Effective Heuristic Algorithms Solving the Jobshop Scheduling Problem with Release Dates

Tao Ren 1,*, Yan Zhang 1, Shuenn-Ren Cheng 2, Chin-Chia Wu 3,1, Meng Zhang 1, Bo-yu Chang 1, Xin-yue Wang 1 and Peng Zhao 1

1 College of Software, Northeastern University, Shenyang 110819, China; 1801251@stu.neu.edu.cn (Y.Z.);
   z18604016585@163.com (M.Z.); cby19680713@163.com (B.-y.C.); chinaxywang@163.com (X.-y.W.);
   zhaopeng_608@163.com (P.Z.)
2 Department of Business Administration, Cheng Shiu University, Kaohsiung 83347, Taiwan;
   k0252@gcloud.csu.edu.tw
3 Department of Statistics, Feng Chia University, Taichung 40724, Taiwan; cchwu@fcu.edu.tw
* Correspondence: chinarentao@163.com; Tel.: +86-1384-036-6956

Received: 8 June 2020; Accepted: 23 July 2020; Published: 25 July 2020

Abstract: Manufacturing industry reflects a country’s productivity level and occupies an important share in the national economy of developed countries in the world. Jobshop scheduling (JSS) model originates from modern manufacturing, in which a number of tasks are executed individually on a series of processors following their preset processing routes. This study addresses a JSS problem with the criterion of minimizing total quadratic completion time (TQCT), where each task is available at its own release date. Constructive heuristic and meta-heuristic algorithms are introduced to handle different scale instances as the problem is NP-hard. Given that the shortest-processing-time (SPT)-based heuristic and dense scheduling rule are effective for the TQCT criterion and the JSS problem, respectively, an innovative heuristic combining SPT and dense scheduling rule is put forward to provide feasible solutions for large-scale instances. A preemptive single-machine-based lower bound is designed to estimate the optimal schedule and reveal the performance of the heuristic. Differential evolution algorithm is a global search algorithm on the basis of population, which has the advantages of simple structure, strong robustness, fast convergence, and easy implementation. Therefore, a hybrid discrete differential evolution (HDDE) algorithm is presented to obtain near-optimal solutions for medium-scale instances, where multi-point insertion and a local search scheme enhance the quality of final solutions. The superiority of the HDDE algorithm is highlighted by contrast experiments with population-based meta-heuristics, i.e., ant colony optimization (ACO), particle swarm optimization (PSO) and genetic algorithm (GA). Average gaps 45.62, 63.38 and 188.46 between HDDE with ACO, PSO and GA, respectively, are demonstrated by the numerical results with benchmark data, which reveals the domination of the proposed HDDE algorithm.

Keywords: jobshop scheduling; release date; heuristic algorithm; discrete differential evolution

1. Introduction

Smart manufacturing has become a new competitive advantage in many countries. Therefore, governments have issued smart-manufacturing-related strategies, such as ‘Made in China 2025’ in China and ‘Industrial 4.0.’ in Germany. One goal of smart manufacturing is to improve customer satisfaction by meeting the requirement of customization, which is a common customer demand pattern in the smart era. Jobshop is a manufacturing system that makes products with bespoke productive processes. For example, a gear is a key part in automatic equipment such as transmission. Precision gears...
are manufactured by computerized numerical control (CNC) machine tools. In accordance with product shape, CNC machine tools handle gear blanks with preset processing routes over time. The manufacturing process of gears can be abstracted as a jobshop scheduling (JSS) model where each job has its own release date. Akers and Friedman [1] were the first to formulate a four-machine JSS problem. Since then, considerable attention has been focused on this research area. A comprehensive survey on the JSS problems can be found in Jain and Meeran [2] and Zhang et al. [3].

Generally, the optimal objectives of the JSS model are makespan that can minimize maximum machine loads, or total completion time (TCT) that can minimize work-in-process inventory. Academically, these criteria are linearized with designated weight factors for substituting bi-objective optimization. However, the error of the weight factors will result in a deviation between linearization and dual objectives. Cheng and Liu [4] indicated the advantage of the total $k$-power completion time (TKCT) criterion, which balances makespan and TCT. On the one hand, TKCT approaches makespan as $k \to \infty$ because only the maximum completion time dominates the objective value in this situation. On the other hand, TKCK is just TCT as $k = 1$. Therefore, the total quadratic completion time (TQCT) criterion is introduced as a trade-off for machine loads and work-in-process inventory.

Garey et al. [5] reported the strong NP-hardness for the JSS TCT problem even for the two-machine case, indicating that the JSS TQCT problem with release dates is at least as difficult as the former. Therefore, no polynomial time algorithm can solve the studied problem unless $P = NP$. Constructive heuristic and meta-heuristic algorithms are designed to handle different scale instances. The shortest processing time, dense schedule (SPT-DS) heuristic, is proposed to achieve feasible solutions for large-scale instances. A preemptive single-machine-based lower bound is designed to estimate the optimal solution and reveal the performance of the heuristic. A hybrid discrete differential evolution (HDDE) algorithm is provided to achieve near-optimal solutions for medium-scale instances, where multi-point insertion and a local search scheme enhance the quality of final solutions. The superiority of the presented algorithms are confirmed through conducting extensive contrast experiments.

Compared with the known literature, the contributions of this study mainly include three aspects:

1. A JSS model with the TQCT criterion is established, where each job is available at its own release date. This scheduling model simulates the production environment in which jobs arrive to the system over time.
2. An HDDE algorithm is presented to achieve high-quality schedules in a given time, where tri-point insertion in crossover operator and local search scheme with exchange neighbourhood enhance the quality of final solutions.
3. An innovative heuristic algorithm combining SPT and dense scheduling rule is proposed, which provides dominant feasible solutions on large-scale instances.
4. A preemptive single-machine-based lower bound is proposed to estimate the optimal schedule, which can serve as an estimation of the optimal solution to evaluate the performance of approximate algorithms.

The remainder of the paper is arranged as follows. A literature survey is presented in Section 2. Section 3 establishes a mixed integer programming (MIP) model for the JSS problem. Section 4 proposes an SPT-DS heuristic algorithm and a new lower bound. Section 5 provides the DDE algorithm to deal with medium-scale instances. Section 6 executes numerical experiments and reports the analysis results. Section 7 concludes the paper and indicates future research directions.

2. Literature Review

The JSS model is one of the most complex problems in combination optimization [6]. Except for several polynomial-solvable cases, most JSS problems are NP-complete [7]. Therefore, related research for JSS problems mainly puts attention on the application of metaheuristic algorithms to achieve near-optimal solutions.
The criterion for most single-objective optimizations is to minimizing makespan. Khadwilard et al. [8] adopted firefly algorithm and used one-third fractional factorial experimental design to set parameters. Gao et al. [9] presented a hybrid particle-swarm tabu search (TS) algorithm, where particle swarm optimization (PSO) with a balanced strategy provides diverse and elite initial solutions for the TS algorithm. Qiu and Lau [10] proposed a novel hybrid algorithm integrating clonal selection, immune network, and PSO concept. Wang and Duan [11] designed a hybrid biogeography-based optimization algorithm that introduced chaos theory and strategy to explore and improve population diversity. Keesari and Rao [12] applied teaching-learning-based optimization (TLBO), in which the learner phase in the original TLBO is modified. Asadzadeh [13] addressed an agent-based local search genetic algorithm (GA). Peng et al. [14] presented a hybrid TS/path relinking algorithm that referred to features including a path construction procedure on the basis of distances of solutions and a special mechanism to select the reference solution. Kurdi [15] proposed a hybrid island model GA with naturally inspired self-adaptation phase strategy, which stroked a balance between diversification and intensification during search. Cheng et al. [16] presented the hybrid evolutionary algorithm (HEA), in which a TS procedure is incorporated into the framework of an evolutionary algorithm. HEA embraces several distinguishing features such as the longest common sequence-based recombination operator and a similarity-and-quality-based replacement criterion for population updating. Dao et al. [17] applied the concept of a parallel processing to bat algorithm.

Some extended studies on JSS problems have been conducted to minimize makespan. Saidi-Mehrabad et al. [18] considered a specific production system constituted by a warehouse, a network guide-path, several machines, and the transportation between machines. They built an integrated mathematical model composed of the JSS and conflict-free routing problems for automated guided vehicles to convey jobs and applied a two-stage ant colony algorithm. Sundar et al. [19] discussed the JSS problem with no-wait constraint and presented a hybrid artificial bee colony (HABC) algorithm where the artificial bee colony algorithm coupled with a local search effectively coordinates the various components of HABC. Kuhpfahl and Bierwirth [20] considered the JSS problem with total weighted tardiness objective and developed an approach on the cornerstone of disjunctive graphs to capture the neighborhood structure. They found a structural classification of neighbourhood operators and some new analytical results. Kundakcı and Kulak [21] introduced efficient hybrid GA methodologies for the dynamic JSS problem. Ku and Beck [22] evaluated four MIP formulations for the classical JSS problem using three modern optimization software.

Some researchers have conducted studies on bi-objective optimization. Phanden et al. [23] applied a simulation-based GA algorithm with restart scheme to handle three special cases to minimize mean tardiness and makespan. Nguyen et al. [24] developed four multi-objective genetic programming-based hyper-heuristic methods for the automatic design of scheduling policies and simultaneous handling of multiple scheduling decisions, owing to the complexity of each scheduling decision as well as the interactions among different decisions. They proposed a diversified multi-objective co-evolution method by which different multiple scheduling decisions can evolve in sub-populations. May et al. [25] merged and upgraded NSGA-II and SPEA-II to build a green GA to optimize productivity and environmental objectives. Salido et al. [26] focused on the JSS problem with machine speed scaling where variable-speed machines have different energy efficiencies to minimize makespan and energy use. They also designed a GA to model and solve the JSS problem. On the basis of the machine speed scaling framework, Zhang and Chiong [27] minimized the total weighted tardiness and the total energy consumption and combined a multi-objective GA incorporated with two problem-specific local improvement strategies.

The known studies on the JSS problems are basically concentrated on makespan criterion for research convenience. However, in an industrial environment, the makespan criterion is too simple to offer insight for complex scheduling event. Therefore, this article focuses on the TQCT criterion, which can effectively balance energy consumption of machines and cost of work-in-process inventory.
3. MIP Model

In a JSS system, a series of \( m \) machines handles a number of \( n \) jobs following their own preset processing routes. \( O_{i,j} \) denotes the \( i \)-th execution of job \( j \). \( Q_{i,j} \) denotes that job \( j \) is processed on machine \( i, i=1,2,\ldots, m, j=1,2,\ldots, n \). The processing time of job \( j \) on machine \( i \) is denoted as \( p_{i,j} \), where \( p_{i,j} = 0 \) indicates that operation \( O_{i,j} \) is nothingness. Release date \( r_{j} \) is the earliest time when job \( j \) is available.

Each machine processes a job at most once and no re-circulation occurs in the system. The intermediate buffer between any adjacent machines has unlimited storage capacity. At any moment, each machine can process at most one job, and each job can access at most one machine. Any preemptive phenomenon is prohibitive, i.e., a starting operation must be continued until it is completed. The optimal objective is to achieve a feasible schedule for minimizing the total quadratic completion time.

The definition of symbol for the MIP model is shown in Table 1.

| Variables | Meanings |
|-----------|----------|
| \( n \)   | total number of jobs; |
| \( m \)   | total number of machines; |
| \( r_{j} \) | release date of job \( j \); |
| \( p_{i,j} \) | processing time of job \( j \) on machine \( i \); |
| \( C_{i,j} \) | completion time of job \( j \) on machine \( i \); |
| \( C_{j} \) | maximum completion time of job \( j \); |
| \( a_{i,j,l} \) | \( 1 \), if job \( j \) is processed on machine \( i \) before machine \( l \); \( 0 \), otherwise. |
| \( z_{i,j,k} \) | \( 1 \), if job \( k \) precedes job \( j \) immediately on machine \( i \); \( 0 \), otherwise. |
| \( y_{i,j} \) | \( 1 \), if the first operation of job \( j \) is processed on machine \( i \); \( 0 \), otherwise. |
| \( M \) | a large positive number. |

Therefore, the following MIP model is established.

Minimize \( \sum_{j=1}^{n} C_{j}^{2} \)

Subject to:

\[
C_{i,j} + M \times (1 - y_{i,j}) \geq r_{j} + p_{i,j}, i = 1, \ldots, m, j = 1, \ldots, n \tag{1}
\]

\[
\sum_{j=0}^{n} z_{i,j,l} = 0, i = 1, \ldots, m \tag{2}
\]

\[
\sum_{k=0}^{n} z_{i,j,k} = 1, j = 0, \ldots, n, i = 1, \ldots, m \tag{3}
\]

\[
\sum_{k=0}^{n} z_{i,j,k} \leq 1, i = 1, \ldots, m, k = 0, \ldots, n, j = 0, \ldots, n \tag{4}
\]

\[
C_{i,j} - p_{i,j} + M \times (1 - a_{i,j,l}) \geq C_{i,j}, i,l = 1, \ldots, m, j = 1, \ldots, n \tag{5}
\]

\[
C_{i,j} - C_{j,k} + M \times (1 - z_{i,j,k}) \geq p_{i,j}, i = 1, \ldots, m, k = 1, \ldots, n \tag{6}
\]

\[
C_{j} \geq C_{i,j}, i = 1, \ldots, m, j = 1, \ldots, n \tag{7}
\]

\[
a_{i,j,l} \in \{0,1\}, z_{i,j,k} \in \{0,1\}, C_{i,j} \geq 0, C_{j} \geq 0, i,l = 1, \ldots, m, j,k = 1, \ldots, n \tag{8}
\]

Constraint (1) states that job \( j \) must be processed after its release date \( r_{j} \). Constraints (2) to (5) ensure the uniqueness of decision variable \( z_{i,j,k} \). Constraint (6) illustrates the relationship of completion time between two adjacent operations of the same job. Constraint (7) defines the relationship between the completion times of two adjacent jobs on the same machine. Constraint (8) explains the meaning of decision variable \( C_{j} \), which is the maximum completion time of each job. Constraint (9) specifies the 0–1 variables and value ranges for the inputs.
4. SPT-DS Heuristic and Lower Bound

Given that the JSS problem is NP-completed, a heuristic algorithm can effectively achieve feasible solutions in a very short time for massive production setting, in which continuous production is more important than optimal solution. Townsend [28] reported the optimality of the shortest processing first (SPT) rule for the single-machine TQCT problem. An SPT-based heuristic is proposed to handle the JSS TQCT problem with release dates. A preemptive single-machine lower bound based on the optimality of the shortest remaining processing time (SRPT) rule for the preemptive single-machine TQCT problem is presented to evaluate the performance of the heuristic [29].

4.1. SPT-DS Heuristic

Matrix \( T \) stores the processing route of each job. According to the precedence of operations in matrix \( T \), all current operations are stored in set \( A \), where current operation means an immediate successor of the operation that has just been processed for a given job. Matrix \( S \) and \( C \) store the start time and completion time of all operations, respectively, where \( s_{ij} \) denotes the start time of job \( j \) on machine \( i \), i.e., \( Q_{ij} \), and \( c_{ij} \) denotes the completion time of job \( j \) on machine \( i \), \( \forall j \in n, \forall i \in m, s_{ij} \in S, c_{ij} \in C \). Each job \( j \) has a non-negative release date \( r_j \) and processing time \( p_{ij} \) denotes the execution duration of \( Q_{ij} \).

Step 1: Set the start time \( s_{ij} \) equals to release date \( r_j \) and let \( c_{ij} = 0, \forall j \in n, \forall i \in m \), Go to Step 2.

Step 2: Assign operation \( Q_{ij} \) with the earliest start time \( s_{ij} \) in set \( A \). If there is more than one job with the earliest start time, then schedule the job with the shortest processing time. Go to Step 3.

Step 3: Update the completion time \( c_{ij} = s_{ij} + p_{ij} \). Delete operation \( Q_{ij} \) in set \( A \). Check the elements in matrix \( T \) to determine the machine on which the next execution of job \( j \) is processed. If machine \( k \) is the target, then add operation \( Q_{kj} \) in set \( A \). Go to Step 4.

Step 4: Update the start time of the remaining operations of job \( j \), let \( s_{ij} = \max(s_{k,j}, c_{ij}), 1 \leq k \leq m \), and the start processing time of machine \( i \), \( s_{ih} = \max(s_{ij}, c_{ij}), 1 \leq h \leq n \). Go to Step 5.

Step 5: If set \( A \neq \emptyset \), turn to Step 2. Otherwise, stop the algorithm and calculate the objective value.

A numerical example is provided below to better explain the SPT-DS heuristic.

Example 1. A JSS problem with three jobs \( \{J_1, J_2, J_3\} \) and three machines \( \{M_1, M_2, M_3\} \) is involved. The input data including processing times, release dates and processing routes are presented in Table 2.

| Jobs | Processing Routes | Processing Times | Release Dates |
|------|-------------------|------------------|--------------|
| \( J_1 \) | \( M_2, M_3, M_1 \) | 4, 3, 2 | 1 |
| \( J_2 \) | \( M_3, M_2, M_1 \) | 6, 2, 2 | 0 |
| \( J_3 \) | \( M_1, M_3, M_2 \) | 2, 5, 1 | 2 |

The processing times and the processing routes of three jobs in Table 2 are restored in Matrices \( P \) and \( T \). Matrices \( S \) and \( C \) record the start and completion times of each operation \( Q_{ij} \), \( 1 \leq i \leq 3, 1 \leq j \leq 3 \). In initial status, the completion times of all operations are set to zero, and the start times of jobs on all machines are equal to their release dates. The first raw in matrix \( T \) is \( [2, 3, 1] \) which means the first execution of job \( j \), \( 1 \leq j \leq 3 \), can be processed on machine \( M_2, M_3 \) or \( M_1 \). Therefore, the data stored in set \( A \) is \( \{Q_{2,1}, Q_{3,2}, Q_{1,3}\} \). The elements in each matrix are shown as follows.

\[
T_{3 \times 3} = \begin{bmatrix} 2 & 3 & 1 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix}, \quad P_{3 \times 3} = \begin{bmatrix} 4 & 2 & 1 \\ 2 & 2 & 2 \\ 3 & 6 & 5 \end{bmatrix}, \quad C_{3 \times 3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad S_{3 \times 3} = \begin{bmatrix} 1 & 0 & 2 \\ 1 & 0 & 2 \\ 1 & 0 & 2 \end{bmatrix}.
\]

Schedule the operation in set \( A \) with the earliest start time. For example, \( s_{2,1} = 1, s_{3,2} = 0, s_{1,3} = 2 \). Therefore, the first operation of job \( J_2 \) is scheduled on machine \( M_3 \), i.e., \( Q_{3,2} \). The completion time of
Therefore, each of these SRPT schedules is a lower bound of the JSS TQCT problem. As a larger lower bound can provide better performance for minimization criteria, the dominative one among the m lower bounds is selected as the final lower bound for the JSS TQCT problem.

4.2. Well-Designed Lower Bound

An effective lower bound is proposed for the JSS TQCT problem to estimate the optimal value for large-scale instances. The basic idea of designing the lower bound is to relax the constraints of the relationship between machines and non-preemption for each job. Letting $I_{i,j,k}^{SRPT}$ denote that operation $O_{i,j}$ is scheduled according to the SRPT rule, lower bound $Z_{ij}^{LB}$ is shown as follows.

$$Z_{ij}^{LB} = \max_{1 \leq j \leq m} \left\{ \sum_{i=1}^{n} \left( C_{i,j}^{LB} \right)^2 \right\}$$  \hspace{1cm} (10)

where

$$C_{i,j}^{LB} = \max_{1 \leq j \leq S_i} \left\{ r_x + \sum_{k=x}^{S_i} I_{i,j,k}^{SRPT} \right\}$$

Given the NP-hardness of the JSS TQCT problem, i.e., it cannot be solved in polynomial time for large-scale instances, an effective lower bound is usually served as a substitute of the optimal schedule to evaluate approximation algorithms. Generally, the lower bound is the optimal solution of a relaxation version of the original problem. An m-machine JSS TQCT problem is reduced to m preemptive versions of the single-machine scheduling (SMS) problem to minimize the TQCT criterion with release dates. Bai [29] proved that the SRPT rule is optimal for the preemptive SMS TQCT problem. Therefore, each of these SRPT schedules is a lower bound of the JSS TQCT problem. As a larger lower bound can provide better performance for minimization criteria, the dominative one among the m lower bounds is selected as the final lower bound for the JSS TQCT problem.

To better explain the lower bound, a numerical example is provided as follows.

**Example 2.** The input data are similar to those in Example 1. The lower bound schedule with the SRPT rule on the three machines is shown in Figure 2. The scheduling on machine $M_2$ is explained in detail for better understanding. At time $t = 0$, only job $J_2$ is available. At time $t = 1$, job $J_1$ is released and the remaining time of job $J_2$ is 5 > $p_{2,1} = 4$. On the basis of the SRPT rule, job $J_2$ is preempted by job $J_1$. At time $t = 2$, similarly, job $J_1$ is preempted by job $J_3$. Given that no extra job is released after the completion of job $J_3$, the remaining parts of
The associated lower bound value is

\[ Z^{LB} = \max\{2^2 + 4^2 + 6^2, 11^2 + 3^2 + 6^2, 9^2 + 4^2 + 14^2\} = 293. \]

5. Effective HDDE Algorithm

The SPT-DS heuristic algorithm can quickly output a feasible solution for large-scale instances. However, the quality of the SPT-DS schedule weakens for medium-scale instances. For example, the mean gap is approximately 185% for 20 random instances with the combination \( m \times n = 3 \times 30 \). Relatively, meta-heuristic is a type of higher level heuristic that controls the entire search process. Thus, the global optimal solutions can be achieved systematically and efficiently. The differential evolution (DE) algorithm is a population-based evolutionary algorithm for global optimization in a continuous search space. The DE algorithm improves candidates by executing mutation and crossover operations, and renews the population through greedy one-to-one selection. Compared with the classical evolutionary-based meta-heuristics, the superiority of the DE algorithm involves easy implementation, simple structure, robust, and fast convergence. The traditional DE algorithm was originally used to solve the global optimization problem in the continuous search space, in which individuals were encoded by floating point numbers that are invalid for discrete variables. This section transforms individuals into operation-based sequences and proposes the HDDE algorithm to solve the JSS problem.

5.1. Encoding and Decoding

Given the diversity of process routes for each job in a JSS model, a job-number-based (JNB) representation is introduced to encode and decode between individuals and feasible schedules in the DDE algorithm. The encoding scheme linearizes the operations in a feasible schedule according to starting times and represents an individual by the corresponding job number. The decoding scheme transforms individuals into operation-based sequences and proposes the HDDE algorithm to solve the JSS problem.

Example 3. A JSS model with three jobs \( \{J_1, J_2, J_3\} \) and three machines \( \{M_1, M_2, M_3\} \) is provided to explain the procedure in detail. The input data are similar to those in Example 1. The data in Matrices \( T_{3 \times 3} \) and \( P_{3 \times 3} \) are presented as follows.
\[ T_{3 \times 3} = \begin{bmatrix} 2 & 3 & 1 \\ 3 & 2 & 3 \\ 1 & 1 & 2 \end{bmatrix} \text{ and } \hat{P}_{3 \times 3} = \begin{bmatrix} 4 & 6 & 2 \\ 3 & 2 & 5 \\ 2 & 2 & 1 \end{bmatrix} \]

where \( t_{23} = 3 \) indicates that machine \( M_3 \) processes operation \( O_{2,3} \) and \( \hat{p}_{23} = 5 \) is its associated processing time. The SPT-DS schedule in Figure 1 can be linearized as \( \pi = \{ O_{1,2}, O_{1,1}, O_{1,3}, O_{2,2}, O_{2,1}, O_{3,2}, O_{2,3}, O_{3,1}, O_{3,3} \} \). In encoding, the individual of schedule \( \pi \) is denoted as \( JS = \{2, 1, 3, 2, 1, 2, 3, 1, 3\} \), and the corresponding machine sequence is \( MS = \{3, 2, 1, 2, 3, 1, 3, 1, 2\} \). In decoding, the operations of individual \( JS \) are assigned to permutation \( MS \) and scheduled with the dense scheduling rule, which generates a feasible schedule.

5.2. Initialization

The main task of initialization is to generate the initial population and set the parameters. The initial population is randomly generated by swapping adjacent genes in a chromosome. Orthogonal tests are designed to determine four parameters, including population size \( \Lambda \), the maximum number of iterations \( \tau_{\text{max}} \), mutant factor \( Z \), and crossover factor \( Y \), thereby enhancing the performance of the algorithm. The pseudo-code for initialization is provided in Procedure 1.

**Procedure 1: Initialization**

1. **Input:** \( n, \Lambda, m / n \) is the number of jobs, \( m \) is the number of machines, \( \Lambda \) is the size of the population
2. **Output:** \( \text{pop[} \Lambda \text{]} [m \times n] \) ← Array to store initial population;
3. **begin**
4. \( j \leftarrow 1; \)
5. for \( (i \text{ from } 0 \text{ to } m \times n - 1) \) do
6. \( \text{for } (x \text{ from } i \text{ to } i + m - 1) \) do
7. \( \text{pop}[0][x] \leftarrow j; \)
8. \( \text{end for} \)
9. \( i \leftarrow i + m; j \leftarrow j + 1; \)
10. \( \text{end for} \)
11. for \( (i \text{ from } 1 \text{ to } \Lambda) \) do
12. \( \text{for } (j \text{ from } 0 \text{ to } m \times n - 1) \) do
13. \( P[i] \leftarrow \text{pop}[i - 1][j]; \)
14. \( \text{end for} \)
15. \( t \leftarrow \text{rand}(1, m \times n); \)
16. \( f \leftarrow \text{rand}(1, m \times n); \)
17. if \( (t \neq f) \) \( \text{temp} \leftarrow P[t - 1]; \)
18. \( P[t - 1] \leftarrow P[f - 1]; \)
19. \( P[f - 1] \leftarrow \text{temp}; \)
20. else \( i \leftarrow i - 1; \)
21. \( \text{Continue}; \)
22. \( \text{end if} \)
23. for \( (i \text{ from } 0 \text{ to } m \times n - 1) \) do
24. \( \text{pop}[i][j] \leftarrow P[i]; \)
25. \( \text{end for} \)
26. \( \text{return pop[} \Lambda \text{]} [m \times n]; \)
27. **end**
5.3. Mutation and Crossover

The mutation operator is executed on all individuals in the current population to generate the mutant individual \( V^k_h = [x^k_{h1}, x^k_{h2}, \ldots, x^k_{h_m}] \) at iteration \( k \), where \( h \in [1, \Lambda] \). Two pairs of target individuals \( \{X^k_{a1}, X^k_{b1}\} \) and \( \{X^k_{a2}, X^k_{b2}\} \) are randomly selected from the \((k-1)\)th population to implement the mutation operation with current optimal individual \( X^k_{best} \) (the individual with the optimal objective value in the \((k-1)\)th population). The mutation operator is expressed as follows.

\[
V^k_h = X^k_{best} \oplus Z \odot (X^k_{a1} - X^k_{b1}) \oplus Z \odot (X^k_{a2} - X^k_{b2})
\]

where \( Z \in [0,1] \) is the mutant factor. In Equality (11), operator \( \odot \) is executed as follows.

\[
C^k_{h_z} = Z \otimes (X^k_{a_z} - X^k_{b_z}) = \begin{cases} 
X^k_{a_z} - X^k_{b_z}, & \text{if } rand(\cdot) < Z, \\
0, & \text{otherwise}
\end{cases}, w \in m \times n
\]

where \( rand(\cdot) \) is a random number assigned to each element of the difference individual and \( x = 1, 2 \). Operator \( \oplus \) in Equality (11) is executed as follows.

\[
V^k_h = \text{mod}(\left(X^k_{best} + C^k_{h_1} + C^k_{h_2} + n - 1\right), n) + 1, w \in m \times n
\]

where \( \text{mod}(\cdot) \) is the modular operator, which guarantees that each vector in a mutation individual represents a valid job number. The pseudo-code for the mutation operator is provided in Procedure 2.

| Procedure 2: Mutation operator |
|--------------------------------|
| 1 | Input: pop[\[1, \Lambda]\] \([m \times n]\)/initial population |
| 2 | Output: \( V^k_h \) |
| 3 | Begin |
| 4 | do |
| 5 | |
| 6 | \( \alpha_x, \beta_x \leftarrow \text{random}(1,\Lambda)/x = 1,2 \) |
| 7 | while(\( \alpha_x, \beta_x \) (\( x = 1,2 \) is pairwise different)); |
| 8 | /Operation of operator \( \odot \)/ |
| 9 | \( X^k_{a_x,j} \) \(-\) An individual randomly selected from the population; |
| 10 | \( X^k_{b_x,j} \) \(-\) Another non-repeating individual randomly selected from the population; |
| 11 | for (\( j \) from 1 to \( m \times n \)) do //\( n \) is the number of jobs and \( m \) is the number of machines |
| 12 | \( \gamma \leftarrow \text{random}(0,1) \); |
| 13 | if (\( \gamma < Z \)); |
| 14 | \( C^k_{h_x,j} = X^k_{a_x,j} - X^k_{b_x,j} \); |
| 15 | else |
| 16 | \( C^k_{h_x,j} = 0 \); |
| 17 | end if |
| 18 | end for |
| 19 | /\(^*\) Operation of operator \( \oplus \)/ |
| 20 | for (\( j \) from 1 to \( m \times n \)) do |
| 21 | \( V^k_{h,j} = \text{mod}(\left(X^k_{best} + C^k_{h_1,j} + C^k_{h_2,j} + n - 1\right), n) + 1 \); |
| 22 | end for |
| 23 | return \( V^k_{h} \); |
| 24 | end |

The crossover operator selectively inserts a mutation individual into a target individual \( X^k_{h} \) to generate a trial individual \( U^k_{h} \). A series of random numbers is assigned to the vectors in a mutant
individual $V^k_h$. For each element in the mutation individual, if the random number $rand(\cdot) < Y$, then the element is retained; otherwise, it is removed, where $Y \in (0, 1)$ is a crossover factor. Tri-point insertion is executed to obtain a temporary individual, where the remaining elements of a mutation individual are divided into three parts and inserted in three randomly selected positions on target individual $X^{k-1}_h$. The trial individual $U^k_h$ is achieved by deleting the extra elements (repeat more than $m$ times) in the temporary individual from left to right. The pseudo-code for crossover operator is presented in Procedure 3.

**Procedure 3: Crossover operator**

1. **Input**: $V^k_h$
2. **Output**: $U^k_h$
3. Begin
4. for $(j$ from $0$ to $m \times n - 1)$ do
   5. $r \leftarrow$ random $(0,1)$;
   6. if $(r >= Y)$ do
      7. $V^k_h \leftarrow$ Remove $v^k_{hj}$, $V^k_h$
   8. end if
9. end for
10. $U^k_h \leftarrow X^{k-1}_h$;
11. $V^k_h \leftarrow$ Randomly split $V^k_h$ into three parts;
12. Generate three random insertion points of $U^k_h$;
13. The divided three parts of $V^k_h$ is inserted into the three random insertion points of $U^k_h$ in sequence;
14. $U^k_h \leftarrow$ Remove duplicate operations;
15. return $U^k_h$
End

A numerical example is provided below to further describe the mutation and crossover operations.

**Example 4.** A JSS model with three jobs \{J\_1, J\_2, J\_3\} and three machines \{M\_1, M\_2, M\_3\} is provided to explain the procedure in detail. Letting mutation and crossover factors be set to $Z = 0.3$ and $Y = 0.5$, respectively. The procedure of mutation operation is illustrated in Tables 3 and 4, while that of the crossover operation is illustrated in Table 5 and Figure 3.

**Table 3. Generation of temporary vector $C^k_{h\_1}$ ($Z = 0.3$).**

| Individual | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ |
|------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| $X_{\alpha 1}$ | 2 | 1 | 3 | 1 | 2 | 1 | 3 | 2 | 3 |
| $X_{\beta 1}$ | 2 | 3 | 1 | 3 | 2 | 3 | 1 | 1 | 2 |
| $X_{\alpha 1} - X_{\beta 1}$ | 0 | -2 | 2 | -2 | 0 | -2 | 2 | 1 | 1 |
| $rand$ | 0.20 | 0.37 | 0.06 | 0.18 | 0.26 | 0.68 | 0.86 | 0.50 | 0.15 |
| $C^k_{h\_1}$ | 0 | 0 | 2 | -2 | 0 | 0 | 0 | 0 | 1 |

**Table 4. Generation of temporary vector $C^k_{h\_2}$ ($Z = 0.3$).**

| Individual | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ |
|------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| $X_{\alpha 2}$ | 3 | 2 | 3 | 1 | 2 | 1 | 3 | 1 | 2 |
| $X_{\beta 2}$ | 2 | 1 | 1 | 3 | 2 | 3 | 1 | 3 | 2 |
| $X_{\alpha 2} - X_{\beta 2}$ | 1 | 1 | 2 | -2 | 0 | -2 | 2 | -2 | 0 |
| $rand$ | 0.56 | 0.02 | 0.38 | 0.26 | 0.66 | 0.18 | 0.59 | 0.89 | 0.15 |
| $C^k_{h\_2}$ | 0 | 1 | 0 | -2 | 0 | -2 | 0 | 0 | 0 |
Table 5. Generation of trial individual \((Y = 0.5)\).

| Individual | \(x_1\) | \(x_2\) | \(x_3\) | \(x_4\) | \(x_5\) | \(x_6\) | \(x_7\) | \(x_8\) | \(x_9\) |
|------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| \(X^k_{\text{best}}\) | 2 | 3 | 1 | 2 | 1 | 3 | 3 | 1 | 2 |
| \(G^k_h\) | 0 | 0 | 2 | −2 | 0 | 0 | 0 | 0 | 1 |
| \(G^k_{h_2}\) | 0 | 1 | 0 | −2 | 0 | −2 | 0 | 0 | 0 |
| \(\oplus\) | 2 | 4 | 3 | −2 | 1 | 1 | 3 | 1 | 2 |
| \(Y^k\) | 2 | 1 | 3 | 1 | 1 | 1 | 3 | 1 | 2 |
| rand | 0.56 | 0.28 | 0.32 | 0.48 | 0.66 | 0.18 | 0.59 | 0.89 | 0.15 |
| \(Y^k_h\) | 1 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Figure 3. Diagram of crossover operation.

5.4. Hill-Climbing-Based Improvement Strategy

A hill-climbing-based improvement strategy is introduced to enhance the quality of the final solution, aiming to balance diversification and intensification. Hill-climbing is a local search algorithm, which starts with an arbitrary solution of a given problem, then attempts to seek a better solution by making an incremental change to the solution. The change procedure executes continuously until no further improvements can be found.

However, the hill-climbing consumes considerable running time to obtain the local optimal solution in practical settings. For saving computing resource and improving algorithm efficiency, the improvement strategy is presented as follows. If the local optimal solution is found within the specified iterations, the solution is recorded; otherwise, the current best solution is recorded. The dominative parent individual is denoted as \(U_{h}^k\). Ten iterations are conducted for the improvement strategy. Each iteration generates 20 neighborhood solutions by exchanging two elements in the parent individual. The optimal one among the 20 neighborhood solutions is compared with the current optimal individual (COI). If the new optimal solution is dominated, then the COI is updated by it and the next iteration is executed. Otherwise, the improvement strategy will continue to be dominated by the original COI until it reaches the maximum number of iterations. The improvement strategy is performed with probability \(\theta\) to save computing time. The pseudo-codes for the improvement strategy are shown in Procedure 4.

\[
\begin{align*}
X^k_{h} & \quad 2 \quad 1 \quad 3 \quad 2 \quad 2 \quad 1 \quad 3 \quad 1 \quad 3 \\
Y^k_h & \quad 1 \quad 3 \quad 1 \quad 1 \quad 2 \\
\text{insert} & \\
1 \quad 3 \quad 2 \quad 1 \quad 3 \quad 1 \quad 1 \quad 2 \quad 2 \\
\text{Deduplication} & \\
1 \quad 3 \quad 2 \quad 1 \quad 3 \quad 1 \quad 2 \quad 2 \quad 3
\end{align*}
\]
Procedure 4: Hill-climbing-based improvement strategy

1. Input: θ, \( U^k_h \)
2. Output: \( U^k_h \)
3. Begin
4. \( q \leftarrow \text{random}(0,1) \);
5. If \( q > \theta \)
6. for (i from 0 to \( m \times n - 1 \))
7. better\[i\] \( \leftarrow \) \( U^k_h, i \); //store better individual;
8. end for
9. for (z from 1 to 10)
10. \( \text{init} \leftarrow \text{the value of the objective function corresponding to better individual} \);
11. for (y from 1 to 20)
12. Neighboor\[y\] \( \leftarrow \) Randomly exchange the two elements of better to produce neighborhood individual.
13. obj\[y\] \( \leftarrow \text{the value of objective function of Neighboor}[y] \);
14. end for
15. best_nei \( \leftarrow \) the best one from 20 neighborhood individuals;
16. best_obj \( \leftarrow \) the objective value of best_nei;
17. if (best_obj < \( \text{init} \))
18. for (i from 0 to \( m \times n - 1 \))
19. better\[i\] \( \leftarrow \) best_nei\[i\];
20. end for
21. end if
22. end for
23. end if
24. return \( U^k_{h,i'} \);
25. End

5.5. Selection

Population updating simulates the law of natural selection: survival of the fittest. Trial individual \( U^k_h \) is decoded as a feasible schedule. If the objective value of \( U^k_h \) dominates that of the target individual \( X^{k-1}_{h'} \), then the individual in the current population will be updated; otherwise, the target individual will be retrained. Executing the mutation, crossover and selection operations for all target individuals generate a new population. The iteration procedure is repeated until the termination condition is satisfied.

5.6. Framework of the HDDE Algorithm

With the previous procedures combined, the entire framework of the HDDE algorithm is provided in Algorithm 1.
Algorithm 1: the HDDE algorithm:

```plaintext
1 Input: Parameter $\tau_{\text{max}}$, $Z$, $Y$, $\Lambda$, $\theta$
2 Output: $X_{k_{\text{best}}}$
3 Begin
4 /* Initialization phase */
5 $\text{pop}[\Lambda][m \times n] \leftarrow \text{Randomly generate the initial population};$
6 Evaluate the initial population;
7 $k \leftarrow 1;$
8 $X_{k_{\text{best}}} \leftarrow \text{Individual with the optimal objective value in the current population} ;$
9 while ($k \leq \tau_{\text{max}}$) do
10   for ($h$ from 1 to $\Lambda$) do
11      /* Mutation phase */
12      $V_{k_{h}} \leftarrow \text{the mutant individual after the mutation operation};$
13      /* Crossover phase */
14      $U_{k_{h}} \leftarrow \text{the variant individual after the crossover operation};$
15      /* Improvement strategy phase */
16      Enter the improvement strategy operator to update $U_{k_{h}}$;
17      /* Selection Phase */
18      if ($F(U_{k_{h}}) < F(X_{k-1_{h}})$) // $F(X)$ is the objective value corresponding to the individual $X$;
19         $X_{k_{h}} = U_{k_{h}}$;
20      else
21          $X_{k_{h}} = X_{k-1_{h}}$;
22      end if
23      Update $X_{k_{\text{best}}}$;
24      $k = k + 1$;
25 end while
26 return $X_{\tau_{\text{max}}_{\text{best}}}$;
27 End
```

6. Numerical Simulation Experiment

Numerous simulation experiments were executed to evaluate the effectiveness of the HDDE algorithm and SPT-DS heuristic for medium- and large-scale instances, respectively. The proposed algorithms were implemented in C++ language, and simulations were run on a Dell computer with an Intel Core i5-8400 (2.8 GHz × 6) CPU and 8.00 GB RAM. The input data were generated because no standard benchmark was available for the JSP problem with release dates. Release date $r_j$ was randomly generated from a uniform distribution $U[0, 3n]$. Without loss of generality, at least one job was released at time zero. Processing time $p_{i,j}$ was randomly generated from a uniform distribution $U[1, 10]$. For operation $O_{i,j}$, processing time $p_{i,j} = 0$ indicates that job $j$ does not pass through machine $i$. The processing routes of each job were randomly generated, and the job might not pass through all machines. For each machine-job combination, 10 random trials were conducted and the average values are reported in the following tables.

6.1. Performance of SPT-DS Heuristic

This section designs numerical experiments to verify the performance of the SPT-DS heuristic algorithm for large-scale instances. The proposed algorithms were implemented in C++ language, and simulations were run on a Dell computer with an Intel Core i5-8400 (2.8 GHz × 6) CPU and 8.00 GB RAM. The input data were generated because no standard benchmark was available for the JSP problem with release dates. Release date $r_j$ was randomly generated from a uniform distribution $U[0, 3n]$. Without loss of generality, at least one job was released at time zero. Processing time $p_{i,j}$ was randomly generated from a uniform distribution $U[1, 10]$. For operation $O_{i,j}$, processing time $p_{i,j} = 0$ indicates that job $j$ does not pass through machine $i$. The processing routes of each job were randomly generated, and the job might not pass through all machines. For each machine-job combination, 10 random trials were conducted and the average values are reported in the following tables.

6.1. Performance of SPT-DS Heuristic

This section designs numerical experiments to verify the performance of the SPT-DS heuristic algorithm for large-scale instances. The machine-job combinations $m = [3, 5, 8]$ with $n = [100, 300, 500]$ were tested with the previous input settings. Measure $\text{GAP} = \frac{Z^H - Z^L}{Z^L}$ was used for evaluating the error of algorithm, where $Z^H$ and $Z^L$ are the objective values of the SPT-DS heuristic and lower bound, respectively. Mean GAP values are presented in the following table.

In Table 6, the GAPs are stable for the constant machine number. For the five-machine example, the GAP values float between 1.0664 and 1.2134 as the problem scale increases from 100 to 500,
where the fluctuation range is about 7.5%. Furthermore, the growth trend of GAP values indicates that the SPT-DS heuristic deteriorates with the increase in \( m \) for a fixed problem scale. For the 300-job example, the GAP value increases from 0.7349 to 1.4221 as the machine scale increases from 3 to 8. This phenomenon might be attributed to the large machine scale increasing the idle time between adjacent operations in a feasible schedule, thereby weakening the performance of the heuristic.

### Table 6. GAPs of SPT-DA heuristic.

| \( n/m \) | \( m = 3 \)     | \( m = 5 \)    | \( m = 8 \)    |
|----------|----------------|----------------|----------------|
| \( n = 100 \) | 0.7808         | 1.0644         | 1.4948         |
| \( n = 300 \) | 0.7349         | 1.2143         | 1.4221         |
| \( n = 500 \) | 0.7217         | 1.1150         | 1.4003         |

6.2. Improvement of the HDDE Algorithm

To highlight the improvement schemes of the tri-insertion and local search, comparative experiments were executed between the standard DDE (SDDE) and the HDDE algorithms. The machine-job combinations \( m = \{3, 5, 8\} \) with \( n = \{50, 100, 150\} \) were tested under the input settings presented at the beginning of this section. The performance of a meta-heuristic mainly depends on the parameter settings. Thus, a series of orthogonal experiments was conducted to determine the parameters of the algorithm (Appendix A), such as mutation factor \( Z \), crossover factor \( Y \), population size \( \Lambda \), and local search probability \( \theta \) (Table A2). The parameters are set as \( Z = 0.2, Y = 0.1, \Lambda = 200, \tau_{\max} = 300, \) and \( \theta = 0.8 \).

Relative difference percentage \( RDP = \frac{Z^H - Z^*}{Z^*} \times 100\% \) was used for evaluation of the effectiveness of the HDDE algorithm, where \( Z^H \) is the final objective value obtained by a meta-heuristic for each test, and \( Z^* \) is the minimum value among all \( Z^H \) values.

Table 7 shows the average RDP (ARDP), minimum RDP (MinRDP), maximum RDP (MaxRDP), and standard deviation (SD) for SDDE and HDDE. ARDP is a performance measure that evaluates an algorithm numerically. The ARDPs of HDDE evidently dominate those of SDDE. For example, the mean values of ARDPs obtained by HDDE and SDDE are 0.6011 and 10.1811, respectively. SD aims to reveal the stability performance of an algorithm. The SD values of HDDE and SDDE are stable in [0, 1.21] and [2.91, 4.34], respectively, except for the maximum values, indicating the robustness of the former relative to the latter. In the minimum objective values of the 90 numerical tests, about 94% (85 out of 90) of the data were obtained by HDDE and about 6% of the data (5 out of 90) were obtained by SDDE. These final experimental outcomes show that the improvement schemes considerably enhance the performance of the HDDE algorithm.

### Table 7. Improvement effect of the HDDE algorithm.

| \( m \times n \) | SDDE | HDDE |
|-----------------|------|------|
|                 | ARDP | MaxRDP | MinRDP | SD  | ARDP | MaxRDP | MinRDP | SD  |
| 3 x 50          | 26.11| 55.36  | 16.42  | 14.98| 4.81 | 48.13  | 0      | 14.43|     |
| 3 x 100         | 14.64| 20.39  | 9.35   | 3.91 | 0    | 0      | 0      | 0    |     |
| 3 x 150         | 7.38 | 11.75  | 0      | 3.73 | 0.40 | 4.04   | 0      | 1.21 |     |
| 5 x 50          | 13.82| 23.55  | 6.59   | 4.34 | 0    | 0      | 0      | 0    |     |
| 5 x 100         | 7.81 | 15.51  | 2.36   | 4.03 | 0    | 0      | 0      | 0    |     |
| 5 x 150         | 5.82 | 12.29  | 0      | 3.44 | 0.06 | 0.55   | 0      | 0.17 |     |
| 8 x 50          | 4.78 | 10.8   | 0      | 3.66 | 0.05 | 0.48   | 0      | 0.14 |     |
| 8 x 100         | 7.32 | 13.5   | 3.53   | 2.91 | 0    | 0      | 0      | 0    |     |
| 8 x 150         | 3.95 | 11.38  | 0      | 3.34 | 0.09 | 0.91   | 0      | 0.27 |     |
6.3. Comparison between HDDE and Other Optimization Algorithms

6.3.1. Comparison between HDDE and ACO

Ant Colony Optimization (ACO) algorithm is selected as a contrast objective to indicate the superiority of the HDDE algorithm. The machine-job combinations and input parameters are identical with those in Section 6.2. The RDP measure is used in this section. The parameters used in ACO algorithm are presented as follows. \( P_{ij}^k \) is the probability of selecting operation \( j \) when ant \( k \) climbs to position \( i \) (i.e., the moment when the current operation is selected).

\[
P_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{s \in \Omega_k} [\tau_{is}]^\alpha \cdot [\eta_{is}]^\beta}, & j \notin \Omega_k \\ 0, & j \in \Omega_k \end{cases}
\]  

(14)

where \( \Omega_k \) is the tabu list for ant \( k \), which stores operations that have been processed. \( \tau_{ij} \) is the pheromone concentration of operation \( j \). \( \eta_{ij} \) is the current heuristic factor of operation \( j \) at present.

\[
\eta_{ij} = \frac{1}{d_{ij}}
\]  

(15)

where \( d_{ij} \) is the processing time of operation \( j \). Parameters \( \alpha \) and \( \beta \) indicate the relative importance of pheromones and heuristics.

\[
\Delta \tau_{ij}^k = \begin{cases} \frac{Q}{C_k}, & \text{if operation } j \text{ is machined in position } i \\ 0, & \text{others} \end{cases}
\]  

(16)

where \( Q \) is a constant, and \( C_k \) is the objective values of the operations in which ant \( k \) has been processed.

\[
\Delta \tau_{ij} = \sum_{k=1}^{\Lambda} \Delta \tau_{ij}^k
\]  

(17)

\( \Delta \tau_{ij} \) is the total increment of the pheromone of operation \( j \).

\[
\tau_{ij} = \rho \cdot \tau_{ij} + \Delta \tau_{ij}
\]  

(18)

\( \tau_{ij} \) is the current pheromone concentration of operation \( j \), \( \Delta \tau_{ij} \) is the pheromone concentration of operation \( j \) in the previous iteration stage, and \( \rho \) is the evaporation coefficient.

After previous preparation, a basic framework of the ACO algorithm is presented as follows.

Step 1. Set parameters for the algorithm, such as the ant colony size \( \Lambda \), the maximum number of iterations \( t_{\text{max}} \), constant \( Q \), evaporation coefficient \( \rho \), and parameters \( \alpha \) and \( \beta \). The initial pheromone concentration of each operation is set to a certain constant. Set the current number of iterations \( t = 0 \).

Step 2. Generate the initial colony. With Equation (14), each ant uses roulette method to continuously select the operations that need to be processed. If all operations are included in the sequence of each ant, the initial ant colony is formed. Use Equations (16)–(18) to update the pheromone of each operation.

Step 3. Set \( h = 1 \), and start iteration.

Step 4. The current ant uses the roulette method to select an operation with Equation (14). Once an operation is selected, the pheromone increment of the operation is obtained by Equation (16).

Step 5. If \( h < \Lambda \), set \( h = h + 1 \), and then go to Step 4; otherwise go to Step 6.

Step 6. If \( t < t_{\text{max}} \), set \( t = t + 1 \). The pheromone increment of each operation is obtained by Equation (16). Update the pheromone of each operation in current iteration with Equation (17), and then go to step 3; otherwise go to step 7.
Step 7. Terminate procedure if the termination condition is satisfied. The optimal individual of the current ant colony is output as the final solution.

Similarly, the parameters of ant colony algorithm are set by orthogonal experiment, as shown in Appendix B. The parameters are set as $\alpha = 1.0, \beta = 0.5, \rho = 0.9, Q = 0.5, \Lambda = 200$ and $t_{max} = 300$. Table 8 shows the average RDP (ARDP), minimum RDP (MinRDP), maximum RDP (MaxRDP), and standard deviation (SD) for ACO and HDDE. The ARDPs of HDDE evidently dominate those of ACO. For example, the mean values of ARDPs obtained by HDDE and ACO are 0.3789 and 22.4422, respectively. The SD values of HDDE and ACO are stable in [0, 1.78] and in [3.12, 9.69], respectively, except for the maximum values, indicating the robustness of the former relative to the latter. In the minimum objective values of the 90 numerical tests, about 92% (83 out of 90) of the data were obtained by HDDE and about 8% of the data (7 out of 90) were obtained by ACO. These numerical results reveal that the HDDE algorithm completely dominates the ACO algorithm.

| $m \times n$ | ACO | | | | HDDE | | | |
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|
|              | ARDP | MaxRDP | MinRDP | SD | ARDP | MaxRDP | MinRDP | SD |
| 3 x 50       | 27.54 | 50.31 | 14.36 | 12.88 | 0 | 0 | 0 | 0 |
| 3 x 100      | 9.07  | 22.34 | 0     | 7.48 | 0.81 | 5.70 | 0 | 1.78 |
| 3 x 150      | 2.67  | 10.76 | 0     | 3.75 | 1.89 | 12.43 | 0 | 3.72 |
| 5 x 50       | 13.62 | 26.73 | 0     | 9.69 | 0.43 | 4.32 | 0 | 1.29 |
| 5 x 100      | 41.12 | 55.66 | 33.50 | 6.00 | 0 | 0 | 0 | 0 |
| 5 x 150      | 37.16 | 43.71 | 32.24 | 3.35 | 0 | 0 | 0 | 0 |
| 8 x 50       | 5.76  | 16.99 | 0     | 6.53 | 0.28 | 2.84 | 0 | 0.85 |
| 8 x 100      | 34.16 | 39.27 | 28.12 | 3.51 | 0 | 0 | 0 | 0 |
| 8 x 150      | 30.88 | 35.04 | 25.83 | 3.12 | 0 | 0 | 0 | 0 |

6.3.2. Comparison between HDDE and PSO

To highlight the dominance of the HDDE algorithm, a series of comparison experiments are conducted between the HDDE and PSO algorithms. The concrete procedure of the PSO algorithm is referred to [30,31]. The parameters of the HDDE algorithm are identical with those in Section 6.3.1. The parameters of the PSO algorithm are set by orthogonal experiment (shown in Appendix C), i.e., population size $\Lambda = 200$, maximum number of iterations $t_{max} = 300$, weight $w = 5$, maximum speed $V_{max} = 4$, maximum position $S_{max} = 4$, cognitive coefficient $c_1 = 3$, and social coefficient $c_2 = 3$.

The data in Table 9 are the values of ARDP, MinRDP, MaxRDP, and SD for PSO and HDDE. The results of HDDE are all 0 for the 90 trials. This phenomenon indicates that the HDDE algorithm completely dominates the PSO algorithm.

| $m \times n$ | PSO | | | | HDDE | | | |
|-------------|-----|-----|-----|-----|-----|-----|-----|-----|
|              | ARDP | MaxRDP | MinRDP | SD | ARDP | MaxRDP | MinRDP | SD |
| 3 x 50       | 45.96 | 79.70 | 19.87 | 19.89 | 0 | 0 | 0 | 0 |
| 3 x 100      | 44.23 | 65.99 | 23.06 | 14.33 | 0 | 0 | 0 | 0 |
| 3 x 150      | 30.60 | 48.25 | 13.48 | 9.878 | 0 | 0 | 0 | 0 |
| 5 x 50       | 62.99 | 83.20 | 44.53 | 12.414 | 0 | 0 | 0 | 0 |
| 5 x 100      | 48.66 | 55.63 | 40.24 | 5.29 | 0 | 0 | 0 | 0 |
| 5 x 150      | 42.89 | 43.71 | 32.24 | 3.35 | 0 | 0 | 0 | 0 |
| 8 x 50       | 51.02 | 63.66 | 41.41 | 7.13 | 0 | 0 | 0 | 0 |
| 8 x 100      | 41.63 | 53.69 | 33.94 | 5.52 | 0 | 0 | 0 | 0 |
| 8 x 150      | 37.70 | 43.38 | 32.99 | 3.34 | 0 | 0 | 0 | 0 |

6.3.3. Comparison between HDDE and GA

This section conducted comparison experiments on the HDDE and GA algorithms. The experimental parameters of the GA algorithm are set as follows: mutation probability $Z_1 = 0.5,$
crossover probability $Y_1 = 0.5$, population size $\Lambda = 200$, and maximum number of iterations $T_{\text{max}} = 300$. The detailed parameter setting process is presented in Appendix D.

The data of ARDP, MinRDP, MaxRDP, and SD for GA and HDDE are shown in Table 10. It is obvious that HDDE’s ARDPs is superior to GA’s ARDPs. For example, the mean values of ARDPs obtained by HDDE and GA are 0.158 and 23.769, respectively. The SD values of GA are stable in $[5.70, 12.91]$, except for the maximum values, and the SD value of HDDE is only equal to 3.04 at one scale ($3 \times 150$), but equal to zero at all other scales, indicating the robustness of the latter relative to the former. In the minimum objective values of the 90 numerical tests, about 97.8% (88 out of 90) of the data were obtained by HDDE and about 2.2% of the data (2 out of 90) were obtained by GA. These numerical results reveal that the HDDE algorithm completely dominates the GA algorithm.

### Table 10. Comparison between HDDE and GA.

| $m \times n$ | GA          | HDDE        |
|-------------|-------------|-------------|
|             | ARDP MaxRDP MinRDP SD | ARDP MaxRDP MinRDP SD |
| $3 \times 50$ | 29.09 62.36 9.19 16.67 | 0 0 0 0 |
| $3 \times 100$ | 20.07 46.35 0.58 12.69 | 0 0 0 0 |
| $3 \times 150$ | 10.34 30.00 0 9.04 | 1.42 8.18 0 3.04 |
| $5 \times 50$ | 36.05 59.27 18.34 12.91 | 0 0 0 0 |
| $5 \times 100$ | 18.02 33.99 0.99 10.70 | 0 0 0 0 |
| $5 \times 150$ | 14.92 25.37 4.26 6.73 | 0 0 0 0 |
| $8 \times 50$ | 23.23 32.33 16.81 5.70 | 0 0 0 0 |
| $8 \times 100$ | 30.18 51.12 13.14 11.54 | 0 0 0 0 |
| $8 \times 150$ | 32.02 46 16.94 9.24 | 0 0 0 0 |

### 6.4. Comparison under JSS Problem Benchmarks

Currently, no standard benchmark is provided for the JSS problem with release dates. Therefore, the basic benchmarks proposed by Taillard, E. [32] is combined with generated data of release dates from a uniform distribution $[0, 3\tau]$ to verify the performance of the proposed algorithm.

To control the meta-heuristics running in an appropriate time, the data of processing times with machine-job combinations $m \times n = 15 \times 50$, $20 \times 50$, and $20 \times 100$ were selected from the benchmarks. The data in Table 11 were the final objective values of the HDDE, ACO, PSO, and GA algorithms, where $\text{gap}_1 = (Z_{\text{ACO}} - Z_{\text{HDDE}})/Z_{\text{HDDE}} \times 100\%$, $\text{gap}_2 = (Z_{\text{PSO}} - Z_{\text{HDDE}})/Z_{\text{HDDE}} \times 100\%$, $\text{gap}_3 = (Z_{\text{GA}} - Z_{\text{HDDE}})/Z_{\text{HDDE}} \times 100\%$, $Z_{\text{HDDE}}$, $Z_{\text{ACO}}$, $Z_{\text{PSO}}$, and $Z_{\text{GA}}$ are the final objective values obtained by HDDE, ACO, PSO, and GA, respectively. It is obvious that the HDDE algorithm is superior to other population-based meta-heuristics.

### Table 11. Comparison under JSS problem benchmarks.

| $m \times n$ | HDDE $10^8$ | ACO $10^8$ | PSO $10^8$ | GA $10^8$ | $\text{gap}_1$ | $\text{gap}_2$ | $\text{gap}_3$ |
|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| $15 \times 50$ | $5.34 \times 10^8$ | $7.88 \times 10^8$ | $9.18 \times 10^8$ | $1.28 \times 10^8$ | 47.68 | 71.92 | 139.70 |
|              | $5.44 \times 10^8$ | $7.87 \times 10^8$ | $9.35 \times 10^8$ | $1.27 \times 10^8$ | 44.52 | 71.84 | 132.34 |
|              | $5.12 \times 10^8$ | $7.54 \times 10^8$ | $8.34 \times 10^8$ | $1.45 \times 10^8$ | 47.17 | 62.95 | 182.17 |
|              | $5.15 \times 10^8$ | $7.99 \times 10^8$ | $8.90 \times 10^8$ | $1.15 \times 10^8$ | 55.23 | 73.03 | 123.65 |
|              | $5.15 \times 10^8$ | $7.92 \times 10^8$ | $8.84 \times 10^8$ | $1.39 \times 10^8$ | 53.94 | 71.84 | 170.79 |
|              | $5.54 \times 10^8$ | $8.36 \times 10^8$ | $9.35 \times 10^8$ | $1.17 \times 10^8$ | 50.98 | 68.90 | 111.56 |
|              | $5.32 \times 10^8$ | $8.22 \times 10^8$ | $8.91 \times 10^8$ | $1.51 \times 10^8$ | 54.50 | 67.38 | 183.35 |
|              | $5.71 \times 10^8$ | $8.40 \times 10^8$ | $9.94 \times 10^8$ | $1.38 \times 10^8$ | 47.06 | 74.12 | 142.28 |
|              | $5.16 \times 10^8$ | $7.43 \times 10^8$ | $8.86 \times 10^8$ | $1.28 \times 10^8$ | 44.02 | 71.71 | 147.24 |
|              | $5.50 \times 10^8$ | $8.13 \times 10^8$ | $8.67 \times 10^8$ | $1.48 \times 10^8$ | 47.87 | 57.99 | 168.68 |
A high-end non-dominated sorting-based meta-heuristic will be presented to obtain Pareto optimal solutions.

Author Contributions: Supervision, T.R.; methodology—designing algorithms, Y.Z. and T.R.; formal analysis, B.-y.C., X.-y.W. and P.Z.; conceptualization—problem modeling, S.-R.C.; writing—original draft preparation, M.Z.; writing—review and editing, C.-C.W. All authors have read and agreed to the published version of the manuscript.

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

However, the tested meta-heuristics consumed different CPU times. For an \( m \times n = 20 \times 100 \) instance, the CPU time expended by the HDDE, ACO, PSO, and GA algorithms are 17,890, 22,055, 837.853, and 238.448 s, respectively. The results reveals that better performance of an algorithm will cost more CPU time. As a trade-off between solution quality and running time, the HDDE algorithm dominates the other meta-heuristics.

7. Conclusions

This study investigates the JSS problem to minimize the total quadratic completion time where each job is available at its release date. No polynomial algorithm can solve the problem because of its NP-hardness. For large-sized instances, the SPT-DS heuristic algorithm is proposed to achieve approximate solutions in a short computing time. For medium-sized instances, the HDDE algorithm is provided to achieve high-quality solutions, where well-designed tri-insertion crossover and lower search schemes significantly enhance the performance of the meta-heuristic. A series of random experiments demonstrates the effectiveness of the proposed algorithms.

Future researches mainly focus on two aspects. On the one hand, the acceleration scheme will be proposed to save running time for the HDDE algorithm. On the other hand, the model will be generalized to a multi-objective version that is more popular in a production scheduling environment. A non-dominated sorting-based meta-heuristic will be presented to obtain Pareto optimal solutions. A high-efficiency improvement scheme, such as variable neighborhood search, will be combined with the meta-heuristic for JSS problems.

### Table 11. Cont.

| \( m \times n \) | HDDE       | ACO       | PSO       | GA        | gap\(_1\) | gap\(_2\) | gap\(_3\) |
|----------------|------------|-----------|-----------|-----------|-----------|-----------|-----------|
| 20 \times 50   | 7.56 \times 10^8 | 1.05 \times 10^9 | 1.18 \times 10^9 | 2.01 \times 10^9 | 39.26 | 55.67 | 166.45 |
|                | 7.81 \times 10^8 | 1.16 \times 10^9 | 1.24 \times 10^9 | 2.31 \times 10^9 | 47.97 | 59.22 | 195.26 |
|                | 7.08 \times 10^8 | 1.02 \times 10^9 | 1.13 \times 10^9 | 1.79 \times 10^9 | 43.55 | 59.82 | 152.24 |
|                | 6.49 \times 10^8 | 9.59 \times 10^9 | 1.10 \times 10^9 | 1.77 \times 10^9 | 47.66 | 69.11 | 172.34 |
|                | 7.01 \times 10^8 | 1.03 \times 10^9 | 1.25 \times 10^9 | 2.04 \times 10^9 | 46.63 | 78.74 | 190.4 |
|                | 7.16 \times 10^8 | 1.08 \times 10^9 | 1.23 \times 10^9 | 2.07 \times 10^9 | 50.28 | 71.38 | 189.73 |
|                | 6.92 \times 10^8 | 1.04 \times 10^9 | 1.20 \times 10^9 | 2.20 \times 10^9 | 50.31 | 72.82 | 217.56 |
|                | 6.91 \times 10^8 | 9.99 \times 10^8 | 1.17 \times 10^9 | 1.82 \times 10^9 | 44.70 | 69.10 | 162.85 |
|                | 7.21 \times 10^8 | 1.11 \times 10^9 | 1.19 \times 10^9 | 2.29 \times 10^9 | 53.57 | 65.19 | 218.07 |
|                | 7.57 \times 10^8 | 1.08 \times 10^9 | 1.20 \times 10^9 | 2.08 \times 10^9 | 42.33 | 58.00 | 174.95 |

| \( m \times n \) | HDDE       | ACO       | PSO       | GA        | gap\(_1\) | gap\(_2\) | gap\(_3\) |
|----------------|------------|-----------|-----------|-----------|-----------|-----------|-----------|
| 20 \times 100  | 4.68 \times 10^9 | 6.86 \times 10^9 | 7.54 \times 10^9 | 1.20 \times 10^{10} | 46.60 | 61.20 | 156.75 |
|                | 4.31 \times 10^9 | 6.14 \times 10^9 | 6.89 \times 10^9 | 1.55 \times 10^{10} | 42.60 | 59.89 | 260.32 |
|                | 4.67 \times 10^9 | 6.62 \times 10^9 | 7.35 \times 10^9 | 1.32 \times 10^{10} | 41.79 | 57.41 | 182.91 |
|                | 4.49 \times 10^9 | 6.44 \times 10^9 | 6.99 \times 10^9 | 1.57 \times 10^{10} | 43.37 | 55.8 | 250.04 |
|                | 4.58 \times 10^9 | 6.56 \times 10^9 | 7.19 \times 10^9 | 1.53 \times 10^{10} | 43.01 | 56.74 | 233.26 |
|                | 4.51 \times 10^9 | 6.41 \times 10^9 | 6.82 \times 10^9 | 1.60 \times 10^{10} | 42.13 | 51.11 | 254.59 |
|                | 4.71 \times 10^9 | 6.20 \times 10^9 | 6.94 \times 10^9 | 1.66 \times 10^{10} | 31.57 | 47.45 | 252.63 |
|                | 4.44 \times 10^9 | 6.40 \times 10^9 | 6.86 \times 10^9 | 1.64 \times 10^{10} | 43.91 | 54.35 | 268.87 |
|                | 4.62 \times 10^9 | 6.16 \times 10^9 | 6.88 \times 10^9 | 1.55 \times 10^{10} | 33.14 | 48.78 | 235.07 |
|                | 4.42 \times 10^9 | 6.23 \times 10^9 | 6.97 \times 10^9 | 1.40 \times 10^{10} | 41.19 | 57.91 | 217.75 |
Appendix A

Orthogonal experiments are designed to determine the appropriate parameters of the DDE algorithm, including population size $A$, mutation factor $Z$, crossover factor $Y$, maximum iteration $\tau_{\text{max}}$ and local search probability $\theta$, because the performance of the DDE algorithm is mainly dependent on parameter setting. The DDE algorithm parameters table is shown in Table A1.

To save computing time, the test scale is fixed at $m = 3$ and $n = 60$. The mean improvement percent $\text{MIP} = \frac{Z^{\text{FIN}} - Z^{\text{INI}}}{Z^{\text{INI}}} \times 100\%$ is used to measure the effect of the parameters, where $Z^{\text{INI}}$ is the minimum objective value among the initial population, and $Z^{\text{FIN}}$ is the final objective value of the output solution at the termination of algorithm. The orthogonal experiment results for the parameters of the HDDE algorithm are shown in Table A2. The main effect graph of the mean value for the orthogonal experiment results is shown in Figure A1. Referring to the peak point of each parameter (2-1-1-5-2) in Figure A1, it is obvious that the best parameter combinations are $Z = 0.2$, $Y = 0.1$, $\Lambda = 200$, $\tau_{\text{max}} = 300$ and $\theta = 0.8$.

![Figure A1. The main effect graph of HDDE.](image)

| Level | $Z$ | $Y$ | $\Lambda$ | $\tau_{\text{max}}$ | $\theta$ |
|-------|-----|-----|-----------|---------------------|--------|
| 1     | 0.1 | 0.1 | 200       | 100                 | 0.75   |
| 2     | 0.2 | 0.2 | 300       | 150                 | 0.8    |
| 3     | 0.3 | 0.3 | 400       | 200                 | 0.85   |
| 4     | 0.4 | 0.4 | 500       | 250                 | 0.9    |
| 5     | 0.5 | 0.5 | 600       | 300                 | 0.95   |

| No. | $Z$ | $Y$ | $\Lambda$ | $\tau_{\text{max}}$ | $\theta$ | $\text{MIP}$ |
|-----|-----|-----|-----------|---------------------|--------|-------------|
| 1   | 1   | 1   | 1         | 1                   | 1      | 63.047      |
| 2   | 1   | 2   | 2         | 2                   | 2      | 56.857      |
| 3   | 1   | 3   | 3         | 3                   | 3      | 62.44       |
| 4   | 1   | 4   | 4         | 4                   | 4      | 61.173      |
| 5   | 1   | 5   | 5         | 5                   | 5      | 56.198      |
| 6   | 2   | 1   | 2         | 3                   | 4      | 68.973      |
| 7   | 2   | 2   | 3         | 4                   | 5      | 63.566      |
| 8   | 2   | 3   | 4         | 5                   | 1      | 62.135      |
| 9   | 2   | 4   | 5         | 1                   | 2      | 51.484      |
Table A2. Cont.

| No. | Z | Y | Α | τ_{max} | θ | MIP   |
|-----|---|---|----|---------|---|-------|
| 10  | 2 | 5 | 1  | 2       | 3 | 63.784|
| 11  | 3 | 1 | 3  | 5       | 2 | 76.275|
| 12  | 3 | 2 | 4  | 1       | 3 | 48.861|
| 13  | 3 | 3 | 5  | 2       | 4 | 47.98 |
| 14  | 3 | 4 | 1  | 3       | 5 | 57.46 |
| 15  | 3 | 5 | 2  | 4       | 1 | 56.328|
| 16  | 4 | 1 | 4  | 2       | 5 | 65.379|
| 17  | 4 | 2 | 5  | 3       | 1 | 60.104|
| 18  | 4 | 3 | 1  | 4       | 2 | 57.64 |
| 19  | 4 | 4 | 2  | 5       | 3 | 55.235|
| 20  | 4 | 5 | 3  | 1       | 4 | 44.366|
| 21  | 5 | 1 | 5  | 4       | 3 | 58.799|
| 22  | 5 | 2 | 1  | 5       | 4 | 64.755|
| 23  | 5 | 3 | 2  | 1       | 5 | 40.386|
| 24  | 5 | 4 | 3  | 2       | 1 | 42.218|
| 25  | 5 | 5 | 4  | 3       | 2 | 48.76 |

Appendix B

Similarly, orthogonal experiments are designed to determine the appropriate parameters of the ACO algorithm, including ant colony size Λ, pheromone quantity Q, decay coefficient ρ, maximum iteration τ_{max}, accumulated information factor α, and inspiration information factor β. The ACO algorithm parameters level table is shown in Table A3.

To save computing time, the test scale is fixed as m = 3 and n = 60. MIP is used to measure the effect of the parameters. The orthogonal experiment results for the parameters of the ACO algorithm are shown in Table A4. The main effect graph of mean values for the orthogonal experiment results is shown in Figure A2. Referring to the peak point of each parameter (5-4-1-5-1-5) in Figure A2, it is obvious that the best parameter combinations are α = 1.0, β = 0.5, ρ = 0.9, Q = 0.5, Λ = 200, and τ_{max} = 300.

Table A3. ACO algorithm parameters level.

| Level | α | β | ρ | Q | Λ | τ_{max} |
|-------|---|---|---|---|---|---------|
| 1     | 0.6| 0.2| 0.9| 0.9| 200| 100     |
| 2     | 0.7| 0.3| 1  | 0.8| 300| 150     |
| 3     | 0.8| 0.4| 0.8| 0.7| 400| 200     |
| 4     | 0.9| 0.5| 0.6| 0.6| 500| 250     |
| 5     | 1  | 0.6| 0.7| 0.5| 600| 300     |

Table A4. ACO algorithm parameters orthogonal experiment results.

| Level | α | β | ρ | Q | Λ | τ_{max} | MIP   |
|-------|---|---|---|---|---|---------|-------|
| 1     | 1 | 1 | 1 | 1 | 1 | 1       | 19.133|
| 2     | 1 | 2 | 2 | 2 | 2 | 2       | 17.13 |
| 3     | 1 | 3 | 3 | 3 | 3 | 3       | 21.664|
| 4     | 1 | 4 | 4 | 4 | 4 | 4       | 25.079|
| 5     | 1 | 5 | 5 | 5 | 5 | 5       | 24.833|
| 6     | 2 | 1 | 2 | 3 | 4 | 5       | 12.356|
| 7     | 2 | 2 | 3 | 4 | 5 | 1       | 17.048|
| 8     | 2 | 3 | 4 | 5 | 1 | 2       | 22.346|
| 9     | 2 | 4 | 5 | 1 | 2 | 3       | 22.964|
Table A4. Cont.

| Level | α | β | ρ | Q | Λ | $t_{\text{max}}$ | MIP  |
|-------|---|---|---|---|---|----------------|------|
| 10    | 2 | 5 | 1 | 2 | 3 | 4              | 19.987 |
| 11    | 3 | 1 | 3 | 5 | 2 | 4              | 15.748 |
| 12    | 3 | 2 | 4 | 1 | 3 | 5              | 17.635 |
| 13    | 3 | 3 | 5 | 2 | 4 | 1              | 21.003 |
| 14    | 3 | 4 | 1 | 3 | 5 | 2              | 20.2  |
| 15    | 3 | 5 | 2 | 4 | 1 | 3              | 24.685 |
| 16    | 4 | 1 | 4 | 2 | 5 | 3              | 14.161 |
| 17    | 4 | 2 | 5 | 3 | 1 | 4              | 22.272 |
| 18    | 4 | 3 | 1 | 4 | 2 | 5              | 29.234 |
| 19    | 4 | 4 | 2 | 5 | 3 | 1              | 23.501 |
| 20    | 4 | 5 | 3 | 1 | 4 | 2              | 22.543 |
| 21    | 5 | 1 | 5 | 4 | 3 | 2              | 11.02 |
| 22    | 5 | 2 | 1 | 5 | 4 | 3              | 23.61 |
| 23    | 5 | 3 | 2 | 1 | 5 | 4              | 26.278 |
| 24    | 5 | 4 | 3 | 2 | 1 | 5              | 30.529 |
| 25    | 5 | 5 | 4 | 3 | 2 | 1              | 25.445 |

Figure A2. The main effect graph of ACO.

Appendix C

The parameters of the PSO algorithm mainly include population size $\Lambda$, maximum iteration $\tau_{\text{max}}$, weights $w$, maximum speed $V_{\text{max}}$, maximum position $S_{\text{max}}$, cognitive coefficient $c_1$, and social coefficient $c_2$. To unify comparison criterion, the population size and maximum iteration of PSO are set to the same as those of HDDE. The remaining parameters of PSO are set by orthogonal experiments. The PSO algorithm parameters level table is shown in Table A5.

The test scale is $m \times n = 3 \times 60$, and test procedure is similar with that in Appendix A. The results for the parameters of the PSO algorithm are shown in Table A6. The main effect graph of the mean values for the orthogonal experiment results is shown in Figure A3. Referring to the peak point of each parameter (1-2-3-1-3) in Figure A3, it is obvious that the best parameter combinations are $\Lambda = 200$, $\tau_{\text{max}} = 300$, $w = 5$, $V_{\text{max}} = 4$, $S_{\text{max}} = 4$, $c_1 = 3$, $c_2 = 3$.

Table A5. PSO algorithm parameters level.

| Level | $\omega$ | $v_{\text{max}}$ | $s_{\text{max}}$ | $c_1$ | $c_2$ |
|-------|----------|-----------------|-----------------|-------|-------|
| 1     | 0.9      | 5               | 6               | 3     | 1     |
| 2     | 1.0      | 4               | 5               | 2.5   | 2     |
| 3     | 1.1      | 3               | 4               | 2     | 3     |
| 4     | 1.2      | 2               | 3               | 1.5   | 2.5   |
| 5     | 1.3      | 1               | 2               | 1     | 1.5   |
Table A6. PSO algorithm parameters orthogonal experiment results.

| NO. | \(\omega\) | \(v_{\text{max}}\) | \(s_{\text{max}}\) | \(c1\) | \(c2\) | MIP   |
|-----|-----------|----------------|----------------|------|------|-------|
| 1   | 1         | 1             | 1             | 1    | 1    | 75.467|
| 2   | 1         | 2             | 2             | 2    | 2    | 80.659|
| 3   | 1         | 3             | 3             | 3    | 3    | 87.806|
| 4   | 1         | 4             | 4             | 4    | 4    | 44.144|
| 5   | 1         | 5             | 5             | 5    | 5    | 43.162|
| 6   | 2         | 1             | 2             | 3    | 4    | 68.973|
| 7   | 2         | 2             | 3             | 4    | 5    | 64.049|
| 8   | 2         | 3             | 4             | 5    | 1    | 46.286|
| 9   | 2         | 4             | 5             | 1    | 2    | 68.278|
| 10  | 2         | 5             | 1             | 2    | 3    | 74.803|
| 11  | 3         | 1             | 3             | 5    | 2    | 60.749|
| 12  | 3         | 2             | 4             | 1    | 3    | 88.783|
| 13  | 3         | 3             | 5             | 2    | 4    | 58.687|
| 14  | 3         | 4             | 1             | 3    | 5    | 44.56 |
| 15  | 3         | 5             | 2             | 4    | 1    | 65.843|
| 16  | 4         | 1             | 4             | 2    | 5    | 40.564|
| 17  | 4         | 2             | 5             | 3    | 1    | 60.727|
| 18  | 4         | 3             | 1             | 4    | 2    | 58.335|
| 19  | 4         | 4             | 2             | 5    | 3    | 48.848|
| 20  | 4         | 5             | 3             | 1    | 4    | 48.898|
| 21  | 5         | 1             | 5             | 4    | 3    | 58.814|
| 22  | 5         | 2             | 1             | 5    | 4    | 44.572|
| 23  | 5         | 3             | 2             | 1    | 5    | 36.048|
| 24  | 5         | 4             | 3             | 2    | 1    | 57.198|
| 25  | 5         | 5             | 4             | 3    | 2    | 49.239|

Figure A3. The main effect graph of PSO.

Appendix D

The parameters of GA mainly include mutation probability \(Z_1\), crossover probability \(Y_1\), population size \(\Lambda\), and maximum iteration \(\tau_{\text{max}}\). Similarly, an orthogonal experiment with four factors and five levels was performed to determine parameters. The parameters level table of GA is shown in Table A7. The results for the parameters of GA are shown in Table A8. The main effect graph of mean values for the orthogonal experiment results is shown in Table A4. Referring to the peak point of each parameter (5-5-1-5) in Figure A4, it is obvious that the best parameter combinations are \(Z_1 = 0.5\), \(Y_1 = 0.5\), \(\Lambda = 200\), \(\tau_{\text{max}} = 300\).
Appendix D

The parameters of GA mainly include mutation probability $Z_1$, crossover probability $Y_1$, population size $\Lambda$, and maximum iteration $\tau_{\text{max}}$. Similarly, an orthogonal experiment with four factors and five levels was performed to determine parameters. The parameter $s$ level table of GA is shown in Table A7. The results for the parameters of GA are shown in Table A8. The main effect graph of mean values for the orthogonal experiment results is shown in figure D1. Referring to the peak point of each parameter (5-5-1-5) in Figure A4, it is obvious that the best parameter combinations are $Z_1 = 0.5$, $Y_1 = 0.5$, $\Lambda = 200$, $\tau_{\text{max}} = 300$.

Figure A4. The main effect graph of GA.

Table A7. GA parameters level.

| Level | $Z_1$ | $Y_1$ | $\Lambda$ | $\tau_{\text{max}}$ |
|-------|-------|-------|-----------|-------------------|
| 1     | 0.1   | 0.1   | 200       | 100               |
| 2     | 0.2   | 0.2   | 300       | 150               |
| 3     | 0.3   | 0.3   | 400       | 200               |
| 4     | 0.4   | 0.4   | 500       | 250               |
| 5     | 0.5   | 0.5   | 600       | 300               |

Table A8. GA parameters orthogonal experiment results.

| NO. | $Z_1$ | $Y_1$ | $\Lambda$ | $\tau_{\text{max}}$ | $MIP$ |
|-----|-------|-------|-----------|---------------------|-------|
| 1   | 1     | 1     | 1         | 1                   | 15.438|
| 2   | 1     | 2     | 2         | 2                   | 19.518|
| 3   | 1     | 3     | 3         | 3                   | 28.87 |
| 4   | 1     | 4     | 4         | 4                   | 24.818|
| 5   | 1     | 5     | 5         | 5                   | 29.768|
| 6   | 2     | 1     | 2         | 3                   | 22.523|
| 7   | 2     | 2     | 3         | 4                   | 28.46 |
| 8   | 2     | 3     | 4         | 5                   | 28.97 |
| 9   | 2     | 4     | 5         | 1                   | 20.75 |
| 10  | 2     | 5     | 1         | 2                   | 26.568|
| 11  | 3     | 1     | 3         | 5                   | 32.46 |
| 12  | 3     | 2     | 4         | 1                   | 24.465|
| 13  | 3     | 3     | 5         | 2                   | 22.413|
| 14  | 3     | 4     | 1         | 3                   | 29.891|
| 15  | 3     | 5     | 2         | 4                   | 35.587|
| 16  | 4     | 1     | 4         | 2                   | 22.361|
| 17  | 4     | 2     | 5         | 3                   | 27.872|
| 18  | 4     | 3     | 1         | 4                   | 36.13 |
| 19  | 4     | 4     | 2         | 5                   | 30.854|
| 20  | 4     | 5     | 3         | 1                   | 21.569|
| 21  | 5     | 1     | 5         | 4                   | 24.833|
| 22  | 5     | 2     | 1         | 5                   | 38.551|
| 23  | 5     | 3     | 2         | 1                   | 27.793|
| 24  | 5     | 4     | 3         | 2                   | 33.15 |
| 25  | 5     | 5     | 4         | 3                   | 31.272|
References

1. Akers, S.B.; Friedman, J. A non-numerical approach to production scheduling problems. *J. Oper. Res. Soc. Am.* 1955, 3, 429–442. [CrossRef]

2. Jain, A.; Meieran, S. Deterministic job-shop scheduling: Past, present and future. *Eur. J. Oper. Res.* 1999, 113, 390–434. [CrossRef]

3. Zhang, J.; Ding, G.; Zou, Y.; Qin, S.; Fu, J. Review of job shop scheduling research and its new perspectives under Industry 4.0. *J. Intell. Manuf.* 2019, 30, 1809–1830. [CrossRef]

4. Cheng, T.C.E.; Liu, Z. Parallel machine scheduling to minimize the sum of quadratic completion times. *IEEE Trans.* 2004, 36, 11–17. [CrossRef]

5. Garey, M.R.; Johnson, D.S.; Sethi, R. The Complexity of Flowshop and Jobshop Scheduling. *Math. Oper. Res.* 1976, 1, 117–129. [CrossRef]

6. Chen, B.; Potts, C.N.; Woeginger, G.J. A Review of Machine Scheduling: Complexity, Algorithms and Approximability. In *Handbook of Combinatorial Optimization*; Du, D.-Z., Pardalos, P., Eds.; Kluwer Academic Publishers: London, UK, 1998; pp. 21–169.

7. Brucker, P. *Scheduling Algorithms*, 5th ed.; Springer: Berlin/Heidelberg, Germany, 2007.

8. Khadwilard, A.; Chansombat, S.; Thepphakorn, T.; Thapatsuwan, P.; Chainate, W.; Pongcharoen, P. Application of firefly algorithm and its parameter setting for job shop scheduling. *J. Ind. Technol.* 2012, 8, 49–58.

9. Gao, H.; Kwong, S.; Fan, B.; Wang, R. A hybrid particle-swarm tabu search algorithm for solving job shop scheduling problems. *IEEE Trans. Ind. Inform.* 2014, 10, 2044–2054. [CrossRef]

10. Qiu, X.; Lau, H. An AIS-based hybrid algorithm for static job shop scheduling problem. *J. Intell. Manuf.* 2014, 25, 489–503. [CrossRef]

11. Wang, X.; Duan, H. A hybrid biogeography-based optimization algorithm for job shop scheduling problem. *Comput. Ind. Eng.* 2014, 73, 96–114. [CrossRef]

12. Keesari, H.S.; Rao, R.V. Optimization of job shop scheduling problems using teaching-learning-based optimization algorithm. *OPSEARCH* 2014, 51, 545–561. [CrossRef]

13. Asadzadeh, L. A local search genetic algorithm for the job shop scheduling problem with intelligent agents. *Comput. Ind. Eng.* 2015, 85, 376–383. [CrossRef]

14. Peng, B.; Lü, Z.; Cheng, T.C.E. A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Comput. Oper. Res.* 2015, 53, 154–164. [CrossRef]

15. Kurdi, M. A new hybrid island model genetic algorithm for job shop scheduling problem. *Comput. Ind. Eng.* 2015, 88, 273–283. [CrossRef]

16. Cheng, T.C.E.; Peng, B.; Lü, Z. A hybrid evolutionary algorithm to solve the job shop scheduling problem. *Ann. Oper. Res.* 2016, 242, 223–237. [CrossRef]

17. Dau, T.K.; Pan, T.S.; Nguyen, T.T.; Pan, J.S. Parallel bat algorithm for optimizing makespan in job shop scheduling problems. *J. Intell. Manuf.* 2018, 29, 451–462. [CrossRef]

18. Saidi-Mehrabad, M.; Dehnavi-Arani, S.; Evazabadian, F.; Mahmoodian, V. An Ant Colony Algorithm (ACA) for solving the new integrated model of job shop scheduling and conflict-free routing of AGVs. *Comput. Ind. Eng.* 2015, 86, 2–13. [CrossRef]

19. Sundar, S.; Suganthan, P.N.; Jin, C.T.; Xiang, C.T.; Soon, C.C. A hybrid artificial bee colony algorithm for the job-shop scheduling problem with no-wait constraint. *Soft Comput.* 2017, 5, 1193–1202. [CrossRef]

20. Kuhpfahl, J.; Bierwirth, C. A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective. *Comput. Oper. Res.* 2016, 66, 44–57. [CrossRef]

21. Kundakci, N.; Kulak, O. Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem. *Comput. Ind. Eng.* 2016, 96, 31–51. [CrossRef]

22. Ku, W.Y.; Beck, J.C. Mixed integer programming models for job shop scheduling: A computational analysis. *Comput. Oper. Res.* 2016, 73, 165–173. [CrossRef]

23. Phandear, R.K.; Jain, A.; Verma, R. A genetic algorithm-based approach for job shop scheduling. *J. Manuf. Technol. Manag.* 2012, 23, 937–946. [CrossRef]

24. Nguyen, S.; Zhang, M.; Johnston, M.; Tan, K.C. Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Trans. Evol. Comput.* 2013, 18, 193–208. [CrossRef]
25. May, G.; Stahl, B.; Taisch, M.; Prabhu, V. Multi-objective genetic algorithm for energy-efficient job shop scheduling. *Int. J. Prod. Res.* 2015, 53, 7071–7089. [CrossRef]

26. 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]

27. Zhang, R.; Chiong, R. Solving the energy-efficient job shop scheduling problem: A multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption. *J. Clean. Prod.* 2016, 112, 3361–3375. [CrossRef]

28. Townsend, W. The single machine problem with quadratic penalty function of completion times: A branch-and-bound solution. *Manag. Sci.* 1978, 24, 530–534. [CrossRef]

29. Bai, D. Asymptotic analysis of online algorithms and improved scheme for the flow shop scheduling problem with release dates. *Int. J. Syst. Sci.* 2015, 46, 1994–2005. [CrossRef]

30. Ehsan, N.; Mahdi, P.K.; Matti, L. Transmission expansion planning integrated with wind farms: A review, comparative study, and a novel profound search approach. *Int. J. Electr. Power Energy Syst.* 2020, 115, 105460.

31. Ehsan, N.; Mahdi, P.K.; Hamdi, A. An efficient particle swarm optimization algorithm to solve optimal power flow problem integrated with FACTS devices. *Appl. Soft Comput. J.* 2019, 80, 243–262.

32. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* 1993, 64, 278–285. [CrossRef]