespan compared with five state-of-the-art algorithms in the literature. Ultimately, 109 out of 120 problem instances were further improved on makespan criterion.

Keywords: flow shop scheduling; makespan; hybrid algorithm; genetic algorithms; simulated annealing

1. Introduction

The flow shop scheduling problem (FSSP) was first proposed by Johnson in 1954 [1]. Due to its extensive production applications in industries such as food processing, steel manufacturing, plastics production, and chemical manufacturing, the problem has received much attention since it was introduced. Researchers from all over the world have conducted in-depth research on this issue, and in the literature [2,3], they have developed branch-and-bound algorithms to solve the flow shop scheduling problem. However, these exact solution algorithms are only suitable for small-sized scheduling problems, as the calculation time increases exponentially with the problem size. As FSSP has proved to be an NP-hard problem [4], exact algorithms have not been suitable for large-scale scheduling problems. After that, several heuristic algorithms were proposed. Among them,

the Nawaz–Enscore–Ham (NEH) heuristic algorithm [5], proposed by Nawaz et al., was used to solve an FFSP with n workpieces and m machines. The comparison between NEH and 15 other algorithms showed that it can quickly obtain better solutions and is suitable for solving large-scale flow shop scheduling problems. However, the quality of the feasible solutions obtained by the NEH algorithm is not high [6]. After the NEH algorithm was proposed, many researchers combined NEH with other optimization algorithms to achieve better results. Marichelvam et al. [7] combined NEH with the cuckoo search algorithm to form an improved cuckoo search algorithm (ICS), which proved to be more effective at achieving optimal solutions than the other algorithms in the literature. Qi et al. [6] proposed an improved variable neighborhood search algorithm and proved that the NEH heuristic algorithm could generate better initial solutions. Burdett and Kozan [8] addressed the representation and construction of accurate train schedules by a hybrid job shop approach and proposed a constructive algorithm to construct a schedule using NEH insertion, backtracking, and dynamic route selection mechanisms. The experiment results showed the constructive algorithm required less CPU time and obtained better solutions.

Rathinam [9] used the decision tree (DT) algorithm to minimize the makespan of the FSSP. In addition, Govindan et al. [10] provided a hybrid algorithm (ADE) that combined a scatter search algorithm with decision tree which exceeded other existing hybrid algorithms. The main disadvantage is that the tree size of the ADE and DT algorithms increases with the number of jobs and machines in large-scale FSSPs. In [11], three kinds of hybrid discrete artificial bee colony (ABC) algorithm (hDABC1, hDABC2, and hDABC3), with the initial solution generated by MME-A and MME-B and new solutions generated by the adaptation strategy and distribution estimation, were proposed. Pan and Huang [12] developed a hybrid genetic algorithm (GA) based on two different local search strategies (insertion search and the insertion search repair and cut) and an orthogonal-array-based crossover. Gao et al. [13] proposed two Bertollisi heuristics [14], which were based on a job pair comparison approach, to generate a sequence of jobs and constructive heuristics combined with standard deviation heuristics. The algorithms integrated with local search strategies to improve the quality of the solution. The results showed the effectiveness of the proposed algorithm, especially for large-scale FSSPs. Nowicki and Smutnicki [15] proposed an approximation algorithm based on a tabu search technique with a specific neighborhood definition to solve a permutation flow shop problem. Sayoti et al. [16] proposed an adaptation of a new approach called the golden ball algorithm (GBA), which was inspired by concepts and strategies for soccer playing. The result showed GBA was able to find the optimal schedule rapidly for the small-scale flow shop schedule problem. In [17], a genetic algorithm integrated with a Hopfield network was proposed and proved to be an effective method to solve NP-hard problems. Although researchers have developed many algorithms to solve FSSPs, there still exist some shortcomings in these algorithms, such as local optimization and high computational cost, especially when solving large-scale problems. Table 1 summarizes some of the algorithms proposed for FSSPs in the literature. Most of these studies demonstrate that combinational heuristic and metaheuristic approaches which include advantages of more than one algorithm are very useful for solving flow shop problems. For instance, Tseng et al. [18] studied a hybrid genetic algorithm for a no-wait flow shop problem. Ding et al. [19] presented an improved iterated greedy algorithm with a tabu-based reconstruction strategy. An iterated greedy algorithm in [20], a high-performing memetic algorithm in [21], and a discrete self-organizing migrating algorithm in [22] have been used for flow shop problems with the minimum makespan criterion and obtained better results. Eddaly et.al [23] proposed a hybrid combinatorial particle swarm optimization (HCPSO) algorithm as a resolution technique for solving this problem. Burdett and Kozans [24] proposed a new multiparent operator which could greatly improve the performance of a GA search.

The GA and the simulated annealing (SA) algorithm are well-known metaheuristics that have been successfully used in flow shop problems. The genetic algorithm has the advantages of strong global optimization ability, fast speed, strong versatility, and easy implementation. However, it has the drawback of poor local search ability, which decreases search efficiency, especially in the late period of

optimization. On the other hand, the simulated annealing algorithm has strong local search capabilities to make up the shortcomings of the genetic algorithm. GA + SA has been used to solve job shop [25], open shop [26], and flexible flow shop problems [27]. However, to the best of our knowledge, there are no reports on its application in flow shop scheduling, especially with improved crossover and mutation operators and adaptive simulated annealing.

Therefore, we propose a hybrid genetic simulated annealing (HGSA) algorithm to solve flow shop scheduling problems. The main contributions of this paper include the following:

(1) We propose the novel HGSA algorithm to solve FSSPs. It is characterized by an operational coding in GA and a hormone regulation mechanism in the simulated annealing part of the algorithm. The MME [28] algorithm, combined with the NEH [5] and MinMax (MM) [29] heuristic algorithms, is used for initialization of the population. We use location-based intersection and two-point intersection for crossover operations with either one of these two being randomly selected. In the mutation process, twors mutation or inversion mutation [30] is randomly employed to mutate the population. After the crossover and mutation operation is completed, the best individuals are retained, and the simulated annealing operation is performed on these solutions.

(2) Using the widely adopted Taillard benchmark FSSPs, we conducted extensive experiments and showed that our HGSA algorithm achieved better results than the baseline algorithms to which it was compared in our study. We showed that our HGSA algorithm's high performance for FSSPs is based on its hybrid search strategy, twors/inversion mutation, location-based/two-point crossovers, and their combination with the MME heuristic algorithm for population initialization.

The rest of this paper is organized as follows: Section 2 introduces the definition of the problem and mathematical model of the flow shop scheduling problem, Section 3 describes the HGSA, Section 4 shows the experimental results of HGSA, and Section 5 provides conclusions and future works.

Table 1. Survey of algorithms for solving the flow shop scheduling problem (FSSP).

Author, Year, and Reference	Algorithm	Optimization Criteria	Benchmarks	Summary
Koulamas (2001) [31]	Heuristics with ratio performance guarantees (HRPG)	Makespan		- Analyzed the worst-case absolute bound for a heuristic based on compact vector summation techniques - Pointed out that the heuristic becomes asymptotically optimal with a large number of jobs
Chang et al. (2002) [32]	Gradual-priority weighting (GPW)	Makespan, total flowtime, total tardiness		- GPW approach is proposed to search the Pareto optimal solution
Andreas et al. (2003) [33]	Metaheuristics	Total flowtime	Taillard	- The application of different kinds of metaheuristics from a practical point of view - Examined the trade-off between running time and solution quality as well as the knowledge and efforts needed to implement and calibrate the algorithms
Wang et al. (2005) [34]	Polynomial algorithm	Makespan, total flowtime		- Proposed a polynomial algorithm - Demonstrated that the classical Johnson's rule is not the optimal solution for two-machine flow shop scheduling

Table 1. *Cont.*

Author, Year, and Reference	Algorithm	Optimization Criteria	Benchmarks	Summary
Agarwal (2006) [35]	Improvement heuristic approach (IHA)		Taillard, Carlier, Heller	- A weight parameter is used to perturb the data of the original problem to obtain improved solutions.
Rajendran (2007) [36]	Proposed ant colony algorithms	Makespan, total flowtime	Taillard	- PACO is evaluated by considering the benchmark problems and upper bound values for makespan - Minimizing total flowtime compared with the best heuristic solutions
Yagmahan et al. (2008) [37]	Ant colony optimization (ACO)	Makespan, total flow time, total machine idle time	Reeves	- Proposed that ACO algorithm can be applied for single or multiple objective cases considering other criteria such as mean flow time, total tardiness, and maximum tardiness
Zhang et al. (2009) [38]	Hybrid particle swam optimization (PSO) algorithm	Makespan, maximal machine workload		- A PSO algorithm and a tabu search (TS) algorithm are combined to solve the multiobjective FJSP with several conflicting and incommensurable objectives
Sayadi et al. (2010) [39]	Firefly metaheuristic (FMH)	Makespan	Demirkol	- Presented a new discrete firefly metaheuristic to minimize the makespan
Pan et al. (2011) [40]	Discrete artificial bee colony (DABC) algorithm	Total weighted earliness and tardiness penalties		- DABC algorithm is proposed to solve the lot-streaming flow shop scheduling problem with the criterion of total weighted earliness and tardiness penalties under both the idling and no-idling cases
Deng et al. (2012) [41]	Hybrid discrete differential evolution (HDDE) algorithm	Makespan	Ruiz	- Novel speed-up method based on network representation is proposed to evaluate the whole insert neighborhood of a job permutation and employed in HDDE
Li et al. (2013) [42]	Cuckoo search (CS)-based memetic algorithm (HCS)	Makespan, total flow time	Car, Rec	- A largest-ranked-value (LRV)-rule-based random key is used to convert the continuous position in CS into a discrete job permutation
Xie et al. (2014) [43]	Hybrid teaching–learning-based optimization (HTLBO) algorithm	Makespan, maximum lateness	Carlier's, Reeves Yamada's	- Combines a novel teaching–learning-based optimization algorithm for solution evolution and a variable neighborhood search (VNS) for fast solution improvement
Lin et al. (2015) [44]	Hybrid backtracking search algorithm (HBSA)	Makespan	Civicioglu	- HBSA is proposed for Permutation Flow-shop Scheduling Problem (PFSP) with the objective to minimize the makespan. - Crossover and mutation strategies as well as a simulated annealing (SA) mechanism are used to avoid premature and random insertion local search

Table 1. *Cont.*

Author, Year, and Reference	Algorithm	Optimization Criteria	Benchmarks	Summary
Lin et al. (2016) [45]	Hybrid biogeography-based optimization (HBBO) algorithm	Makespan	Hatami	- The path relinking heuristic is employed in the migration phase as a product local search strategy to optimize the assembly sequence - An insertion-based heuristic is used in the mutation phase
Deng et al. (2017) [46]	Competitive memetic algorithm (CMA)	Makespan, total tardiness	Taillard	- Some objective-specific operators are designed for each population - A special interaction mechanism between two populations is designed
Peng et al. (2017) [47]	Hybrid backtracking search (HBSA)	Makespan, energy consumption	Car	- In HBSA, the SA is a hybrid with original backtracking search to update the population
Bewoor et al. (2017) [48]	Hybrid particle swarm optimization (PHPSO) algorithm	Total flow time	Taillard	- The random key representation rule for converting the continuous position information values of particles to a discrete job permutation
Sun et al. (2017) [49]	Hybrid estimation of the distribution algorithm and cuckoo search (HEDA_CS)	Total Tardiness	Ruiz	- Designed some objective-specific operators and a special interaction mechanism - A competition mechanism is proposed to adaptively adjust the selection rates
Meng et al. (2018) [50]	Improved migrating birds optimization (IMMBO)	Makespan	Randomly generated	- Harmony-search-based scheme is designed to construct neighborhood of solutions - A leaping mechanism is introduced to avoid being trapped in the local optimum

2. Flow Shop Scheduling Problem Description

The flow shop scheduling problem can be described as follows: There are n jobs with the same process route to be processed, which need to be processed continuously on m machines. The layout of the flow shop is shown in Figure 1. The machining process satisfies the following assumptions: (1) All jobs and machines are available at time zero. (2) There is no prior priority among jobs. (3) Each job has m processing steps. (4) Every process must be processed on different machines and the process sequence cannot be changed. (5) Each machine can only process only one job at a time, and each workpiece can only be machined by one machine at a time. (6) The machining process cannot be interrupted during the processing. (7) All jobs must obey the first-in/first-out rule during processing. Note that the symbols used in this paper and their meanings are shown in Table 2.

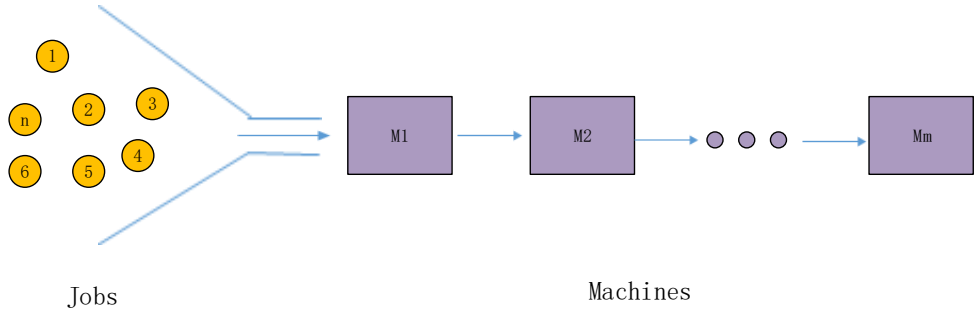

Figure 1. Layout of the flow shop processing environment.

Table 2. The meaning of the parameters used in this paper.

Parameter	Meaning
$J = \{1, 2, \ldots, n\}$	Set of n jobs
$M = \{1_1, 2, \ldots, m\}$	Set of m machines
p_{ij}	The processing time when job i is processed on machine tool j
S_{ij}	The starting time when job i is processed on machine tool j
C_{ij}	The finishing time when job i is processed on machine tool j
$\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$	A sequence of jobs
Π	Set of all jobs' sequences
$C_{max}(\pi)$	Makespan of one job's sequence π

The mathematical model of the FSSP makespan can be expressed as [4]

$$\min(\mathrm{Cmax}\pi) \tag{1}$$

S.t.

$$\sum_{k=1}^{n} X_{ik} = 1, i \in \{1, 2, \ldots n\} \tag{2}$$

$$\sum_{i=1}^{n} X_{ik} = 1, i \in \{1, 2, \ldots n\} \tag{3}$$

$$C_{ij} \leq \quad S_{i(j+1)} \tag{4}$$

$$C_{ij} \geq C_{(i-1)(j+1)} \tag{5}$$

$$C_{ij} = S_{ij} + P_{ij}. \tag{6}$$

Formula (2) is a constraint which indicates that different processes of the same job cannot be proceeded at the same time. Formula (3) is a machine constraint which indicates that one machine can only process one job at a time. Formula (4) is time constraint which indicates that the next process of a job cannot start before the current process is completed. Formula (5) indicates that if the subsequent machine $j + 1$ is in the machining state, the job will delay at machine j until machine $j + 1$ is idle. Formula (6) shows that for any job i, the completion time is determined by the starting time and the processing time on machine j. Therefore, Cmax(π) and C_{ij} can be calculated from Formulas (7) to (8):

$$C_{ij} = \max\left\{C_{i(j-1)} + t_{ij}, C_{(i-1)(j+1)}\right\}, i = 1, 2, \ldots n, j = 1, 2, \ldots m;\ where\ C_{i0} = 0, C_{0j} = 0, C_{i(m+1)} = 0 \tag{7}$$

$$C_{max}(\pi) = C_{nm}. \tag{8}$$

3. Hybrid Genetic Simulated Annealing Algorithm

3.1. Overview of the HGSA Algorithm

There are many algorithms which have been used to solve flow shop problems, such as the GA, particle swam optimization (PSO), the SA algorithm, the harmony search (HS) algorithm, ant colony optimization (ACO), the ABC, etc. Among these algorithms, the genetic algorithm has the advantages of strong global optimization ability, fast speed, strong versatility, and easy implementation. However, it has the drawback of poor local search ability, which decreases search efficiency, especially in the late period of optimization. Fortunately, the simulated annealing algorithm has strong local search capabilities to make up the shortcomings of the genetic algorithm. Hence, in this paper, the advantages of the two algorithms are combined. GA is employed to get an optimal or near-optimal solution among the solution space, and then SA is utilized to seek a better one based on the solution. In addition, in order to improve the search efficiency of SA, an annealing rate method based on the hormone regulation mechanism is used in this paper. The flowchart of the algorithm is shown in Figure 2. The simple pseudocode of HGSA is as follows:

```
HGSA {Hybrid Genetic Simulated Annealing Algorithm}

Initialize population by MME

while (not stop condition) do
    Step 1: Select the population
    Step 2: Crossover operation
        If(random==0)
            location-based Crossover
        else
            two-point Crossover
    Step 3: Mutation operation
        If(random==0)
            Insertion Mutation
        else
            Reverse order Mutation
    Step 4: Simulated annealing opearation
end_while
```

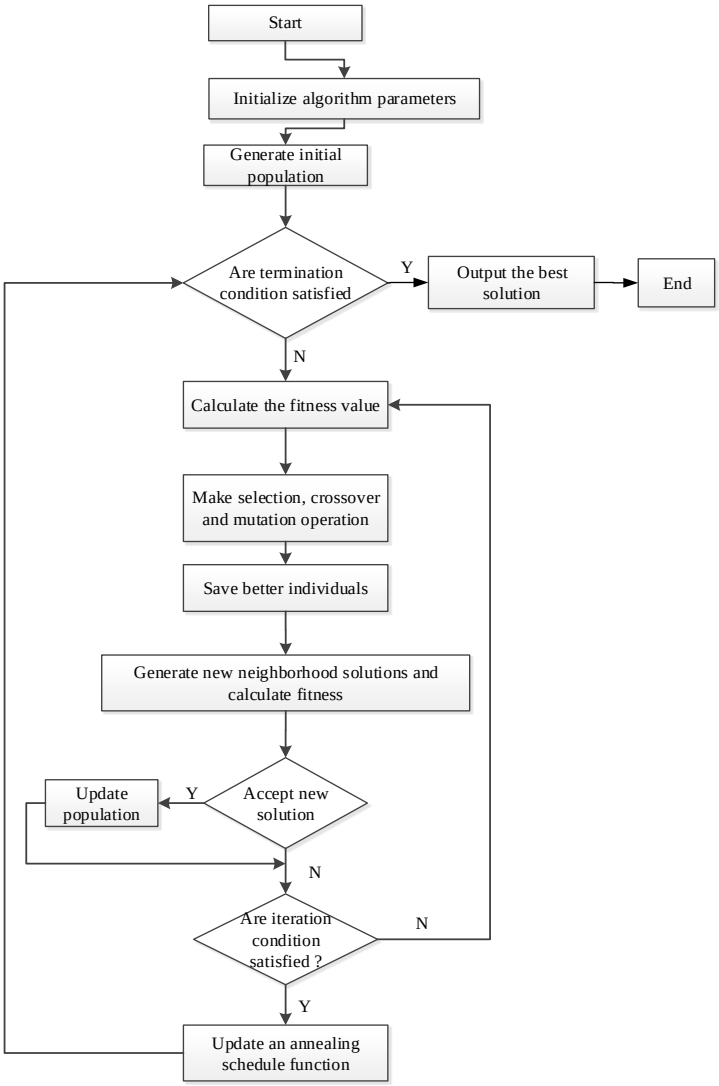

Figure 2. Algorithm flowchart of the hybrid genetic simulated annealing (HGSA) algorithm.

3.2. Encoding Representation

In this paper, the job ordering is used as the solution chromosome. The number of genes in the chromosome is equal to the quantity of jobs to be processed, which is n. If there are 10 jobs to be processed, p1 = [3, 5, 8, 7, 9, 6, 4, 2, 1, 10] can be taken as one of the chromosomes in the population. The number in the chromosome indicates the ID of the jobs being machined, and the position in the chromosome indicates the processing order of the jobs.

3.3. Initial Population

The initial population quality has a significant impact on the performance of an evolutionary algorithm. Good initial solutions can significantly improve the convergence rate and solution quality of the algorithm [51]. In this paper, the MME algorithm, combined with the NEH [5] and MM [29] heuristic algorithms, is used to generate the initial population. Ronconi [28] proved that the MME algorithm has better performance than the NEH algorithm for solving FSSPs. In [52], it was also proved that the initial solutions obtained by the MME algorithm explored the specific characteristics of the blocking condition, which made it outperform the NEH for minimizing the makespan of the Blocking Flow-Shop Scheduling (BFSS) problem.

The basic flow of the MME algorithm is as follows:

(1) Calculate the total machining time required for all the steps of each jobs. Rank the job with the smallest total processing time in the first position of the job sorting. Rank the job with the second smallest processing time in the last position of the job sorting and set $i = 2$.

(2) The rest of the n-2 jobs are arranged in ascending order according to the label function value of Formula (9). The job that takes ai will be sorted on i-th position of the job sequence, fixes the order, and records as π_0:

$$ai = r \times \sum_{j=1}^{m-1} \left| t_{i,j} - t_{i-1,j+1} \right| + (1 - r) \times \sum_{1}^{m} t_{ik}. \quad (9)$$

In Formula (9), r is a random number between [0, 1], $\sum_{j=1}^{m-1} \left| p_{i,j} - p_{i-1,j+1} \right|$ means the modulus of the difference between the two consecutive jobs on the adjacent machines, and $(1 - r) \times \sum_{1}^{m} p_{ik}$ indicates the job with a smaller total machining time is prioritized.

(3) $i = i + 1$. If $i < n$, go to step 2, otherwise go to step 4.

(4) Exchange the first two jobs in π_0 and add them to the machining sequence π_1. Calculate the makespan before and after the order exchanging. Then, reserve the sequence with the least makespan and fix the order of the two jobs. Denote the sequence as π_1 and set $k = 2$.

(5) Randomly select a job in the unprocessed sequence to insert to possible positions of π_1. Calculate the makespan after the job is added, then select the position that can minimize the completion time.

(6) Delete a job from the unprocessed sequence after its position is determined. Perform step 5 until all job positions are determined to form a new chromosome. Repeat the above steps 1–6 for p times to generate an initial population of p size individuals.

3.4. Crossover Operation

In this paper, we randomly select location-based crossover and two-point crossover, both of which are often used in genetic algorithms. The process is to generate an integer out of 0 and 1 randomly. If it is 0, select the location-based crossover operation; otherwise, select the two-point crossover operation. In the following description, P1 and P2 represent two parent individuals, and C1 and C2 represent two child individuals. The specific steps are shown as follows.

3.4.1. Location-Based Crossover

(1) Exchange the genes at several randomly selected positions of P1 and P2. Keep the genes in other locations unchanged.

(2) Perform conflict detection to delete the same gene as the exchange position in the original parent gene.

(3) Fill the gene vacancy with unused genes sequentially.

As shown in Figure 3a, there are 10 genes in this chromosome. Select positions 1, 3, 5, and 8 for exchange. The genes at the exchange position of P1 and P2 are directly exchanged to the same position of C2 and C1. Then, delete the same genes as the swap position in C1 and C2 and fill them with unused genes.

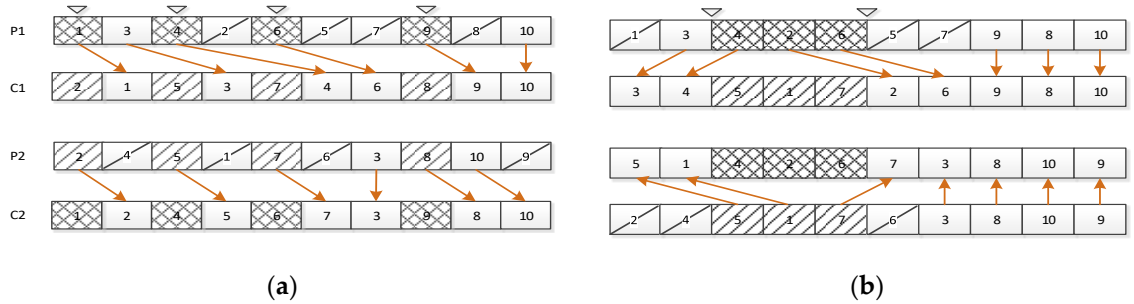

Figure 3. Procedure of location-based crossover operation (**a**) and two-point crossover operation (**b**).

3.4.2. Two-Point Crossover

(1) Two intersections are randomly generated, the gene fragments between the two points are copied from P1 and P2 to the corresponding positions of C1 and C2, and their positions and order are kept unchanged.

(2) The genes contained in the intersections between P1 and P2 are removed, the jobs not included in the intersections are copied to C2 and C1, and their order is maintained.

As shown in Figure 3b, the chromosome contains 10 genes (jobs). Select positions 3 and 6 for crossover. Then, the genes in P1 and P2 located between position 3 and 6 are directly copied to C1 and C2. The repetitive gene is removed, the remaining genes in P2 (white gene fragments in the figure) are sequentially filled into C1 to produce new individuals, and C2 is produced in the same manner.

3.5. Mutation Operation

In the genetic algorithm, the role of the mutation operator is to perturb the original chromosome to improve the local search ability and capability to jump out of local optima. In this paper, two commonly used mutation operations twors mutation and inversion mutation are randomly selected [53].

3.5.1. Twors Mutation

(1) Two positions in the chromosome are randomly selected.

(2) Insert the top-ranked gene into the back of another gene.

As shown in Figure 4a, the chromosome contains 10 genes (jobs). Positions 3 and 8 are selected. The gene 3 is inserted after the gene 8. Then, the other genes are sequentially moved back to obtain a new chromosome.

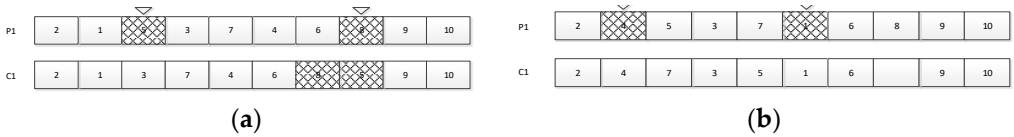

Figure 4. Procedure of insert mutation operation (**a**) and reverse mutation operation (**b**).

3.5.2. Inversion Mutation

(1) Two positions in the gene are randomly selected.

(2) The genes between the two positions were reversed, and the other gene positions are unchanged. The specific operation mode is shown in Figure 4b.

3.6. Simulated Annealing Operation

In the process of the HGSA algorithm, better individuals obtained from GA are sent to SA for improvement. The SA operation can help solutions to avoid falling into local optima. SA starts from a higher temperature (i.e., the initial temperature) and randomly finds the global optimal solution of the objective function in the neighborhood along with the continuous decrease at a certain annealing rate of temperature. At last, the algorithm ends and outputs the optimal solution when the termination condition of the algorithm is reached. The search efficiency of the SA is affected by some parameters such as the initial temperature, the neighborhood structure, the annealing rate, and the termination condition. These factors play a significant role in the performance of the HGSA and should be carefully implemented as follows.

3.6.1. Neighborhood Structure

The neighborhood structure will directly affect the local search efficiency. This paper adopts the more common neighborhood structure in the production scheduling problem, namely, exchange and insertion [54].

(1) Two points exchange. Exchange the genes at two positions which are randomly generated. For example, the coding of a chromosome is "154837629. The randomly generated gene positions are 3 and 6. Then, the new chromosome produced by exchanging genes at these two positions is "15783429".

(2) Insertion. Select two gene positions randomly. Then, the gene with the larger position number is inserted into the previous position of the gene with the smaller position number, and the smaller number position gene and the subsequent gene sequence is postponed. For example, the code of one chromosome is "15783429", the two randomly selected position numbers are 2 and 4, and the new chromosome obtained after the insertion operation is "18573429".

3.6.2. Initial Temperature

The initial temperature T_0 of the SA should be properly set because a too high initial temperature causes waste of calculation time, and a too small initial temperature causes the efficiency of the global search to decrease. This article uses the following formula to set the initial temperature:

$$T_0 = U_{max} - U_{min}. \tag{10}$$

In the formula, U_{max} is the maximum value of all feasible solutions generated by the MME algorithm, and U_{min} is the minimum value of all feasible solutions generated by the MME algorithm.

3.6.3. Annealing Rate

The annealing rate has a significant effect on the efficiency of the simulated annealing algorithm [55]. In order to improve the search efficiency of the algorithm, a new annealing rate method based on the hormone regulation mechanism [27] is used in this paper. The annealing rate can be obtained by Formulas (10)–(12):

$$T(k+1) = \alpha \times F_{down}(k) - \frac{k \times \Delta T}{\exp(k)} \tag{11}$$

$$F_{down}(k) = \frac{1}{1+k^n} \tag{12}$$

$$\Delta T = T(k+1) - T(k) \tag{13}$$

where α is a small constant, k represents the number of iterations, n represents the Hill coefficient ($n \geq 1$), ΔT is the difference between the current temperature ($k + 1$) and the previous temperature (k), and $\Delta T < 0$. The effect of the annealing rate function in the case of $k = 20$, $n = 1; 1.2; 1.5; 2.5$ is shown in Figure 5. In this paper, the value of n is chosen to be 1.5.

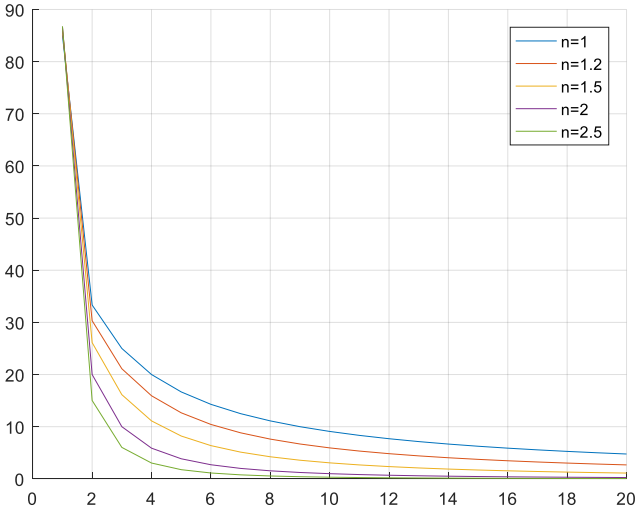

Figure 5. Annealing rate function.

3.6.4. The Terminating Condition

In SA, the termination criterion of the annealing procedure consists of the Metropolis sampling stability criterion of the inner loop and the external loop termination criterion. That is, when the new state or the current state's neighborhood solution space has been searched, the inner loop is jumped out, and the temperature is returned to the outside cycle. The outer loop termination criterion is also the algorithm termination criterion. In the algorithm design, the termination temperature is adopted. When the current temperature reaches the termination temperature requirement, the outer loop is terminated and the algorithm ends.

3.7. Benchmark Selected

In this work, 120 problems that were contributed to the OR-Library by E. TAILLARD [56] were used. The 120 problems called Ta001, Ta002 ... Ta120, respectively, were by E. TAILLARD (1993) and are often used to test the algorithm performance in flow shop problems. All these benchmarks can be downloaded from: http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html.

3.8. Computational Complexity

In this section, we derive the algorithmic complexity of the proposed scheme. The meanings of the symbols used are as follows: n is the number of jobs, m is the number of machines, T0 is the initial temperature, Tf is the termination temperature, P is the population size, and G is the number of iterations. At first, the complexity of MME is O (n^2m). In GA, the complexity of selection and mutation are O(n × logn) and O(nm), respectively. The computational complexity of SA is O($Pn^2m^2logTf/T0$). The total computational complexity of HGSA is O (n^2m) + G × O(n × logn) + G × O(n^2m) + G × O(nm) + G × O($Pn^2m^2logTf/T0$). It can be seen that the complexity of the proposed algorithm is related to population size, iterations, problem size, and parameters.

4. Experimental Results and Analysis

The algorithm of this paper was coded in Matlab language. The program running environment was Intel Corei5, 2.19 GHz CPU, and 8 GB RAM. We selected 120 classic examples proposed by Taillard as test data, and the data size of the Taillard benchmark was from 20jobs-5machines to 500jobs-20machines. The experimental parameters of the algorithm were set as follows [57,58]: the population size was 100, the crossover probability was 0.9, and the mutation probability was 0.1. In order to compare the performance of the HGSA with different Hill coefficients (n = 1; 1.2; 1.5; 2.5), the average relative percentage deviation (ARPD) values of different Hill coefficients with other parameters fixed are shown in Table 3. In these experiments, we chose the 200 × 20 and 500 × 20 instance in Talliard as the benchmarks. The results show that algorithm obtained the minimum ARPD when the Hill coefficient was equal to 1.5. So, the value of the Hill coefficient was chosen to be 1.5.

Table 3. The average relative percentage deviation (ARPD) value of different Hill coefficients on the performance of HGSA.

Problem Size	Function Factor (n)	ARPD%
200 × 20	1.0	9.21
	1.2	7.82
	1.5	5.16
	2.0	8.34
	2.5	11.06
500 × 20	1.0	6.28
	1.2	5.49
	1.5	3.29
	2.0	7.13
	2.5	9.56

In order to analyze the performance of the proposed algorithm, it was compared with the hybrid genetic algorithm (HGA) in [18], the improved iterated greedy algorithm (IIGA) in [19], the iterated greedy algorithm with a referenced insertion scheme (IG-RIS) in [20], the memetic algorithm (MA) in [21], and the discrete self-organizing migrating algorithm (DSOMA) in [22]. Table 4 shows the makespan results of the proposed algorithm and the other four algorithms tested in different sizes of Taillard benchmark. Each data scale has 10 sets of data, and the boldface figure in Table 4 represents the optimal solution obtained by the algorithms. It can be seen from the table that the algorithm proposed in this paper improved the results of 109 out of 120 examples, which indicates that the best results obtained by HGSA are better than the other four metaheuristics, reflecting the better search quality of HGSA. Compared with the MA, IG-RIS, and HIGA algorithms, the results of HGSA were greatly improved. Only in 11 cases with a small problem size were the results of HGSA slightly inferior to DSOMA. As the scale of the study increased, HGSA reflected a more obvious advantage. At the same time, the comparison with the HIGA and HGA algorithms showed that the MME initial population generation method and SA local search method based on hormone regulation mechanism could improve the optimization effect of the algorithm more effectively, especially for large-scale examples. Figure 6 is the Gantt chart of the ta001. The chart shows that the makespan obtained by proposed algorithm was 1324. The optimal schedule we got was [3, 17, 15, 16, 8, 6, 9, 18, 4, 2, 14, 5, 7, 11, 12, 10, 1, 19, 13, 20]. In order to further compare the advantages and disadvantages of the proposed algorithm and the other algorithms, three parameters of average error (AE), improvement percentage (IP), and ARPD were introduced to compare the effects of each algorithm.

Table 4. Makespan comparison of different algorithms on Taillard benchmarks.

Problem Size	Instance	Upper Bound	MA	IG-RIS	HGA	IIGA	DSOMA	HGSA
20 × 5	Ta001	1278	-	-	1449	1486	1374	**1324**
	Ta002	1359	-	-	1460	1528	**1408**	1442
	Ta003	1081	-	-	1386	1460	1280	**1098**
	Ta004	1293	-	-	1521	1588	**1448**	1469
	Ta005	1235	-	-	1403	1449	1341	**1291**
	Ta006	1195	-	-	1430	1481	**1363**	1391
	Ta007	1239	-	-	1461	1483	1381	**1299**
	Ta008	1206	-	-	1433	1482	1379	**1292**
	Ta009	1230	-	-	1398	1469	1373	**1306**
	Ta010	1108	-	-	1324	1377	1283	**1233**
20 × 10	Ta011	1582	-	-	1955	2011	**1698**	1713
	Ta012	1659	-	-	2123	2166	1833	**1718**
	Ta013	1496	-	-	1912	1940	1676	**1555**
	Ta014	1377	-	-	1782	1811	1546	**1516**
	Ta015	1419	-	-	1933	1933	1617	**1573**
	Ta016	1397	-	-	1827	1892	1590	**1457**
	Ta017	1484	-	-	1944	1963	1622	**1622**
	Ta018	1538	-	-	2006	2057	**1731**	1749
	Ta019	1593	-	-	1908	1973	1747	**1624**
	Ta020	1591	-	-	2001	2051	1782	**1722**
20 × 20	Ta021	2297	-	-	2912	2973	2436	**2331**
	Ta022	2099	-	-	2780	2582	**2234**	2280
	Ta023	2326	-	-	2922	3013	**2479**	2480
	Ta024	2223	-	-	2967	3001	**2348**	2362
	Ta025	2291	-	-	2953	3003	**2435**	2507
	Ta026	2226	-	-	2908	2988	2383	**2375**
	Ta027	2273	-	-	2970	3052	2390	**2341**
	Ta028	2200	-	-	2763	2839	2328	**2279**
	Ta029	2237	-	-	2972	3009	**2363**	2410
	Ta030	2178	-	-	2919	2979	**2323**	2401
50 × 5	Ta031	2724	3000	3002	3127	3161	3033	**2731**
	Ta032	2834	3199	3201	3438	3432	3045	**2934**
	Ta033	2621	3011	3011	3182	3211	3036	**2638**
	Ta034	2751	3128	3128	3289	3339	3011	**2785**
	Ta035	2863	3162	3166	3315	3356	3128	**2864**
	Ta036	2829	3166	3169	3324	3347	3166	**2907**
	Ta037	2725	3013	3013	3183	3231	3021	**2764**
	Ta038	2683	3067	3073	3243	3235	3063	**2706**
	Ta039	2552	2908	2908	3059	3072	2908	**2610**
	Ta040	2782	3111	3120	3301	3317	3120	**2784**
50 × 10	Ta041	2991	3638	3638	4251	4274	3638	**3198**
	Ta042	2867	3486	3507	4139	4177	3511	**3020**
	Ta043	2839	3483	3488	4083	4099	3492	**3055**
	Ta044	3063	3656	3656	4480	4399	3672	**3124**
	Ta045	2976	3629	3629	4316	4322	3633	**3129**
	Ta046	3006	3596	3621	4282	4289	3621	**3293**
	Ta047	3093	3692	3696	4376	4420	3704	**3232**
	Ta048	3037	3562	3572	4304	4318	3572	**3390**
	Ta049	2897	3527	3532	4162	4155	3541	**3237**
	Ta050	3065	3622	3624	4232	4283	3624	**3251**

Table 4. *Cont.*

Problem Size	Instance	Upper Bound	MA	IG-RIS	HGA	IIGA	DSOMA	HGSA
50 × 20	Ta051	3850	4479	4500	6138	6129	4511	**4105**
	Ta052	3704	4276	4276	5721	5725	4288	**3992**
	Ta053	3640	4261	4289	5847	5862	4289	**3900**
	Ta054	3720	4366	4377	5781	5788	4378	**3921**
	Ta055	3610	4261	4268	5891	5886	4271	**4020**
	Ta056	3681	4280	4280	5875	5863	4202	**3971**
	Ta057	3704	4304	4308	5937	5962	4315	**4093**
	Ta058	3691	4317	4326	5919	5926	4326	**4090**
	Ta059	3743	4315	4316	5839	5876	4329	**4107**
	Ta060	3756	4413	4428	5935	5958	4422	**4113**
100 × 5	Ta061	5493	6143	6151	6492	6397	6151	**5536**
	Ta062	5268	6022	6022	6353	6234	6064	**5302**
	Ta063	5175	5927	5927	6148	6121	6003	**5221**
	Ta064	5014	5756	5772	6080	6026	5786	**5044**
	Ta065	5250	5957	5960	6254	6200	6021	**5358**
	Ta066	5135	5812	5852	6177	6074	5869	**5197**
	Ta067	5246	5989	6004	6257	6274	6004	**5414**
	Ta068	5094	5856	5915	6225	6130	5924	**5130**
	Ta069	5448	6066	6123	6443	6370	6154	**5546**
	Ta070	5322	6142	6159	6441	6381	6186	**5480**
100 × 10	Ta071	5770	7016	7042	8115	8077	7042	**5964**
	Ta072	5349	6740	6791	7986	7880	6813	**5596**
	Ta073	5676	6878	6936	8057	8028	6943	**5796**
	Ta074	5781	7116	7187	8327	8348	7198	**5928**
	Ta075	5467	6810	6810	7991	7859	6815	**5748**
	Ta076	5303	6614	6666	7823	7801	6685	**5446**
	Ta077	5595	6783	6801	7915	7866	6827	**5679**
	Ta078	5617	6790	6874	7379	7913	6874	**5723**
	Ta079	5871	6981	7055	8226	8161	6092	**5934**
	Ta080	5845	6814	6965	8186	8114	6990	**5998**
100 × 20	Ta081	6202	7796	7844	10,745	10,700	7854	**6395**
	Ta082	6183	7845	7894	10,655	10,594	7910	**6433**
	Ta083	6271	7794	7794	10,672	10,611	7825	**6689**
	Ta084	6269	7797	7899	10,630	10,607	7902	**6419**
	Ta085	6314	7817	7901	10,548	10,539	7901	**6536**
	Ta086	6364	7826	7888	10,700	10,690	7921	**6527**
	Ta087	6268	7923	7930	10,827	10,825	8051	**6542**
	Ta088	6401	7984	8022	10,863	10,839	8025	**6712**
	Ta089	6275	7877	7969	10,751	10,723	7969	**6760**
	Ta090	6434	7913	7933	10,794	10,798	8036	**6621**
200 × 10	Ta091	10,862	13,348	13,406	15,739	15,319	13,507	**11,120**
	Ta092	10,480	13,242	13,313	15,534	15,085	16,458	**10,658**
	Ta093	10,922	13,318	13,416	15,755	15,376	13,521	**11,224**
	Ta094	10,889	13,290	13,344	15,842	15,200	13,686	**11,075**
	Ta095	10,524	13,247	13,360	15,692	15,209	13,547	**10,793**
	Ta096	10,326	13,079	13,192	15,622	15,109	13,247	**10,467**
	Ta097	10,854	13,517	13,598	15,877	15,395	13,910	**11,394**
	Ta098	10,730	13,483	13,504	15,733	15,237	13,830	**11,011**
	Ta099	10,438	13,277	13,310	15,573	15,100	13,410	**10,725**
	Ta100	10,657	13,325	13,439	15,803	15,340	13,744	**10,786**

Table 4. *Cont.*

Problem Size	Instance	Upper Bound	MA	IG-RIS	HGA	IIGA	DSOMA	HGSA
200 × 20	Ta101	11,195	14,912	14,912	20,148	19,681	15,027	**11,642**
	Ta102	11,203	14,876	15,002	20,539	20,096	15,211	**11,683**
	Ta103	11,281	15,057	15,186	20,511	19,913	15,247	**11,930**
	Ta104	11,275	14,975	15,082	20,461	19,928	15,174	**11,791**
	Ta105	11,259	14,733	14,970	20,339	19,843	15,047	**11,728**
	Ta106	11,176	14,861	15,101	20,501	19,942	15,212	**11,690**
	Ta107	11,360	14,988	15,099	20,680	20,112	15,168	**11,958**
	Ta108	11,334	14,926	15,141	20,614	20,056	15,247	**11,730**
	Ta109	11,192	14,885	15,034	20,300	19,918	15,136	**12,138**
	Ta110	11,288	14,921	15,122	20,437	19,935	15,243	**12,084**
500 × 20	Ta111	26,059	35,677	35,372	49,095	46,689	37,064	**26,859**
	Ta112	26,520	35,953	35,743	49,461	47,275	37,419	**27,220**
	Ta113	26,371	35,732	35,452	48,777	46,544	37,059	**27,511**
	Ta114	26,456	36,084	35,687	49,283	46,899	37,014	**26,912**
	Ta115	26,334	35,774	35,417	48,950	46,741	36,894	**26,930**
	Ta116	26,477	35,948	35,747	49,533	46,941	37,372	**27,354**
	Ta117	26,389	35,631	35,395	48,943	46,509	36,698	**26,888**
	Ta118	26,560	35,943	35,568	49,277	46,873	36,944	**27,229**
	Ta119	26,005	35,658	35,304	49,207	46,743	36,862	**28,103**
	Ta120	26,457	36,016	35,643	49,092	46,847	37,098	**27,290**

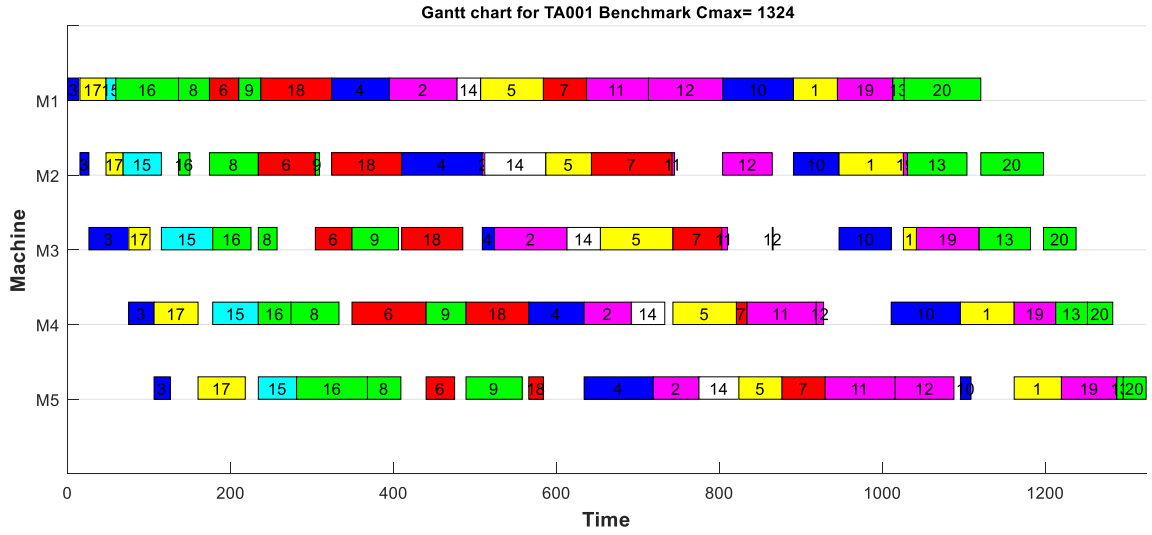

Figure 6. Gantt chart of the Ta001.

(1) AE represents the average error between the makespan of the algorithms and the lowest known upper bound values. The formula is

$$\mathrm{AE} = \frac{1}{K} \cdot \sum_{i=1}^{k} \frac{z_{opt} - z_i}{z_{opt}} \cdot 100 \tag{14}$$

where z_{opt} is the known upper bound value for each instance, z_i is the optimal makespan of each algorithm, and K is the number of instances. The average error comparison results of several algorithms are shown in Figure 7. As can be seen from the figure, the lowest AE value of HGSA in this paper was 5.58, which was lower than the lowest value of 20.516 for the DSOMA algorithm.

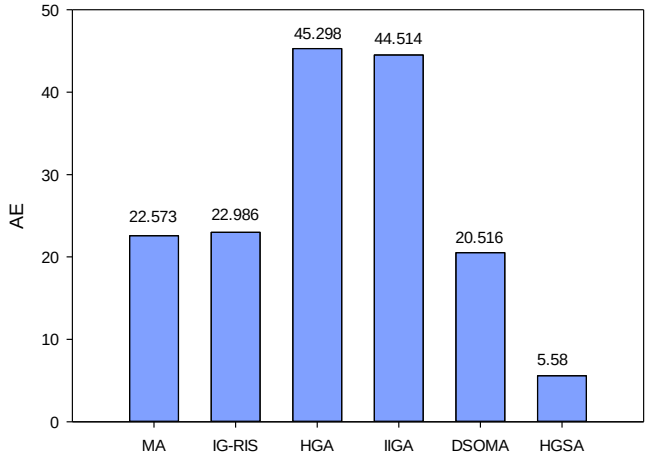

Figure 7. Comparison of average errors (AEs) on Taillard benchmarks.

(2) IP represents the improvement percentage on makespan of the HGSA algorithm compared with the other algorithms. The formula is

$$IP = \frac{C_{HGSA} - C_X}{C_{HGSA}} \cdot 100\% \tag{15}$$

where C_{HGSA} is the minimum makespan obtained by the HGSA algorithm, and C_X is the minimum makespan obtained by the other algorithms. Figure 8 shows the percentage improvement on makespan of the HGSA algorithm compared to the MA, IG-RIS, IIGA, and DSOMA algorithms. It can be seen from the figure that the superiority of the HGSA algorithm is not obvious compared with other algorithms when the scale of the benchmark is small. In the 20 × 20 instances, the HGSA algorithm performed slightly weaker than the DSOMA algorithm. However, with the increase of instances scale, that is, the increasing of workpieces and machine tools number, the advantages of HGSA became increasingly obvious. In the 500 × 20 instances, HGSA had an improvement of 31.7%, 30.5%, 80.6%, 36.1%, and 71.9% compared to MA, G-RIS, HGA, DSOMA, and IIGA, respectively. As the scale of instances increased, the improvement effect was more obvious, indicating that the algorithm performed better when the schedule was larger.

(3) ARPD represents the average relative percentage deviation between the result obtained by the algorithm and the optimal value. The formula is

$$\text{ARPD} = \frac{1}{K} \cdot \sum_{i=1}^{k} \frac{C_{xi} - C_{opti}}{C_{opti}} \times 100\% \tag{16}$$

where K is the number of the same scale instances, C_{xi} is the makespan of the algorithm x on the ith instance of the same size, and C_{opti} is the optimal value on the ith instance of the same size. Figure 9 shows that the ARPD of HGSA was similar to that of the DSOMA algorithm in the 20 × 20 instances, which was approximately 6%. In the other instances, the ARPD of HGSA, which were all less than 9%, was significantly lower than that of the existing algorithms. This demonstrates that the results obtained by our HGSA algorithm can be closer to the theoretical optimal values (upper bound) of these benchmark instances. Furthermore, the results show that HGSA performs well when the population size of jobs and the number of machines increases. Furthermore, the results show that HGSA performs well when the population size of jobs and the number of machines increases.

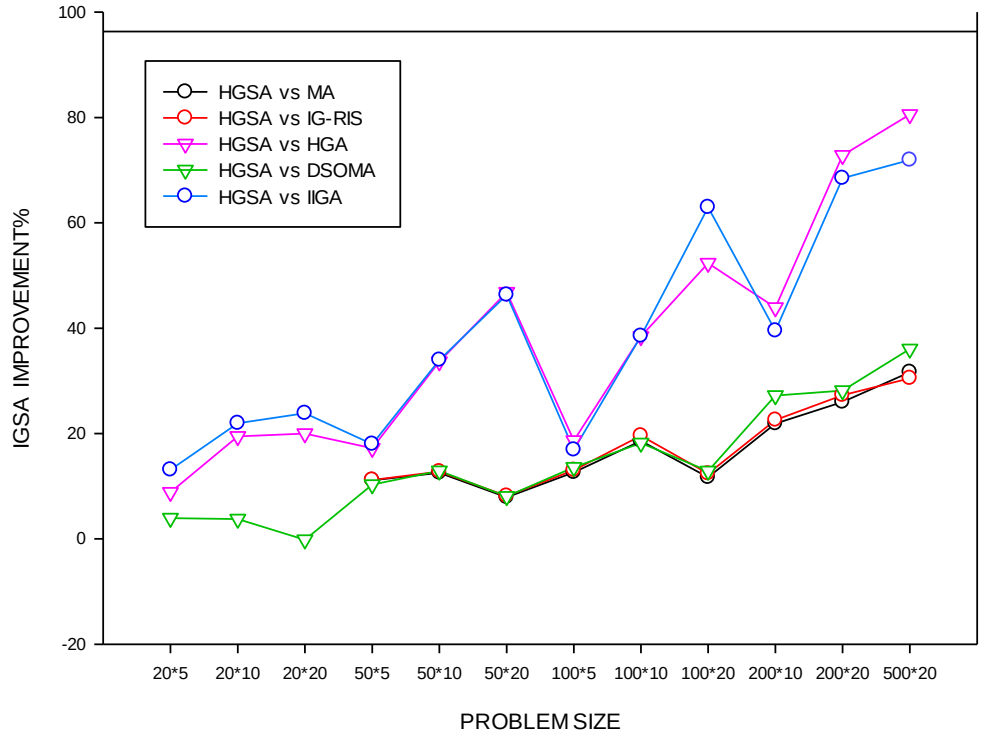

Figure 8. Improvement percentage (IP) of HGSA with other algorithms for makespan criterion.

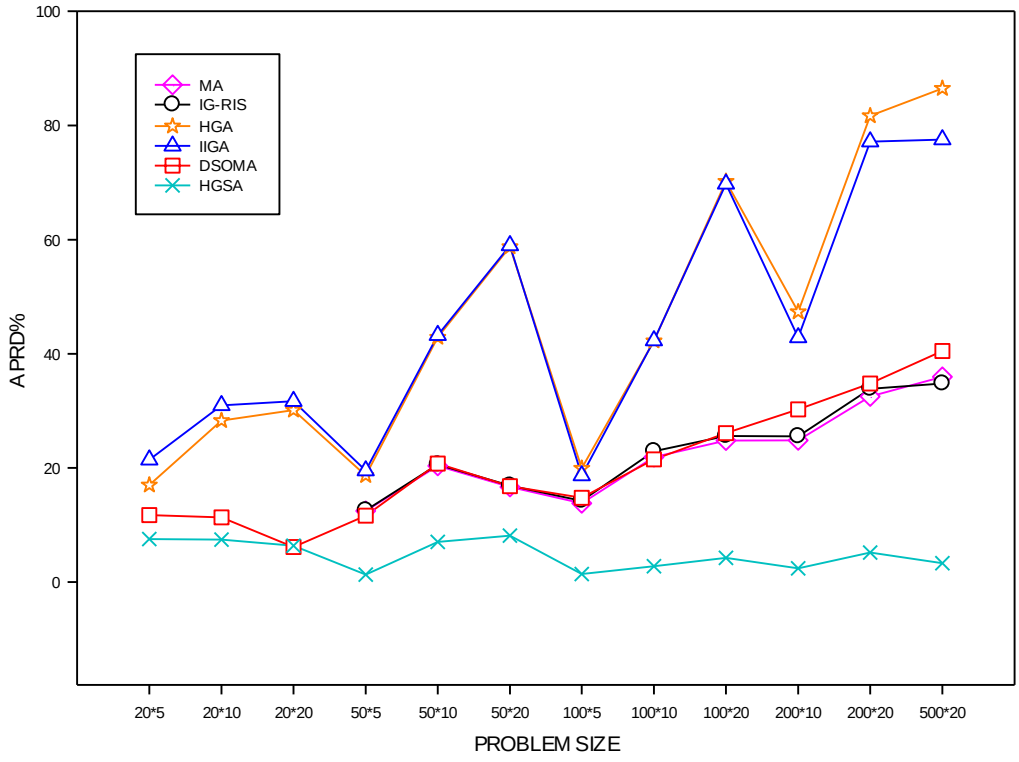

Figure 9. Comparison of ARPD on Taillard benchmarks.

5. Conclusions and Future Work

In this paper, a hybrid genetic simulated annealing algorithm based on the hormone regulation mechanism was designed for the flow shop scheduling problem. In order to ensure the quality of the initial population, the MME algorithm, combined with the NEH and MM algorithms, was used to generate the initial population with a certain quality and diversity. Then, SA based on the hormone

regulation mechanism was combined with GA to balance the efficiency and global search ability of the algorithm. Furthermore, randomly selected cross-mutation methods were introduced in GA to obtain promising results. Through the test on Talliard benchmarks, the results of HGSA were compared with MA, IG-RIS, HGA, IIGA, and DSOMA in terms of makespan, AE, IP, and ARPD. The results of our HGSA algorithm were closer to the upper bound of the Talliard benchmarks, which verified the effectiveness of the HGSA. In future research, we plan to add an energy consumption objective into the flow shop problem. We will also adapt our algorithm and apply it to the flow shop scheduling problem with job random arrivals together with an energy saving model. In addition, the proposed algorithm can also be extended to solve hybrid flow shop scheduling and job shop scheduling problems.

Author Contributions: H.W., S.L., and J.H. conceived of and designed the study. H.W., H.J., and J.H. worked on the algorithm design. H.W. and J.H. wrote the manuscript, made the figures, and reformatted the manuscript. All authors have read and approved the final manuscript.

Funding: This work is supported by the National Natural Science Foundation of China under Grant No. 51475097, 91746116, and 51741101; and Science and Technology Foundation of Guizhou Province under Grant No. [2015]4011, [2016]5013, [2015]02, and [2017]239.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Johnson, S.M. Optimal two- and three-stage production schedules with setup times included. *Naval Res. Logist. Q.* **2010**, *1*, 61–68. [CrossRef]
2. Hong, Z.Y.; Pang, H.L. Study on a constructive heuristic algorithm based on compromise policy for Blocking flow-shop scheduling. *Syst. Eng. Theory Pract.* **2008**, *28*, 114–118.
3. Ignall, E.; Schrage, L. Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems. *Oper. Res.* **1965**, *13*, 400–412. [CrossRef]
4. Bansal, S.P. Minimizing the Sum of Completion Times of n Jobs over m Machines in a Flowshop A Branch and Bound Approach. *AIIE Trans.* **1977**, *9*, 306–311. [CrossRef]
5. Nawaz, M.E.E.E., Jr.; Ham, I. A heuristic algorithm for the m -machine, n -job flow-shop sequencing problem. *Omega* **1983**, *11*, 91–95. [CrossRef]
6. Cui, Q.; Xiuli, W.U.; Jianjun, Y.U. Improved genetic algorithm variable neighborhood search for solving hybrid flow shop scheduling problem. *Comput. Integr. Manuf. Syst.* **2017**, *23*, 1917–1927.
7. Marichelvam, M.K.; Prabaharan, T.; Yang, X.S. Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. *Appl. Soft Comput.* **2014**, *19*, 93–101. [CrossRef]
8. Burdett, R.L.; Kozan, E. A sequencing approach for creating new train timetables. *OR Spectr.* **2010**, *32*, 163–193. [CrossRef]
9. Rathinam, B. Rule based heuristic approach for minimizing total flow time in permutation flow shop scheduling. *Teh. Vjesn.* **2015**, *22*, 25–32. [CrossRef]
10. Govindan, K.; Balasundaram, R.; Baskar, N.; Asokan, P. A Hybrid Approach for Minimizing Makespan In Permutation Flowshop Scheduling. *J. Syst. Sci. Syst. Eng.* **2017**, *26*, 50–76. [CrossRef]
11. Han, Y.Y.; Gong, D.; Sun, X. A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. *Eng. Optim.* **2015**, *47*, 927–946. [CrossRef]
12. Pan, C.H.; Huang, H.C. A hybrid genetic algorithm for no-wait job shop scheduling problems. *Expert Syst. Appl.* **2009**, *36*, 5800–5806. [CrossRef]
13. Gao, K.; Pan, Q.; Suganthan, P.N.; Li, J. Effective heuristics for the no-wait flow shop scheduling problem with;total flow time minimization. *Int. J. Adv. Manuf. Technol.* **2013**, *66*, 1563–1572. [CrossRef]
14. Bertolissi, E. Heuristic algorithm for scheduling in the no-wait flow-shop. *J. Mater. Process. Technol.* **2000**, *107*, 459–465. [CrossRef]
15. Nowicki, E.; Smutnicki, C. A fast tabu search algorithm for the permutation flow-shop problem. *Eur. J. Oper. Res.* **1996**, *91*, 160–175. [CrossRef]
16. Sayoti, F.; Ri, M.E. Golden Ball Algorithm for solving Flow Shop Scheduling Problem. *Ijimai* **2016**, *4*, 15–18. [CrossRef]

17. Kasihmuddin, M.S.B.M.; Mansor, M.A.B.; Sathasivam, S. Genetic Algorithm for Restricted Maximum k-Satisfiability in the Hopfield Network. *Int. J. Interact. Multimedia Artif. Intell.* **2016**, *4*, 52.
18. Tseng, L.; Lin, Y. A hybrid genetic algorithm for no-wait flowshop scheduling problem. *Int. J. Prod. Econ.* **2010**, *128*, 144–152. [CrossRef]
19. Ding, J.Y.; Song, S.; Gupta, J.N.; Zhang, R.; Chiong, R.; Wu, C. An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem. *Appl. Soft Comput.* **2015**, *30*, 604–613. [CrossRef]
20. Tasgetiren, M.F.; Kizilay, D.; Pan, Q.K.; Suganthan, P.N. Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. *Comput. Oper. Res.* **2017**, *77*, 111–126. [CrossRef]
21. Pan, Q.K.; Wang, L.; Sang, H.Y.; Li, J.Q.; Liu, M. A High Performing Memetic Algorithm for the Flowshop Scheduling Problem with Blocking. *IEEE Trans. Autom. Sci. Eng.* **2013**, *10*, 741–756.
22. Davendra, D.; Bialicdavendra, M. Scheduling flow shops with blocking using a discrete self-organising migrating algorithm. *Int. J. Prod. Res.* **2013**, *51*, 2200–2218. [CrossRef]
23. Eddaly, M.; Jarboui, B.; Siarry, P. Combinatorial particle swarm optimization for solving blocking flowshop scheduling problem. *J. Comput. Des. Eng.* **2016**, *3*, 295–311. [CrossRef]
24. Burdett, R.L.; Kozan, E. Evolutionary algorithms for flowshop sequencing with non-unique jobs. *Int. Trans. Oper. Res.* **2000**, *7*, 401–418. [CrossRef]
25. Yin, H.L. Genetic Algorithm Nested with Simulated Annealing for Big Job Shop Scheduling Problems. In Proceedings of the 2013 9th International Conference on Computational Intelligence and Security (CIS), Emei Moutain, China, 14–15 December 2013.
26. Andresen, M.; BräSel, H.; MöRig, M.; Tusch, J.; Werner, F.; Willenius, P. Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop. *Math. Comput. Model.* **2008**, *48*, 1279–1293. [CrossRef]
27. Dai, M.; Tang, D.; Giret, A.; Salido, M.A.; Li, W.D. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robot. Comput.-Integr. Manuf.* **2013**, *29*, 418–429. [CrossRef]
28. Ronconi, D.P. A note on constructive heuristics for the flowshop problem with blocking. *Int. J. Prod. Econ.* **2004**, *87*, 39–48. [CrossRef]
29. Merz, P.; Freisleben, B. Memetic Algorithms for the Traveling Salesman Problem. *Complex Syst.* **1997**, *13*, 297–345.
30. Abdoun, O.; Abouchabaka, J.; Tajani, C. Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem. *Int. J. Emerg. Sci.* **2012**, *2*, 61–77.
31. Koulamas, C.; Kyparisis, G.J. The three-stage assembly flowshop scheduling problem. *Comput. Oper. Res.* **2001**, *28*, 689–704. [CrossRef]
32. Chang, P.; Hsieh, J.; Lin, S. The development of gradual-priority weighting approach for the multi-objective flowshop scheduling problem. *Int. J. Prod. Econ.* **2002**, *79*, 171–183. [CrossRef]
33. Fink, A.; Vos, S. Solving the continuous flow-shop scheduling problem by metaheuristics. *Eur. J. Oper. Res.* **2003**, *151*, 400–414. [CrossRef]
34. Wang, J.; Xia, Z.Q. Flow-shop scheduling with a learning effect. *J. Oper. Res. Soc.* **2005**, *56*, 1325–1330. [CrossRef]
35. Agarwal, A.; Colak, S.; Eryarsoy, E. Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *Eur. J. Oper. Res.* **2006**, *169*, 801–815. [CrossRef]
36. Rajendran, C.; Ziegler, H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *Eur. J. Oper. Res.* **2007**, *155*, 426–438. [CrossRef]
37. Yagmahan, B.; Yenisey, M.M. Ant colony optimization for multi-objective flow shop scheduling problem. *Comput. Ind. Eng.* **2008**, *54*, 411–420. [CrossRef]
38. Zhang, G.; Shao, X.; Li, P.; Gao, L. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Comput. Ind. Eng.* **2009**, *56*, 1309–1318. [CrossRef]
39. Sayadi, M.K.; Ramezanian, R.; Ghaffarinasab, N. A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *Int. J. Ind. Eng. Comput.* **2010**, *1*, 1–10. [CrossRef]
40. Pan, Q.K.; Tasgetiren, M.F.; Suganthan, P.N.; Chua, T.J. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *China Mech. Eng.* **2011**, *181*, 2455–2468. [CrossRef]

41. Deng, G.; Gu, X. A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion. *Comput. Oper. Res.* **2012**, *39*, 2152–2160. [CrossRef]
42. Li, X.; Yin, M. A hybrid cuckoo search via Lévy flights for the permutation flow shop scheduling problem. *Int. J. Prod. Res.* **2013**, *51*, 4732–4754. [CrossRef]
43. Xie, Z.; Zhang, C.; Shao, X.; Lin, W.; Zhu, H. An effective hybrid teaching–learning-based optimization algorithm for permutation flow shop scheduling problem. *Adv. Eng. Softw.* **2014**, *77*, 35–47. [CrossRef]
44. Lin, Q.; Gao, L.; Li, X.; Zhang, C. A hybrid backtracking search algorithm for permutation flow-shop scheduling problem minimizing makespan and energy consumption. *Comput. Ind. Eng.* **2015**, *85*, 437–446. [CrossRef]
45. Lin, J.; Zhang, S. An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem. *Comput. Ind. Eng.* **2016**, *97*, 128–136. [CrossRef]
46. Deng, J.; Wang, L.; Wang, S.Y.; Zheng, X.L. A competitive memetic algorithm for the distributed two-stage assembly flow-shop scheduling problem. *Int. J. Prod. Res.* **2017**, *54*, 3561–3577. [CrossRef]
47. Chen, P.; Wen, W.; Li, R.; Li, X. A hybrid backtracking search algorithm for permutation flow-shop scheduling problem minimizing makespan and energy consumption. In Proceedings of the 2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Singapore, 10–13 December 2017.
48. Bewoor, L.; Prakash, V.C.; Sapkal, S. Evolutionary Hybrid Particle Swarm Optimization Algorithm for Solving NP-Hard No-Wait Flow Shop Scheduling Problems. *Algorithms* **2017**, *10*, 121. [CrossRef]
49. Sun, Z.; Gu, X. Hybrid Algorithm Based on an Estimation of Distribution Algorithm and Cuckoo Search for the No Idle Permutation Flow Shop Scheduling Problem with the Total Tardiness Criterion Minimization. *Sustainability* **2017**, *9*, 953.
50. Meng, T.; Pan, Q.K.; Li, J.Q.; Sang, H.Y. An improved migrating birds optimization for an integrated lot-streaming flow shop scheduling problem. *Swarm Evol. Comput.* **2018**, *38*, 64–78. [CrossRef]
51. Yahyaoui, A.; Fnaiech, N.; Fnaiech, F. A Suitable Initialization Procedure for Speeding a Neural Network Job-Shop Scheduling. *IEEE Trans. Ind. Electron.* **2011**, *58*, 1052–1060. [CrossRef]
52. Liu, S.Q.; Kozan, E. Scheduling a flow shop with combined buffer conditions. *Int. J. Prod. Econ.* **2009**, *117*, 371–380. [CrossRef]
53. Tao, S.; Wang, S. An algorithm for weighted sub-graph matching based on gradient flows. *Inf. Sci.* **2016**, *340–341*, 104–121. [CrossRef]
54. Ku, L. An Adaptive Variable Neighbourhood Search Algorithm for the Hybrid Flowshop Scheduling Problem. *Syst. Eng.* **2015**, *11*, 121–129.
55. Dai, M.; Tang, D.; Zheng, K.; Cai, Q. An Improved Genetic-Simulated Annealing Algorithm Based on a Hormone Modulation Mechanism for a Flexible Flow-Shop Scheduling Problem. *Adv. Mech. Eng.* **2013**, *5*, 124903. [CrossRef]
56. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]
57. Salido, M.A.; Escamilla, J.; Giret, A.; Barber, F. A genetic algorithm for energy-efficiency in job-shop scheduling. *Int. J. Adv. Manuf. Technol.* **2016**, *85*, 1303–1314. [CrossRef]
58. Rajkumar, R.; Shahabudeen, P. An improved genetic algorithm for the flowshop scheduling problem. *Int. J. Prod. Res.* **2009**, *47*, 233–249. [CrossRef]