symmetry

MDPI

Article

A Matheuristic Approach for the No-Wait Flowshop Scheduling Problem with Makespan Criterion

Yu Gao [1,2], Ziyue Wang [1,*], Liang Gao [1] and Xinyu Li [1]

[1] School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China; jefflouisa@hust.edu.cn (Y.G.); gaoliang@mail.hust.edu.cn (L.G.); lixinyu@mail.hust.edu.cn (X.L.)
[2] School of Physics and Electronic Science, Hubei Normal University, Huangshi 435002, China
* Correspondence: wangziyue0829@126.com or wangziyue0829@hust.edu.cn

Abstract: The No-wait Flowshop Scheduling Problem (NWFSP) has always been a research hotspot because of its importance in various industries. This paper uses a matheuristic approach that combines exact and heuristic algorithms to solve it with the objective to minimize the makespan. Firstly, according to the symmetry characteristics in NWFSP, a local search method is designed, where the first job and the last job in the symmetrical position remain unchanged, and then, a three-level neighborhood division method and the corresponding rapid evaluation method at each level are given. The two proposed heuristic algorithms are built on them, which can effectively avoid al-ready searched areas, so as to quickly obtain the local optimal solutions, and even directly obtain the optimal solutions for small-scale instances. Secondly, using the equivalence of this problem and the Asymmetric Traveling Salesman Problem (ATSP), an exact method for solving NWFSP is constructed. Importing the results of the heuristics into the model, the efficiency of the Mil-ler-Tucker-Zemlin (MTZ) model for solving small-scale NWFSP can be improved. Thirdly, the matheuristic algorithm is used to test 141 instances of the Tailard and Reeves benchmarks, and each optimal solution can be obtained within 134 s, which verifies the stability and effectiveness of the algorithm.

Keywords: heuristic; matheuristic; neighborhood search; no-wait

Citation: Gao, Y.; Wang, Z.; Gao, L.; Li, X. A Matheuristic Approach for the No-Wait Flowshop Scheduling Problem with Makespan Criterion. *Symmetry* **2022**, *14*, 913. https://doi.org/10.3390/sym14050913

Academic Editor: Mihai Postolache

Received: 28 March 2022
Accepted: 26 April 2022
Published: 29 April 2022

1. Introduction

Improving production efficiency is one of the final aims of the manufacturing industry. Production scheduling can meet that need, so it is used in manufacturing systems extensively. Considering the differences in factory conditions and job processing requirements, the scheduling problem has been divided into several kinds, such as flow shop [1,2], hybrid flow shop [3,4], job shop [5,6], open shop [7,8], and so on. The flow shop problem (FSP) claims that each job should be processed in the same order through every stage, and the hybrid flow shop refers to parallel machines at one or more stages. A job shop environment permits jobs to pass some steps or repeat them at a given machine route, and the processing order of the job in an open shop can be arbitrary. It can be concluded that the flow shop problem is the basic one, then the others can be seen as its variants. In other words, the FSP reflects some common characteristics of production scheduling problems.

In recent years, flow shop scheduling problems with constraints have become research hotspots owing to the actual technology demand. Typically, no-wait attribute, which stipulates all the jobs should be processed from the first stage to the last without any interruption, fits lots of production industries. For instance, in steel processing, maintaining the temperature is a necessary condition, so it calls for no-wait demand. Once you wait, you need more time and energy to heat to the desired temperature, or accept degradation. As early as 1971, Callahan [9] considered that the steel industry required a dependent processing and the no-wait constraint. Tang and Song [10] modeled the mill roll annealing process as a no-wait hybrid flow shop scheduling problem. Höhn and Jacobs [11] also

considered continuous casting in the multistage production process a variant of no-wait flowshop scheduling. Yuan and Jing [12] explained and affirmed the importance of the no-wait flowshop problem to steel manufacturing. Similarly, Reklaits [13] pointed out that the no-wait constraint is also suitable for the chemical industry due to its instability. Hall [14] emphasized that a similar situation also occurs in plastic molding and silver-ware industries, as well as the food processing industry [15]. In addition, there are also many other practical problems that can be solved by the assistance of being modeled as no-wait scheduling problems, such as controlling health care management costs [16], developing a metro train real-time control system [17], and so on.

In this paper, the NWFSP with makespan criterion is considered because of its universality. It minimizes the maximum completion time of the jobs ($C_{\max}$), maximizing the efficiency of machine usage [18]. The problem can be described as $F_{\mathrm{m}}|nwt|C_{\max}$ when the symbolic notation proposed by Graham [19] is applied. The first field represents the production environment, which is a flow shop with m stages. The second field shows the constraints and "nwt" is for "no-wait constraint". The performance criteria occupy the third field.

The research on NWFSP began around the 1970s. Garey and Johnson [20] proved that the NWFSP is a strongly NP-hard problem when the number of machines is no less than three. At the same time, the problem was documented as NP-hard for more than two machines by Papadimitriou and Kanellakis [21]. Due to the complexity of NWFSP and the wide range of its applications, many heuristic algorithms have emerged or been tried by Bonney and Gundry [22], King and Spachis [23], Gangadharan and Rajendran [24], Laha [25], and so on, which aimed to make the solution as close as possible to the optimal within an acceptable time. In addition, exact methods [26] like branch-and-bound [27] and the column generation method [28] are widely used to obtain optimal solutions for small-scale problems. Metaheuristic algorithms, whose search mechanisms are inspired by phenomena in various fields of the real world are also active for solving NWFSP. Hall [14] and Allahverdi [29] jointly detailed the research progress of NWFSP before 2016. A typical example is the discrete particle swarm optimization algorithm [30]. In recent years, various improved or emerging metaheuristic algorithms have been used to solve the NWFSP. For example, a hybrid ant colony optimization algorithm (HACO [31]), a discrete water wave optimization algorithm (DWWO [32]), a factorial based particle swarm optimization with a population adaptation mechanism (FPAPSO [33]), a single water wave optimization (SWWO [34]), a hybrid biogeography-based optimization with variable neighborhood search (HBV [35]), a quantum-inspired cuckoo co-search (QCCS [36]) algorithm, and an improved discrete migrating birds optimization algorithm (IDMBO [37]) have also been used with good results. In 2021, Lai used the Discrete Wolf Pack Algorithm [38] to solve the NWFSP problem. Each algorithm has its own advantages, which makes it perform better than others in certain applications. In order to give full play to the advantages of each algorithm and enhance the generality of the algorithm, hyper-heuristic (HH) [39] and matheuristic [40] algorithms have appeared. The emergence of a new generation of large-scale mathematical programming optimizers such as CPLEX [41] and Gurobi [42] has greatly improved the performance of matheuristic algorithms that combine heuristic algorithms and exact algorithms. In recent years, matheuristics [43] have been widely used in scheduling problems, including NWFSP. Since NWFSP can be transformed into the ATSP [44], it is effective to incorporate more mathematical computations into its search mechanism.

Existing NWFSP researches mostly focus on designing algorithms from the perspective of instances rather than problems. The heuristic rules or the component mechanisms and parameter selections of the metaheuristics are determined by the quality of the test results, so as to balance the local search and the global search. In this way, the algorithm is dependent on the instances, and the performance of the algorithm will be different for different instances. This paper aims to explore the heuristic rules from the perspective of the problem, so that the rules show consistent effects on the instances, thereby enhancing

the stability of the algorithm. Existing neighborhood structures for local search focus on insertion, swap, destruction-construction, etc., and do not divide the already searched area and the unsearched area, so the same sequence cannot avoid being tested repeatedly. Likewise, good structures are not necessarily preserved after multiple iterations. In this paper, by expanding the search domain gradually, the search process is kept in the unsearched potential areas, and the repeated calculation is reduced. The experimental results in Section 4.1 show that such a search method can also effectively preserve good structures. Exact algorithms are limited by the complexity and scale of the problem, so they are less applicable than heuristics and metaheuristics. Using heuristic rules to guide the search direction of the exact algorithm can improve the solving efficiency and increase the application scope of the exact algorithm. The experiments in Section 4.2 demonstrate that by fully exploiting the advantages of both methods, better results can be obtained than the current mainstream metaheuristic algorithms for solving NWFSP problems.

The rest of this article is organized as follows. Section 2 describes the problem of NWFSP. Section 3 proposes two heuristic algorithms for improving the solutions of NWFSP. Section 4 discusses the performance of the proposed matheuristic algorithm on two benchmarks. Section 5 summarizes the contributions of this article and provides some suggestions for future research directions.

2. No-Wait Flow Shop Scheduling Problem

Compared to FSP, NWFSP permits no interruption when a job is processed from the first stage to the last one. The details are expressed as follows.

Typically, the NWFSP includes m stages, preparing for n jobs to follow the same given route, where the m and n are determined. It also restricts only one machine at each stage, so it contains m machines overall. Furthermore, the processing time of job j on machine i is also known and may be named as $P(i, j)$. It is natural to consider every $P(i, j)$ as the element of row i, column j in processing time matrix P. One job should be processed on only one machine at the same time and it is merely allowed to be transferred to the next stage until the current work is finished; similarly, each machine cannot deal with more than one job at any time. Here, one only considers the situation of ignoring the setup time and transfer time.

2.1. Problem Description

As we known, makespan, the complete time of the last job in a sequence, is one of the indicators that directly measures economic benefits. Let $(C_{\max})_{\min}$ denote the minimum makespan and its expression is shown in Equation (1). The meanings of related symbols are as follows.

The job set: $J = \{1, 2, 3, L, n | n \in Z, n \geq 1\}$
The machine set: $M = \{1, 2, 3, L, m | m \in Z, m \geq 1\}$
π any sequence
Π the set of all possible sequence π
π_k the kth job in the sequence $k \in Z$ and $k \geq 1$
C_{ij} end time of job j on machine i

$$(C_{\max})_{\min} = \min_{\pi \in \Pi}(C_{m,\pi_n}) \tag{1}$$

$DY(j_1, j_2)$ is the delay of the job j_2 when it is arranged after job j_1, $\forall j_1, j_2 \in J,\ j_1 \neq j_2$.
PR_j the tail value of the job j.

$$PR_j = \sum_{i=2}^{m} P(i, j), \forall j \in J, i \in M \tag{2}$$

$P1$ the sum of the processing times of all the jobs on the first machine

$$P1 = \sum_{j=1}^{n} P(1,j) \tag{3}$$

Owing to the no-wait constraint, the delay value between two adjacent jobs is constant and can be calculated by matrix P. As shown in Figure 1, $C_{\max}$ has been divided into three parts and can be calculated from Equation (4). Obviously, $P1$ is a constant, so the change of $C_{\max}$ is only affected by two parts. Then, the simplified objective function was formulated in Equation (5).

$$C_{\max} = \sum_{j=1}^{n-1} DY(\pi_j, \pi_{j+1}) + PR_{\pi_n} + P1 \tag{4}$$

$$C'_{\max} = \sum_{j=1}^{n-1} DY(\pi_j, \pi_{j+1}) + PR_{\pi_n} \tag{5}$$

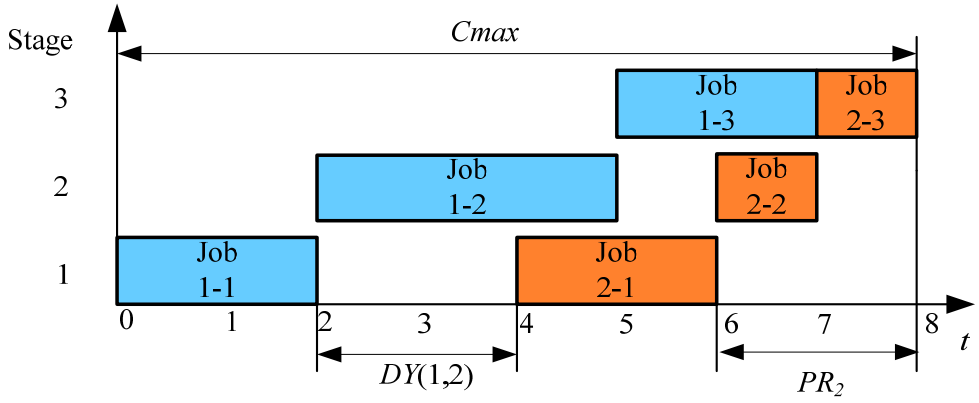

Figure 1. The composition chart of *Cmax*.

2.2. Equivalent Model

Considering jobs as cities and the delay value as the distance between two cities, the NWFSP can be transformed into the ATSP. After adding virtual city/job "0"and "$n+1$", the model can be described as follows. For convenience, "0" and "$n+1$" are counted as the same city, and "0" represents the starting city/job, and "$n+1$" represents the ending city/job. The city/job set is $JC = \{0, 1, 2, 3, L, n+1 | n \in Z, n \geq 1\}$.

$$DY(0, j_2) = P1,\ j_2 \in [1, n] \tag{6}$$

$$DY(j_1, n+1) = PR_{j_1},\ j_1 \in [1, n] \tag{7}$$

The objective:

$$Minimize\left(\sum_{j_2=1}^{n} DY(0, j_2) x_{0,j_2} + \sum_{j_1=1}^{n} \sum_{j_2=1, j_1 \neq j_2}^{n+1} DY(j_1, j_2) x_{j_1,j_2}\right) \tag{8}$$

Subjected to:

$$\sum_{j_2=1}^{n} x_{0,j_2} = 1 \tag{9}$$

$$\sum_{j_2=1, j_2 \neq j_1}^{n+1} x_{j_1,j_2} = 1, \forall j_1 \in [1, n] \tag{10}$$

$$\sum_{j_1=0, j_1 \neq j_2}^{n} x_{j_1,j_2} = 1, \forall j_2 \in [1,n] \tag{11}$$

$$\sum_{j_1=1}^{n} x_{j_1,n+1} = 1 \tag{12}$$

$$\sum_{\substack{j_1, j_2 \in S,\ j_1 \neq j_2 \\ if\ j_1 = 0,\ then\ j_2 \neq n+1 \\ j_1 \in [0,n],\ j_2 \in [1,n+1]}} x_{j_1,j_2} \leq |S| - 1, 2 \leq |S| \leq n, S \subset JC \tag{13}$$

$$x_{j_1,j_2} = \begin{cases} 1 & if\ j_2\ is\ the\ next\ job/city\ of\ j_1 \\ 0 & else \end{cases} \tag{14}$$

3. The Proposed Heuristic Algorithms

The NWFSP can be transformed into the ATSP, so the path-exchange-strategy used in the Lin-Kernighan–Helsgaun (LKH) [45] algorithm is also suitable for solving NWFSP. Based on this idea, we can divide the neighborhood and construct heuristic algorithms.

3.1. Basic Neighborhood Structures

Considering the search efficiency fully, the search solution domain is initially limited to three levels, that is, Neighborhood $N2$, $N3$, $N4$. The characteristics of the three basic neighborhoods are given in Table 1. The $n-exchange$ optimum means that the solution cannot be better by exchanging n areas, which are arbitrarily divided in the sequence.

Table 1. Basic Neighborhood Features.

Name	Total Area Lines	Total Areas	Conditions for Jumping Out of the Neighborhood
N2	2	3	Being the three-exchange optimum
N3	3	4	Being the four-exchange optimum
N4	4	5	Being the five-exchange optimum

The $N2$ neighborhood is shown in Figure 2 as an example, where $L1$ and $L2$ are the two area lines and the three regions are $Aera0$, $Aera1$, $Aera2$.

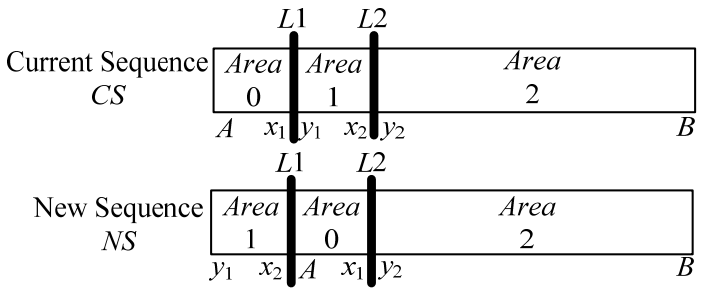

Figure 2. An example diagram of the evolution {102} in the *N2* neighborhood.

3.2. Selected Potential Sub-Neighborhoods

In order to improve the search efficiency and advance the search process in more potential sub-domains, three rules are proposed and three sub-domains are selected, namely $Z2$, $Z3$, and $Z4$. The details are shown in Table 2. Two heuristic algorithms will be carried out along these three neighborhoods.

Table 2. Neighborhood classification.

No	Name	Level	Combinations to Be Tested for Each Sequence	Constant Job
1	*Z*2	A	{102}	Last job
2	*Z*3	B	{2103}	Last job
3	*Z*4	C	{03214}	First and Last job

- Skipping consecutive area rule

Considering that the current solution sequence is almost derived from the last region-exchange operation, the combinations containing the adjacent area number pair (that is "01", "12", "23", "34") as a part do not temporarily need to test again.

- First and last job unchanged rules

From the experimental results, along the evolutionary direction, the job in last position is easier to have coincidence with the optimal solution because of being affected by *PR* values. Then is the first position. When the evolution process has advanced to a certain level, the first and last job can be fixed.

- Low-Level-neighborhood first searched rule

The lower the neighborhood level, the higher the search priority. The algorithm flow chart is shown in Figure 3. "US" is the abbreviation of "Update successfully" and "UF" is the abbreviation of "Update failed". For ease of representation, the structure in the dashed box in Figure 3 is referred to as "PHM".

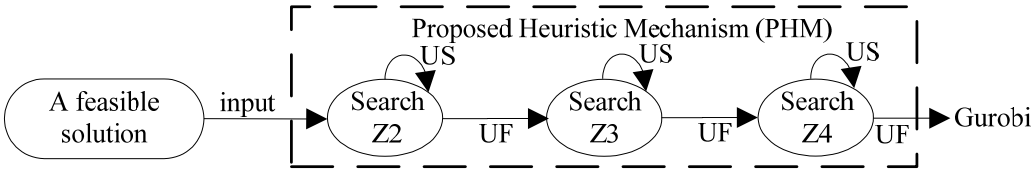

Figure 3. The simplified search mechanism contains three neighborhoods.

3.3. Two Heuristic Algorithms

Based on the mechanism in Figure 3, it is easy to propose two heuristic algorithms according to whether the *PR* value characteristic is highlighted. The two heuristic algorithms HG1 and HG2 differ only in the first step, that is, the input feasible solutions are different. The details are shown in Figure 4. The advantage of HG1 is randomness. HG2 always takes the job with the smallest *PR* value as the final job to start the search, which is more in line with the characteristics of NWFSP and can speed up the algorithm convergence.

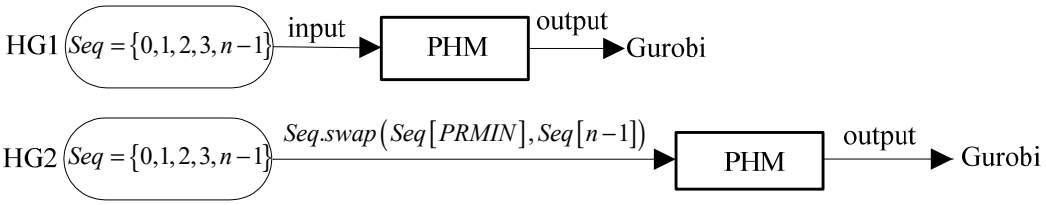

Figure 4. Schematic diagram of the HG1 and HG2.

In order to improve the utilization of historical data and improve the efficiency of neighborhood search. According to the different characteristics of the neighborhoods of *Z*2, *Z*3 and *Z*4, corresponding search algorithms are provided one by one. They are vmcmp2, vmcmp3, and vmcmp4.

3.3.1. Local Evolutionary Algorithm in Z2

Searching the only manner {102} can improve evolution speed. Without the change of PR, the evaluation speed is further improved. Let the size of Δ characterize the degree of improvement of the new sequence; the larger the value, the better the improvement.

$$\Delta = C'_{\max}(CS) - C'_{\max}(NS) = DY(x_1, y_1) + DY(x_2, y_2) - DY(x_2, A) - DY(x_1, y_2) \quad (15)$$

In Figure 2, it can be seen that for the same CS, when $L1$ and $L2$ move, x_1, x_2, y_1, y_2 are changed, but A, B remains unchanged, which can be regarded as constants. Because the first job is stored in A, once the adjustment is made according to {102}, the A value of the new sequence must change, and the value of $DY(x_2, A)$ will also change. As can be seen from Equation (15), in order to make full use of historical data and reduce repeated calculating, Δ is regarded as two components. The first part and the second part are respectively represented by functions f and g, as shown in Equations (16) and (17).

$$f(x_2, A) = DY(x_2, A) \quad (16)$$

$$g(x_1, y_2) = DY(x_1, y_1) + DY(x_2, y_2) - DY(x_1, y_2) \quad (17)$$

The two-dimensional array G is constructed, and the calculation result $g(x_1, y_2)$ of Equation (17) is stored in the x_1th row and the y_2th column of G. Such an operation can ensure that after the sequence is updated, the intermediate result G only needs to recalculate part of the data. For the convenience of description, several definitions are given below.

Definition 1 (Point)**.** *Adjacent job pair makes up a point. If* (x_1, y_1) *is a point,* x_1 *is the predecessor job of* y_1*, and* y_1 *is the successor job of* x_1*.*

Definition 2 (Single-point Attribute (SPA))**.** *The function value calculated by a point only or with information of a stable job. For example, function* f *is the single-point attribute of the point* (x_2, y_2)*, where A is the stable job.*

Definition 3 (Double-point Attribute (DPA))**.** *The function value is calculated by two points and reflects the relationship of four related jobs. For example, function g is the double-point attribute of point* (x_1, y_1) *and point* (x_2, y_2)*.*

When traversing $(L1, L2)$, function f is associated with the attributes of job x_2 which belongs to the second point (x_2, y_2). But in essence, x_2 is still a job in CS. If the index number of the first job in the sequence is 0, then the index range of x_2 is $[1, n-2]$ (denote the index of x_2 in the sequence as $L(x_2)$, then $L(x_2) \in [1, n-2]$). Function f is seen as the single-point attributes of the jobs whose index ranges from 1 to $n-2$. Once CS is given, we can directly calculate the function f. It can be seen from the characteristics of the double-point attribute that if the two points do not change, then their DPAs do not change. That is, as long as the $con1$ is met, the saved values in G need not be updated, and the calculation speed can be increased.

The $con1$ is described as follows.

First, the jobs x_1 and y_2 remain in the original pair order, that is, in the sequence NS, and x_1 is still on the left side of y_2.

Second, to find the g value of the job x and job y, that is, in $g(x, y)$, x cannot be equal to x_1 or x_2 in the last operation; y cannot be equal to y_1 or y_2.

The local optimal algorithm (Algorithm 1) for $Z2$ is described as follows.

Algorithm 1. vmcmp2(*con*1)// Search {102} only

Step 1. compute all the *SPA*s in *CS*, that is, all the f
Step 2. compute all the *DPA*s in *CS*, that is, all the g
Step 3. $(\Delta_{\max}, x_1, x_2) = (g - f)_{\max}$ //Save the best positions to x_1 and x_2 which provide the $\Delta_{\max}$
Step 4. if $(\Delta_{\max} > 0)$
    $L1 = x_1$ and $L2 = x_2$
    CS=*CS* ({102})//Perform operation {102} on *CS* to update *CS*
    update (f); //Recalculate all the f values
    update (g, *con*1); // Recalculate g values that do not meet *con*1
    jump to Step 3
  end if
  else// *CS* is the best of *Z*2 now
    Stop searching in *Z*2
  end else

3.3.2. Local Evolutionary Algorithm in Z3

Since the first job and the last job are constant, the point attribute can be used with the historical data to optimize the algorithm. Figure 5 shows the evolution form of "0213", from which Equation (20) can be obtained.

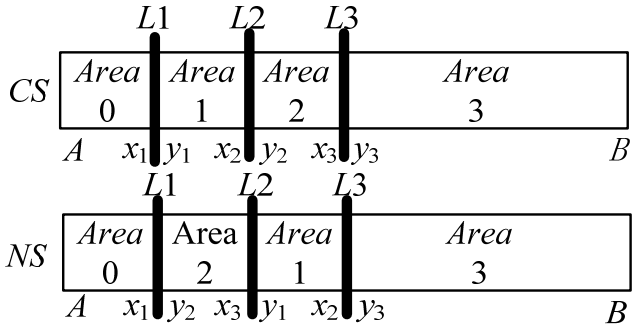

Figure 5. The evolution graph in *Z*3.

Where $SD(j_1, j_2)$ represents the sum of the delay values of all the jobs from the job j_1 to the job j_2 in the current sequence. Equation (20) is the sum of the expressions in the three brackets, and the three sub-expressions, which can be seen as *DPA*s are in a rotation-symmetric relationship. Parentheses 1 and 2 can share the function f_1. Parenthesis 3 is denoted as function f_2.

$$C'_{\max}(CS) = SD(A, x_1) + DY(x_1, y_1) + SD(y_1, x_2) + DY(x_2, y_2) + SD(y_2, x_3) + DY(x_3, y_3) + SD(y_3, B) + PR(B) \quad (18)$$

$$C'_{\max}(NS) = SD(A, x_1) + DY(x_1, y_2) + SD(y_2, x_3) + DY(x_3, y_1) + SD(y_1, x_2) + DY(x_2, y_3) + SD(y_3, B) + PR(B) \quad (19)$$

$$\Delta(0213) = C'_{\max}(CS) - C'_{\max}(NS) = [DY(x_1, y_1) - DY(x_1, y_2)] + [DY(x_2, y_2) - DY(x_2, y_3)] + [DY(x_3, y_3) - DY(x_3, y_1)] \quad (20)$$

f_1 and f_2 are considered as functions of the job numbers and the function values are saved. See Equations (21) and (22) for details.

$$\begin{cases} f_1(x_a, y_b) = DY(x_a, y_a) - DY(x_a, y_b) \\ L(x_a) \in [0, n-3], L(y_b) \in [2, n-1], and\ L(x_a) + 1 < L(y_b) \end{cases} \quad (21)$$

$$\begin{cases} f_2(x_b, y_a) = DY(x_b, y_b) - DY(x_b, y_a) \\ L(x_b) \in [2, n-2], L(y_a) \in [1, n-3], and\ L(x_b) > L(y_a) - 1 \end{cases} \quad (22)$$

Compute and save each $f_1(x_a, y_b)$ value to the $x_a th$ row, $y_b th$ column of the two-dimensional array $G1$. Assume point1 (x_a, y_a), point2 (x_b, y_b), point3 (x_c, y_c), then Equation (20) can be replaced by Equation (23) to reduce double counting.

$$\begin{cases} \Delta(0213) = f_1(x_a, y_b) + f_1(x_b, y_c) + f_2(x_c, y_a) \\ L(x_a) \in [0, n-4], L(x_b) \in [1, n-3], L(x_c) \in [2, n-2], and\ L(x_a) < L(x_b) < L(x_c) \end{cases} \tag{23}$$

In contrast to the search in $S3$, there is only one combination to try, so compute all $\Delta(0213)$ and choose the biggest improvement. If all $\Delta(0213)$ is computed but only updated at most once, the utilization rate of historical data is too low. Then Multiple Iterations in One step (MIO) is proposed to improve this situation.

Traverse the $Z3$ neighborhood of CS, the specific method is to calculate by Formula 23, and save all the values greater than 0 in $DY0$. In Figure 6, $DY0$ is a multiset that sorts the results according to its Δ values, from largest to smallest, and itRS7 is the iterator of $DY0$.

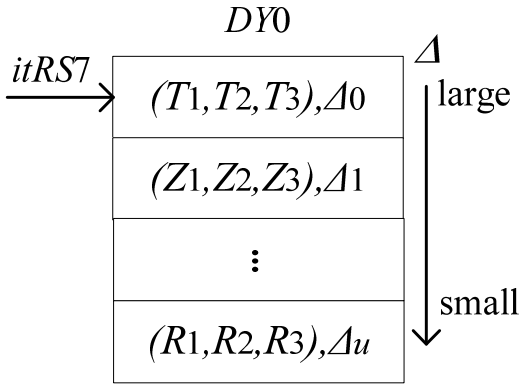

Figure 6. Data storage manner of $DY0$

.

Suppose there is a NWFSP with $n = 12$, and the $DY0$ is shown in Figure 6. As shown in Figure 7, its current solution is $CS^{(0)} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$.

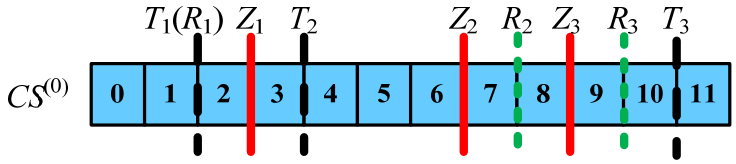

Figure 7. Data characteristics of the 0 iteration of CS.

Take out the element pointed to by the iterator $itRS7$, that is (T_1, T_2, T_3).Then the evolution process can be seen in Figure 8. The indices of points (1, 2), (3, 4), (10, 11) in the sequence are respectively recorded as T_1, T_2, T_3. In Figure 6, (T_1, T_2, T_3) means $T_1 < T_2 < T_3$.

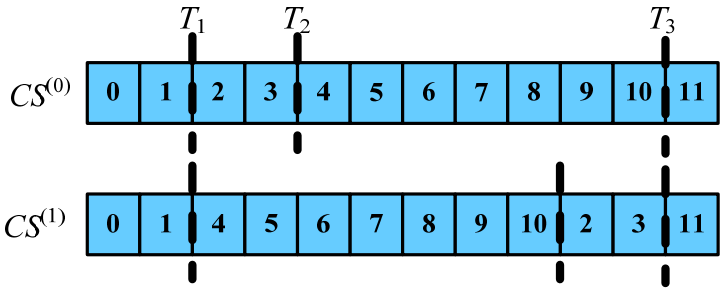

Figure 8. One iteration by the way of (T_1, T_2, T_3).

Because of the constant last job, the PR is constant too. Then the essence of every evolution manner is the change of boundary values. Combining Figures 6 and 7, we can draw the conclusions of Table 3.

Table 3. Changed Paths of Boundary Values.

Evolution Manner	Changed Paths of Boundary Values
(T_1, T_2, T_3)	$DY(1,2) + DY(3,4) + DY(10,11) \rightarrow DY(1,4) + DY(3,11) + DY(10,2)$
(Z_1, Z_2, Z_3)	$DY(2,3) + DY(6,7) + DY(8,9) \rightarrow DY(2,7) + DY(6,9) + DY(8,3)$
(R_1, R_2, R_3)	$DY(1,2) + DY(7,8) + DY(9,10) \rightarrow DY(1,8) + DY(7,10) + DY(9,2)$

It is easy to see from Table 3 that no matter how many iterations are completed, as long as the changed path of boundary values still exist, evolution in this direction will succeed.

It can be seen from Figure 9 that after two iterations, the corresponding point of R_1 cannot be found in $CS^{(2)}$ at this time, so it cannot be evolved in this way. Next, derive the conditions *con*2 whose paths still exist after iterations, as shown in Table 4.

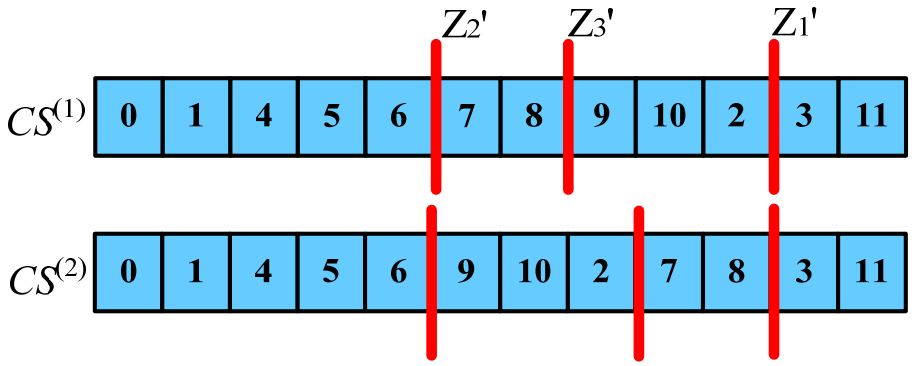

Figure 9. Two iterations by the way of (T_1, T_2, T_3) and (Z_1, Z_2, Z_3).

Table 4. Conditions for *con*2.

Conditions for the Establishment of *con*2
1. Same points constraint. The same 3 points can be found both in sequence before and after the iteration. Suppose their corresponding relationship is: $(Z_1, Z_2, Z_3) \rightarrow (Z_{1'}, Z_{2'}, Z_{3'})$
2. Position constraint. $Z_{1'} < Z_{2'} < Z_{3'}$ or $Z_{2'} < Z_{3'} < Z_{1'}$ or $Z_{3'} < Z_{1'} < Z_{2'}$ (if $Z_1 < Z_2 < Z_3$)

Traverse $DY0$ and determine whether each element meets the *con*2 in turn. If one is satisfied, it is applicable and the iteration can be completed directly. Therefore, $DY0$ is calculated only once, and there is a chance to iterate multiple times. The algorithm (Algorithm 2) vmcmp3(*con*2) suitable for $Z3$ is described as follows.

Algorithm 2. vmcmp3(*con*2)

Step 1. Input $CS^{(0)}$, record Loc (j, $CS^{(0)}$) and save them to vector *k*1, Let its *j*th element, *k*1(*j*) to save the location of job *j* in $CS^{(0)}$
Step 2. *flag* = 0// Set the initial value of the *flag*
Step 3. Compute and save $f1$, $f2$, then compute Δ and save them to multiset *DY*0 when Δ> 0.
Step 4. if (*DY*0.size()! = 0) // if *DY*0 is not empty
for *itRS*7 = *DY*0.begin() to *DY*0.end ()-1)// Traverse *DY*0 by *itRS*7
if(*con*2) //If *con*2 is met
CS= update (*itRS*7, {2013}, *CS*);// According the locations
//provided by *DY*0, give *CS* a {2013} operation
flag=1// Updated successfully
update (Loc (j, *CS*)); // Save the new correspondence of location //and job number
end if(con2)
end for
end if
else
flag=2
end else
if flag==1 // When update successfully
jump to Step2
end if
else // When update failed
Stop searching in Z3
end else

3.3.3. Local Evolutionary Algorithm in Z4

Experiments show that the combination {03214} has a higher hit rate. Its main feature is that the first and last jobs are not changed. The Figure 10 shows the evolution form.

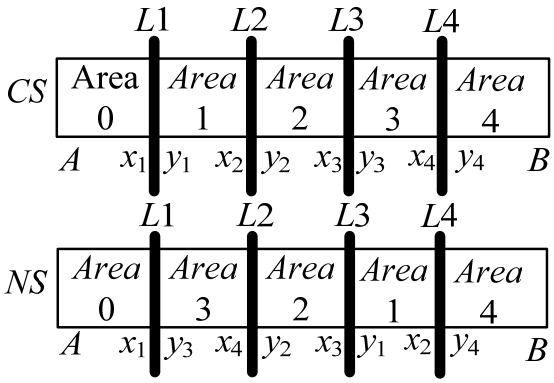

Figure 10. Evolution graph in *Z*4.

Still, start with the expression of $\Delta(03214)$, specific as shown in Equations (24)–(26).

$$C'_{\max}(CS) = SD(A, x_1) + DY(x_1, y_1) + SD(y_1, x_2) + DY(x_2, y_2) + SD(y_2, x_3) + DY(x_3, y_3) + SD(y_3, x_4) + DY(x_4, y_4) + SD(y_4, B) + PR(B) \quad (24)$$

$$C'_{\max}(NS) = SD(A, x_1) + DY(x_1, y_3) + SD(y_3, x_4) + DY(x_4, y_2) + SD(y_2, x_3) + DY(x_3, y_1) + SD(y_1, x_2) + DY(x_2, y_4) + SD(y_4, B) + PR(B) \quad (25)$$

$$\begin{aligned} &\Delta(03214) = C'_{\max}(CS) - C'_{\max}(NS) \\ &= DY(x_1, y_1) + DY(x_2, y_2) + DY(x_3, y_3) + DY(x_4, y_4) - DY(x_1, y_3) - DY(x_4, y_2) - DY(x_3, y_1) - DY(x_2, y_4) \\ &= [DY(x_1, y_1) + DY(x_3, y_3) - DY(x_1, y_3) - DY(x_3, y_1)] + [DY(x_2, y_2) + DY(x_4, y_4) - DY(x_4, y_2) - DY(x_2, y_4)] \end{aligned} \quad (26)$$

Similarly, using Equation (27) to uniformly calculate the two parts separated by parentheses in Equation (26) can simplify the calculation. Let $\Delta(03214)$ be $V(x_a, y_b)$ here.

$$\begin{cases} V(x_a, y_b) = DY(x_a, y_a) + DY(x_b, y_b) - DY(x_a, y_b) - DY(x_b, y_a) \\ L(x_a) \in [0, n-4], L(y_b) \in [3, n-1], and\ L(x_a) < L(y_b) - 1 \end{cases} \quad (27)$$

Compute all the results by Equation (27) and save them in multiset $RS2$.

Use two iterators, $it1$ and $it2$ to traverse $RS2$. Then select all the elements that meet the condition of $con3$, and store them in multiset JSF. See Figures 11 and 12 for details.

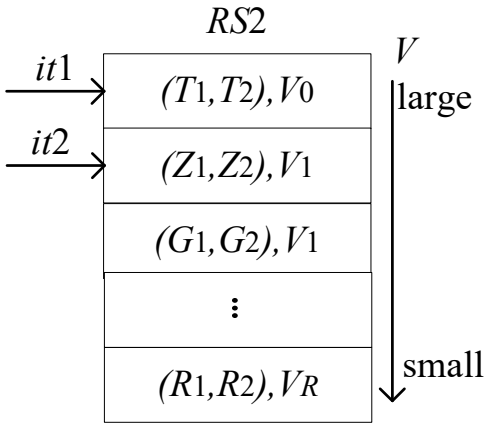

Figure 11. Data storage diagram in RS2.

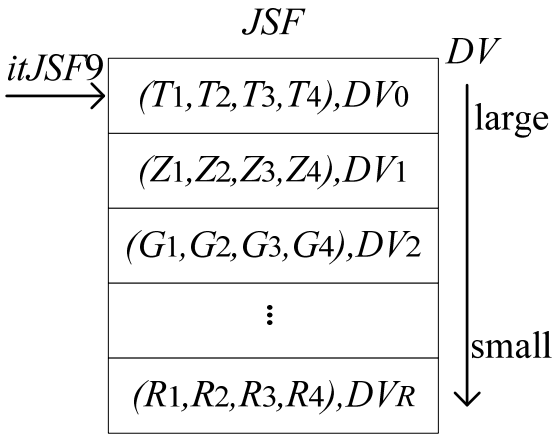

Figure 12. Data storage diagram in JSF.

JSF collects all successful evolution paths. Its iterator is $itRS9$ in Figure 12 ($T_1 < T_2 < T_3 < T_4$). Suppose the variables saved by a single element in $RS2$ are $DOT1$ and $DOT2$, then the conditions for $con3$ to be established are shown in Table 5.

Table 5. Conditions for *con3*.

Conditions for the establishment of *con*3
1. Greater than 0 constraint, that is, $DV > 0$, where DV = *it*1.V + *it*2.V
2. Position constraint.
*it*1.DOT1 < *it*2.DOT1< *it*1.DOT2< *it*2.DOT2 or *it*2.DOT1 < *it*1.DOT1< *it*2.DOT2< *it*1.DOT2

Similarly, use MIO Strategy to construct an algorithm (Algorithm 3) to speed up the search process.

Algorithm 3. vmcmp4(*con*3, *con*4)

Step 1. Input $CS^{(0)}$, record Loc (j, $CS^{(0)}$) and save them to vector *k*1, Let its *j*th element, *k*1(j) to save the location of job j in $CS^{(0)}$
Step 2. *flag* = 0// Set the initial value of the *flag*
Step 3. for x_A = 0 to *n*-4 // The union of the ranges *L*1 and *L*2
for y_B = 3 to $n-1$// The union of the ranges *L*3 and *L*4
cmp*RS*2(x_A,y_B)//compute the *DPA*s of (x_A, y_A, x_B, y_B) and save them in RS2
end for // End the inner loop
end for// End outer loop
Step 4. *it*1begin = *RS*2.begin()
*it*2begin = *it*1 + 1 // Assign initial values to iterators *it*1 and *it*2
for *it*1 = *it*1begin to *it*1end
for *it*2 = *it*2begin to *it*2end // Double loop for traverse *RS*2
if (*con*3(*it*1, *it*2)) //If the elements pointed to by *it*1 and *it*2 satisfy *con*3
JSF. insert (*it*1, *it*2)//Save the evolution path provided
//by the elements which are pointed to by *it*1 and *it*2 in *JSF*
end if
end for// End the inner loop
end for// End outer loop
Step 5. if (*JSF*. size ()≠0)//If JSF is not empty, it means that $CS^{(0)}$ can be
//updated in its neighborhood *Z*4
for *itJSF*9 = *JSF*. begin () to *JSF*. end ()−1 //Traverse multiset *JSF*
//which is the neighborhood Z4 of $CS^{(0)}$
if (*con*4(*itJSF*9))//If the element pointed to by itJSF9 satisfies the con4
CS = update (*JSF*9, {03214}, *CS*)//Update *CS* as specified by *JSF*9
update (Loc (*j*, *CS*)); // Save the new correspondence of
//location and job number
flag = 1; // Assign 1 to the flag bit
end if
end for
end if
else
flag = 2
end else
if *flag* == 1// If there is an update in this round
jump to Step2
end if
else// This round of searching this $CS^{(0)}$ in Z4 has not been updated
Stop searching in *Z*4
end else

Let the condition that satisfies MIO in Z4 be con4. If con4 in the algorithm is to be established, two conditions must be met, as shown in Table 6.

Table 6. Conditions for *con3*.

Conditions for the Establishment of *con*4
1. Same points constraint. The same 4 points can be found both in sequence before and after the iteration. Suppose their corresponding relationship is: (Z_1, Z_2, Z_3, Z_4)→($Z_{1'}$, $Z_{2'}$, $Z_{3'}$, $Z_{4'}$) 2. Position constraint. Suppose (1, 2, 3, 4, *DV*) is taken from *JSF*, then rearrange them from small to large according to their position numbers in the NS. There are 8 orders that satisfies the position constraint, that is {4321, 4123, 3412, 3214, 2341, 2143, 1234, 1432}

3.4. Experimental Results of the Two Heuristic Algorithms

Use HG1 and HG2 to solve Tailard (TA) benchmark. Then record the calculation results in Table 7. The heuristic algorithms were coded in C++. All the experiments executed in Windows 10 on a desktop PC with an 11th Gen Intel(R) Core (TM) i7-1165G7 (2.80GHz) processor and 16.0 GB RAM. The calculation formula of *RPI* (Relative Percentage Increase) and $\overline{RPI}$ are shown in Equations (28) and (29), where $C_{(opt,a1)}$ refers to the optimal $C_{\max}$ of instance a1. $C_{(i1,a1)}$ refers to the result of algorithm $i1$ to solve the instance $a1$.

$$RPI(i1, a1) = \frac{C_{(i1,a1)} - C_{(opt,a1)}}{C_{(opt,a1)}} \times 100 \tag{28}$$

$$\begin{cases} \overline{RPI}|_{from\ b\ \text{to b+9}} = \frac{\sum_{a1=b}^{a1=b+9} RPI(i_1,a1)}{10} \times 100 \\ b \in \{1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101, 111\} \end{cases} \tag{29}$$

In Table 7, HG2 has a smaller $\overline{RPI}$ value than HG1 in all scale instances; its fluctuation range is from 0.8% to 2.58%, the maximum $\overline{RPI}$ value of 2.58% appears on the small-scale instances (scale is 20 × 10), while its $\overline{RPI}$ value on the largest-scale instances (500 × 20) is only 1.28%. On the whole, it performs better for large-scale instances. The average $\overline{RPI}$ is 1.30%, and the fluctuation is not large, indicating that the algorithm has strong local search ability, and can quickly converge to a local optimal solution regardless of the scale, with stable performance.

It can be seen from Table 7 that out of 120 instances, HG1 only exceeds HG2 in 6 instances (the part in bold in the table). Therefore, for the goal of approximating the optimal value, the effect of the HG2 algorithm constructed by introducing the *PR* value is much better than that of HG1, and even HG2 directly obtains the optimum on some small-scale instances (the part in italics in the table). Taking "TA 05" as an example, its Gantt chart is shown in Figure 13. If the result of HG1 is imported into Gurobi, another set of optimal solution sequences can be obtained. This Gantt chart is shown in Figure 14. Therefore, the use of different heuristic algorithms is effective for the breadth search. The same job-blocks in the HG1 sequence and the HG2 sequence can also be considered as high-quality structures by comparing them with the optimal solution sequence. Therefore, the heuristic algorithms can easily construct multiple high-quality solutions. Arrange the jobs according to their *PR* values from small to large, take them as the last job in turn, and then perform the PHM operation.

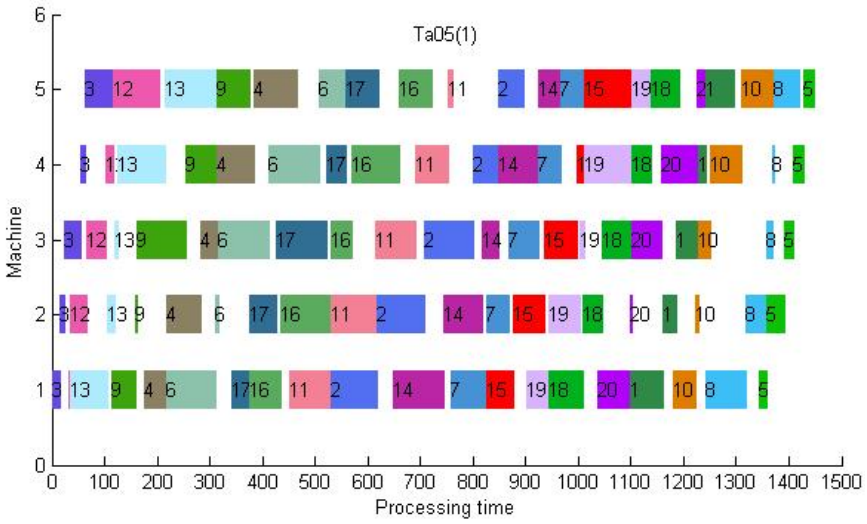

Figure 13. The Gantt chart of TA05 by HG2 (makespan = 1449).

Table 7. Results of HGs on Tailard Benchmark (TA).

TA	HG1		HG2		TA	HG1		HG2		TA	HG1		HG2		TA	HG1		HG2	
	C_{HG1}	$\overline{RPI}$	C_{HG2}	$\overline{RPI}$		C_{HG1}	$\overline{RPI}$	C_{HG2}	$\overline{RPI}$		C_{HG1}	$\overline{RPI}$	C_{HG2}	$\overline{RPI}$		C_{HG1}	$\overline{RPI}$	C_{HG2}	$\overline{RPI}$
01	**1505**		1509		31	3220		3198		61	6488		6453		91	15523		15401	
02	1590		*1528*		32	**3467**		3490		62	6333		6264		92	15326		15164	
03	1586		1462		33	3310		3247		63	6232		6138		93	15536		15352	
04	1634		*1588*		34	3424		3354		64	6148		6055		94	15303		15221	
05	1482	4.10	*1449*	0.8	35	3398	2.23	3392	0.83	65	6274	1.92	6240	0.82	95	15379	2.08	15194	0.92
06	1522		1502		36	3403		3360		66	6203		6099		96	15328		15129	
07	1561		1515		37	3292		3251		67	6326		6267		97	15643		15418	
08	1548		1521		38	3336		3268		68	6210		6144		98	15467		15245	
09	1577		1470		39	3162		3082		69	6487		6412		99	15320		15130	
10	1405		*1377*		40	3407		3324		70	6445		6401		100	15573		15386	
11	2228		2153		41	4448		4283		71	8214		8122		101	**19773**		19840	
12	2248		2230		42	4372		4234		72	8084		7959		102	20331		20179	
13	2051		1987		43	4300		4170		73	8267		8084		103	19998		19986	
14	1945		*1811*		44	**4477**		4513		74	8538		8430		104	20297		20122	
15	2077	5.40	2070	2.58	45	4340	3.40	4340	1.42	75	8049	2.50	8014	1.10	105	20233	2.18	19962	1.61
16	**1925**		1955		46	4390		4330		76	7952		7875		106	20329		20148	
17	2031		1990		47	4653		4496		77	8123		7966		107	20478		20478	
18	2145		2068		48	4555		4361		78	8100		7957		108	20238		20117	
19	2120		1992		49	4270		4255		79	8331		8199		109	20319		20233	
20	2126		2088		50	4381		4359		80	8247		8179		110	20195		20006	
21	3102		3001		51	6260		6210		81	11092		10799		111	47034		46600	
22	3128		2878		52	5990		5935		82	10904		10786		112	47561		47219	
23	3090		3027		53	6098		5972		83	10783		10631		113	46922		46820	
24	3275		3004		54	6001		5883		84	10938		10744		114	47131		47108	
25	3137	5.39	3071	1.45	55	6177	4.07	5924	1.70	85	10762	3.01	10629	1.12	115	46879	1.71	46728	1.28
26	3132		3022		56	6205		5919		86	10898		10702		116	47276		47095	
27	3193		3116		57	6247		6042		87	11128		10880		117	46592		46503	
28	3036		2876		58	**6079**		6112		88	11147		10907		118	47098		46977	
29	3199		3156		59	6116		5949		89	11101		10843		119	47028		46870	
30	3019		3002		60	6194		6024		90	11039		10855		120	47243		46824	

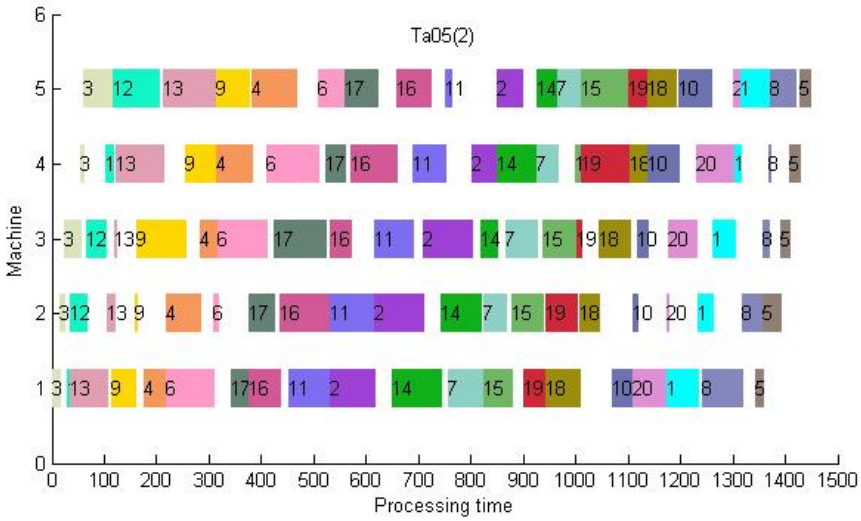

Figure 14. The Gantt chart of TA05 by Gurobi with HG1 (makespan = 1449).

Similarly, we used HG1 and HG2 to solve Reeves (REC) benchmark. The results are recorded in Table 8.

Table 8. Results of HGs on REC.

REC	HG1		HG2		REC	HG1		HG2		REC	HG1		HG2		REC	HG1		HG2	
	C_{HG1}	*RPI*	C_{HG2}	*RPI*		C_{HG1}	*RPI*	C_{HG2}	*RPI*		C_{HG1}	*RPI*	C_{HG2}	*RPI*		C_{HG1}	*RPI*	C_{HG2}	*RPI*
01	1658	8.65	1534	0.52	13	2748	7.98	2653	4.24	25	3841	6.90	3669	2.12	37	8109	1.26	8079	0.89
03	1364	0.22	*1361*	**0.00**	15	2693	6.48	2597	2.69	27	3633	5.89	3531	2.91	39	8689	3.21	8569	1.78
05	1614	6.82	1545	2.25	17	2657	2.71	2657	2.71	29	3544	7.69	3380	2.70	41	8719	3.34	8543	1.26
07	2066	1.18	2066	1.18	19	3009	5.58	2906	1.96	31	4412	2.44	4383	1.76					
09	2167	6.12	2066	1.18	21	2866	1.60	2835	0.50	33	4558	3.03	4477	1.20					
11	**2010**	6.86	2011	6.91	23	2849	5.52	2719	0.70	35	4651	5.78	4407	0.23					

On this benchmark, the values obtained by HG2 are also better than that obtained by HG1. Except for REC11, HG1 does not get a better C_{max} than HG2, and HG2 even gets the optimal solution on REC03. Obviously, the effect of HG2 is better than that of HG1, so HG2 was selected to participate in the calculation in subsequent studies.

4. Solving the Model with Matheuristic Algorithms

4.1. Test on Small-Scale Instances with MTZ Model

When Gurobi solves the NWFSP problem, if all constraints are added directly at the beginning of the optimization, the solution performance will be greatly reduced, for example, when the famous Miller-Tucker-Zemlin model [46] (Formulas (6)–(12), Equation (14) and Formula (30)) is used directly. However, the model has high research value [47]. We combine the heuristic algorithm to improve the efficiency.

$$\begin{cases} \mu_{j_1} - \mu_{j_2} + n \times x_{j_1,j_2} \leq n-1 \\ \mu_j \geq 0 \end{cases} \tag{30}$$

Using Gurobi (The version is 9.1.2.) to solve this model directly, the results are shown in Tables 9 and 10. Among them, TA02 takes a longer time, and TA37 has not completed the calculation within 36,000 s.

Table 9. Results of MTZ model on REC (tolerance is 10^{-4}).

REC	*RPI*(%)	Time(s)	REC	*RPI*(%)	Time(s)	REC	*RPI*(%)	Time(s)	REC	*RPI*(%)	Time(s)
01	**0**	0.05	13	**0**	0.10	25	**0**	0.13	37	**0**	*0.77*
03	**0**	0.08	15	**0**	0.08	27	**0**	*9.50*	39	**0**	1.06
05	**0**	0.07	17	**0**	0.20	29	**0**	0.13	41	**0**	*9.51*
07	**0**	0.26	19	**0**	0.11	31	**0**	0.35			
09	**0**	0.10	21	**0**	2.39	33	**0**	0.24			
11	**0**	0.04	23	**0**	0.11	35	**0**	0.37			

Table 10. Results of MTZ model on TA01–TA60 (tolerance is 10^{-4}).

TA	*RPI*(%)	Time(s)	TA	*RPI*(%)	Time(s)	TA	*RPI*(%)	Time(s)	TA	*RPI*(%)	Time(s)	TA	*RPI*(%)	Time(s)
01	**0**	0.18	13	**0**	0.34	25	**0**	0.29	37	*NA*	*36,000*	49	**0**	*72.10*
02	**0**	*971.08*	14	**0**	0.05	26	**0**	0.30	38	**0**	0.29	50	**0**	0.72
03	**0**	0.05	15	**0**	0.04	27	**0**	0.15	39	**0**	0.37	51	**0**	0.98
04	**0**	0.05	16	**0**	0.09	28	**0**	0.07	40	**0**	0.21	52	**0**	0.84
05	**0**	0.10	17	**0**	0.21	29	**0**	0.08	41	**0**	0.41	53	**0**	0.27
06	**0**	0.11	18	**0**	0.07	30	**0**	0.10	42	**0**	0.37	54	**0**	*8.01*
07	**0**	0.05	19	**0**	0.05	31	**0**	0.36	43	**0**	0.37	55	**0**	0.68
08	**0**	0.06	20	**0**	*59.29*	32	**0**	1.17	44	**0**	0.38	56	**0**	0.66
09	**0**	0.05	21	**0**	0.39	33	**0**	*18.69*	45	**0**	0.61	57	**0**	0.56
10	**0**	0.07	22	**0**	0.49	34	**0**	0.47	46	**0**	0.31	58	**0**	0.42
11	**0**	0.23	23	**0**	0.24	35	**0**	0.35	47	**0**	0.63	59	**0**	1.28
12	**0**	0.26	24	**0**	0.14	36	**0**	0.24	48	**0**	*154.33*	60	**0**	0.47

We select those instances (T > 7s) with longer solution times, and combine heuristics to improve efficiency. NA (not a number) indicates that the calculation has not been completed within 36,000 s.

The method of importing only the result (named as S2) of HG2 as an initial feasible solution to the optimizer is called MH2, then the solution efficiency for instances marked in italics is improved.

Since NWFSP is solved by transforming it into an ATSP problem, i.e., the solution satisfies the characteristic of a "loop". We choose the last three jobs in *S*2 to join the first three jobs to form a test sequence. The test sequence is shown in Formula (31).

$$\{S2[n-2], S2[n-1], S2[n], S2[1], S2[2], S2[3]\} \tag{31}$$

Using adjacent job pairs in the test sequence as constraints one by one, test the running time separately (if a single test exceeds 10s, test the next job pair or end the test). Among the results satisfying $RPI = 0$, select the best (the one with the shortest running time) as the improvement value and store it in the IMH2 column of Table 11. It can be seen that the speed of solving other instances in Table 11 are also improved to varying degrees (the last column in Table 11). This shows that the matheuristic method is effective.

Table 11. Comparison Results of MH2, IMH2 and MTZ only (tolerance is 10^{-4}).

Ins	*RPI*	T_{MTZ}(s)	*RPI*	T_{MH2}(s)	*RPI*	T_{IMH2}(s)	T_{MTZ}/T_{IMH2}
REC27	0	9.50	0	7.22	0	0.13	73.08
REC41	0	9.51	0	2.87	0	0.76	12.51
TA02	0	971.08	0	1726.03	0	0.05	19,421.60
TA20	0	59.29	0	63.50	0	0.08	741.13
TA33	0	18.69	0	6.29	0	5.35	3.49
TA37	NA	>36,000	NA	>36,000	0	0.51	>70,588.24
TA48	0	154.33	0	160.55	0	0.33	467.67
TA49	0	72.10	0	39.87	0	7.07	10.20
TA54	0	8.01	0	10.41	0	5.94	1.35

4.2. Comparison Results of MH2 and Other Algorithms

While we used the matheuristic in the previous subsection to improve the efficiency of the MTZ model, the strategy of dynamically adding constraints using callback functions when solving large-scale instances has greater advantages. We use this strategy to solve the proposed model (Formula 6-Formula 14), also naming the mathematic algorithm that introduces HG2 as MH2. The comparison results of MH2, DWWO [32], SWWO [34], and IDMBO [37] are shown in Tables 12 and 13.

These comparison algorithms are the main intelligent algorithms for solving NWFSP problems in recent years (2018–2020). We carefully reproduced the algorithms following the steps in the literature and set the parameters as required therein. Considering the uniformity, the condition for terminating the program was selected from the literature [34], that is, the upper limit of the running time is $n \times \mathrm{m} \times 90/2$. Each algorithm takes five independent runs for each instance, according to [37]. Calculate with Equations (32)–(34) and fill in Tables 12 and 13 with the results. All algorithms are programmed in C++ (Visual studio 2019) and run on the same desktop PC with an 11th Gen Intel(R) Core (TM) i7-1165G7 (2.80GHz) processor and 16.0 GB RAM.

$$\Delta_{\min,i1,a1} = \frac{C_{(\min,i1,a1)} - C_{(opt,a1)}}{C_{(opt,a1)}} \times 100 \tag{32}$$

$$\Delta_{Avg,i1,a1} = \frac{C_{(Avg,i1,a1)} - C_{(opt,a1)}}{C_{(opt,a1)}} \times 100 \tag{33}$$

$$SD_{i1,a1} = \sqrt{\frac{\sum_{ct=1}^{5}\left(C_{(ct,i1,a1)} - C_{(Avg,i1,a1)}\right)^2}{5}} \tag{34}$$

The minimum and average values of five independent runs of algorithm $i1$ on instance $a1$ are denoted as $C_{(\min,i1,a1)}$ and $C_{(Avg,i1,a1)}$. $\overline{\Delta_{\min}}$, $\overline{\Delta_{Avg}}$, $\overline{SD}$ and $\overline{Time}$ represent the arithmetic mean of $\Delta_{\min}$, Δ_{Avg}, SD and Time of the same scale, respectively, and $C_{(ct,i1,a1)}$ represents the result of the ctth running of the algorithm $i1$ on the instance $a1$.

Table 12 shows the test results of the four algorithms on the Tailard benchmark, in which the first column $\overline{opt.}$ records the averages of the optimal values for instances of the same size. The proposed matheuristic algorithm MH2 achieves the optimal solutions in all instances. Even for the largest instances (500 × 20), the average running time is 112.82 s. The running time of a single instance with a scale of less than or equal to 200 is short, ranging from tens of milliseconds to a dozen seconds. Therefore, the algorithm is efficient and feasible. MH2 does not contain random parameters, so the SD value is always 0, and the algorithm is stable.

It can be seen from column $\overline{\Delta_{\min}}$ that when the job size of the Tailard benchmark is greater than or equal to 50, the three metaheuristic algorithms cannot obtain an optimal solution within n × m × 90/2, even if they are run 5 times independently. As the job size increases, the difference between the output value and the optimal solution also increases. In the case of SWWO [34], which has the best comprehensive ability among the three metaheuristics, when the scale is 200 × 20, the difference between the average output solution and the average optimum is about 200 (19788.4 × 1.00%). The metaheuristic algorithm needs more running time to get a better solution, and its convergence speed is not as fast as MH2.

Table 13 shows the results of the four algorithms for solving the small-scale benchmark Reeves, where the first column records the optimal values of the instances. The operating efficiency of the MH2 algorithm is the highest, and the optimal value of any REC instance can be obtained within 1 s. The second is the SWWO algorithm [34], which can find the optimal solution except REC39 and REC41.

In summary, when $n < 30$, the difference between the four algorithms is not big, and the optimal solutions can basically be found. When $n = 50$, the search ability of the two swarm intelligence algorithms, IDMBO [37] and DWWO [32], are not as good as that of the adaptive algorithm SWWO [34]. SWWO [34] has the ability to find the optimal solution for an instance of size $n = 75$ in 67.5 s. To a certain extent, it verifies the NWFSP, which is more suitable for a search mechanism with more local search than global search. Comparing Table 7, it can be seen that when the scale is increased to 200 × 10 and 500 × 20, the $\overline{RPI}$ of the heuristic algorithm HG2 is 0.92% and 1.28% respectively, that is, the search performance also exceeds the other three intelligent algorithms. This shows that it is feasible to perform a local search with the job with the smallest PR value in the job set as the end job.

Table 12. Comparison Results of MH2, IDMBO, DWWO, and SWWO on TA.

n × m	$\overline{opt.}$	MH2		IDMBO [37]				DWWO [32]				SWWO [34]			
		$\overline{RPI}$ (%)	$\overline{Time}$ (s)	$\overline{\Delta_{min}}$ (%)	$\overline{\Delta_{Avg}}$ (%)	$\overline{SD}$	$\overline{Time}$ (s)	$\overline{\Delta_{min}}$ (%)	$\overline{\Delta_{Avg}}$ (%)	$\overline{SD}$	$\overline{Time}$ (s)	$\overline{\Delta_{min}}$ (%)	$\overline{\Delta_{Avg}}$ (%)	$\overline{SD}$	$\overline{Time}$ (s)
20 × 5	**1480.3**	**0**	0.04	**0**	0.05	0.58	4.50	**0**	**0**	0.04	4.50	**0**	**0**	**0**	4.50
20 × 10	**1983**	**0**	0.08	0.02	0.08	0.77	9.00	**0**	0.12	0.12	9.00	**0**	**0**	0.04	9.00
20 × 20	**2971.9**	**0**	0.11	**0**	0.02	0.39	18.01	**0**	0.01	0.45	18.01	0.03	0.03	0.32	18.00
50 × 5	**3269.5**	**0**	0.18	1.62	1.91	6.86	11.26	0.83	1.09	6.54	11.27	0.02	0.12	2.32	11.25
50 × 10	**4273.6**	**0**	0.29	0.38	0.70	9.58	22.51	0.20	0.37	5.64	22.51	0.03	0.08	1.85	22.50
50 × 20	**5897.4**	**0**	0.64	0.23	0.43	9.04	45.01	0.10	0.28	9.59	44.21	0.01	0.06	2.93	45.00
100 × 5	**6196.1**	**0**	0.99	3.68	4.17	20.42	22.53	3.80	4.27	20.98	22.51	0.34	0.52	8.45	22.50
100 × 10	**7991**	**0**	1.56	1.90	2.23	20.39	45.03	1.65	1.98	18.47	45.01	0.28	0.44	10.13	45.00
100 × 20	**10,658.5**	**0**	2.11	1.13	1.49	24.31	90.03	1.07	1.28	16.76	90.01	0.29	0.40	10.12	90.08
200 × 10	**15,124.7**	**0**	6.39	4.14	4.42	29.88	90.07	5.27	5.61	36.87	90.02	1.09	1.34	25.62	90.00
200 × 20	**19,788.4**	**0**	10.55	2.85	3.13	39.49	180.07	3.42	3.77	46.20	180.02	1.00	1.19	27.51	180.00
500 × 20	**46,284.1**	**0**	112.82	4.92	5.10	57.91	450.21	6.36	6.58	69.18	450.10	2.31	2.53	77.71	450.02

Table 13. Comparison Results of MH2, IDMBO, DWWO, and SWWO on REC.

REC	*opt.*	MH2		IDMBO [37]				DWWO [32]				SWWO [34]			
		$\overline{RPI}$ (%)	$\overline{Time}$ (s)	$\overline{\Delta_{min}}$ (%)	$\overline{\Delta_{Avg}}$ (%)	$\overline{SD}$	$\overline{Time}$ (s)	$\overline{\Delta_{min}}$ (%)	$\overline{\Delta_{Avg}}$ (%)	$\overline{SD}$	$\overline{Time}$ (s)	$\overline{\Delta_{min}}$ (%)	$\overline{\Delta_{Avg}}$ (%)	$\overline{SD}$	$\overline{Time}$ (s)
01	**1526**	**0**	0.08	**0**	**0**	**0**	4.50	**0**	**0**	**0**	4.50	**0**	**0**	**0**	4.50
03	**1361**	**0**	0.03	**0**	**0**	**0**	4.50	**0**	**0**	**0**	4.50	**0**	**0**	**0**	4.50
05	**1511**	**0**	0.06	**0**	0.08	1.47	4.50	**0**	**0**	**0**	4.50	**0**	**0**	**0**	4.50
07	**2042**	**0**	0.13	**0**	**0**	**0**	9.01	**0**	**0**	**0**	9.00	**0**	**0**	**0**	9.00
09	**2042**	**0**	0.05	**0**	0.04	0.40	9.00	**0**	**0**	**0**	9.00	**0**	**0**	**0**	9.00
11	**1881**	**0**	0.04	**0**	**0**	**0**	9.00	**0**	**0**	**0**	9.00	**0**	**0**	**0**	9.00
13	**2545**	**0**	0.07	**0**	**0**	**0**	13.51	**0**	**0**	**0**	13.51	**0**	**0**	**0**	13.50
15	**2529**	**0**	0.09	**0**	**0**	**0**	13.51	**0**	**0**	**0**	13.51	**0**	**0**	**0**	13.50
17	**2587**	**0**	0.07	**0**	**0**	**0**	13.50	**0**	**0**	**0**	13.51	**0**	**0**	**0**	13.50
19	**2850**	**0**	0.04	0.14	0.30	4.84	13.50	**0**	0.25	5.98	13.51	**0**	**0**	**0**	13.50
21	**2821**	**0**	0.10	**0**	0.13	3.25	13.51	**0**	0.17	4.12	13.51	**0**	**0**	**0**	13.50
23	**2700**	**0**	0.06	**0**	0.30	7.29	13.50	**0**	**0**	**0**	13.51	**0**	**0**	**0**	13.50
25	**3593**	**0**	0.13	**0**	0.06	1.83	20.26	**0**	**0**	**0**	20.26	**0**	**0**	**0**	20.25
27	**3431**	**0**	0.15	**0**	0.01	0.40	20.27	**0**	0.01	0.40	20.26	**0**	0.01	0.49	20.25
29	**3291**	**0**	0.06	**0**	**0**	**0**	20.25	**0**	0.06	4.00	20.25	**0**	**0**	**0**	20.25
31	**4307**	**0**	0.41	0.60	0.75	4.92	22.52	0.23	0.36	3.26	22.50	**0**	0.02	2.00	22.50
33	**4424**	**0**	0.14	0.47	1.26	22.92	22.51	0.52	0.77	11.21	22.50	**0**	0.14	8.58	22.50
35	**4397**	**0**	0.24	0.52	0.65	5.71	22.51	0.07	0.32	11.15	22.50	**0**	**0**	**0**	22.50
37	**8008**	**0**	0.69	0.74	1.15	25.67	67.52	0.71	0.84	10.09	67.51	**0**	0.26	12.19	67.50
39	**8419**	**0**	0.77	0.68	0.75	5.69	67.52	0.58	0.96	18.71	67.52	0.13	0.35	14.66	67.50
41	**8437**	**0**	0.96	0.91	1.11	12.28	67.53	0.33	0.76	22.69	67.52	0.08	0.25	12.70	67.50

4.3. Results and Discussion

- This study determined the optimal solutions for 141 instances on the Tailard and Reeves benchmarks. Further analysis based on the optimal solutions can verify the validity of the proposed heuristic rule. The final job of the optimal solution is often the job with a smaller *PR* value.

Step 1. Arrange the jobs according to their *PR* values from small to large. The corresponding rank with the smallest *PR* value is 0.

Step 2. Record the Final Job Ranking (*FJR*) of each optimal sequence. See Table 14 for details.

Table 14. The FJR of TA.

TA	FJR	TA	FJR	TA	FJR	TA	FJR	TA	FJR	TA	FJR	TA	FJR	TA	FJR
01	7	16	10	31	2	46	0	61	5	76	0	91	0	106	1
02	0	17	0	32	0	47	0	62	0	77	2	92	0	107	1
03	0	18	0	33	0	48	0	63	0	78	0	93	1	108	5
04	0	19	1	34	1	49	1	64	6	79	0	94	3	109	14
05	0	20	1	35	0	50	2	65	0	80	5	95	0	110	0
06	0	21	3	36	0	51	2	66	0	81	1	96	0	111	0
07	1	22	0	37	0	52	1	67	0	82	5	97	0	112	0
08	2	23	2	38	0	53	4	68	3	83	0	98	0	113	1
09	0	24	0	39	0	54	3	69	0	84	1	99	0	114	1
10	0	25	5	40	0	55	0	70	0	85	0	100	0	115	0
11	2	26	0	41	0	56	0	71	0	86	0	101	2	116	0
12	1	27	4	42	0	57	1	72	7	87	0	102	15	117	2
13	1	28	0	43	4	58	1	73	6	88	0	103	25	118	0
14	0	29	5	44	1	59	0	74	1	89	1	104	1	119	0
15	1	30	0	45	0	60	5	75	0	90	0	105	30	120	0

It can be seen from Table 14 that the *FJR* values of the instances are distributed between 0 and 30, and the case of "$FJR = 0$" accounts for 55%, as shown in Figure 15.

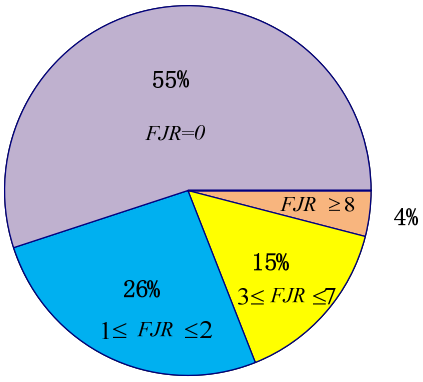

Figure 15. The PPRK value distribution ratio chart.

- The heuristic algorithm can effectively guide the optimization direction of the optimizer and improve the solution rate of complex models. Making full use of the advantages of both can not only find more optimal solutions, but also expand the application range of exact algorithms and obtain high-quality solutions that cannot be obtained by metaheuristic algorithms. Two different optimal sequences are obtained on REC05, REC09, TA01, TA03, TA05, TA21, TA32, TA33, TA34, TA35, TA41, TA42, TA44, TA45 and TA46, respectively.

5. Conclusions and Future Work

5.1. Conclusions

This paper studied the characteristics of NWFSP and verified that the PR value is an important indicator of a job. Taking the job with the smaller PR value as the final job has a higher probability of obtaining the optimal solution.

Along the evolutionary direction of HG1 and HG2, different optimal values can be obtained. This shows that HG1 and HG2 as initial solutions have good dispersion. The PR rule and $Z2$, $Z3$, $Z4$ neighborhoods are also fit to be components of intelligent algorithms.

If the job with the smallest PR value is placed in the last position, and then the neighborhood search is expanded step by step (search $Z2$, $Z3$, $Z4$ in turn), the iterative results are often consistent with the optimal solutions in the first few and last few jobs. Bringing these into the optimizer increases the model solution rate.

Experiments show that the optimal values of the two benchmarks (REC and TA) can be obtained by using the Matheuristic MH2 method within an acceptable time (the running time of a single instance with a scale of 500×20 is less than 134 s). This shows that the algorithm is feasible.

5.2. Future Work

- Considering the computing power of personal laptops, this paper only studies the three-level neighborhood division method, and initially realizes the idea of gradually expanding the search domain of NWFSP. However, for large-scale instances, the coverage of the three-level neighborhood is not enough, so the effect has a certain dependence on the initial value. In the future, more levels of neighborhood division methods and corresponding neighborhood search methods can be further studied to improve the stability of the algorithm.
- The scope of application of the proposed rules and neighborhood partitioning strategies can be generalized in the future, and they can be embedded in the framework of metaheuristic algorithms to solve NWFSP. In the past, heuristic algorithms were mostly used to construct high-quality solutions, but the rules proposed in this paper satisfy the conditions of participating in all stages. The interaction method between heuristic rules driven by problem features and metaheuristics with the advantage of generality can be further studied to improve the ability of metaheuristics to solve NWFSP.
- In real production, distributed requirements are becoming more and more extensive.

Since the matheuristic solves the NWFSP problem well, it can also be studied to solve the Distributed No-wait Flowshop Scheduling Problem (DNWFSP). On the one hand, it is possible to study the method of building a simplified model of DNWFSP based on the existing NWFSP model; on the other hand, how to use the obtained heuristic rules to guide the optimization direction of the optimizer, improve the speed of solving large-scale instances and expand the application range of exact algorithms is also a future research direction.

Author Contributions: Software, Y.G.; validation, Y.G.; writing—original draft preparation, Y.G. and Z.W.; writing—review and editing, Z.W.; supervision, L.G. and X.L.; project administration, L.G. and X.L.; funding acquisition, L.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The study did not report any data.

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

References

1. Arik, O.A. Genetic algorithm application for permutation flow shop scheduling problems. *Gazi Univ. J. Sci.* **2022**, *35*, 92–111. [CrossRef]
2. Red, S.F. An effective new heuristic algorithm for solving permutation flow shop scheduling problem. *Trans. Comb.* **2022**, *11*, 15–27.
3. Li, W.M.; Han, D.; Gao, L.; Li, X.Y.; Li, Y. Integrated production and transportation scheduling method in hybrid flow shop. *Chin. J. Mech. Eng.* **2022**, *35*, 12. [CrossRef]
4. Jemmali, M.; Hidri, L.; Alourani, A. Two-stage hybrid flowshop scheduling problem with independent setup times. *Int. J. Simul. Model.* **2022**, *21*, 5–16. [CrossRef]
5. Zhang, J.; Ding, G.F.; Zou, Y.S.; Qin, S.F.; Fu, J.L. Review of job shop scheduling research and its new perspectives under industry 4.0. *J. Intell. Manuf.* **2019**, *30*, 1809–1830. [CrossRef]
6. Sun, Y.; Pan, J.S.; Hu, P.; Chu, S.C. Enhanced equilibrium optimizer algorithm applied in job shop scheduling problem. *J. Intell. Manuf.* **2022**. Available online: https://www.webofscience.com/wos/alldb/full-record/WOS:000739739300001 (accessed on 1 January 2022). [CrossRef]
7. Aghighi, S.; Niaki, S.T.A.; Mehdizadeh, E.; Najafi, A.A. Open-shop production scheduling with reverse flows. *Comput. Ind. Eng.* **2021**, *153*, 107077. [CrossRef]
8. Ozolins, A. Dynamic programming approach for solving the open shop problem. *Cent. Europ. J. Oper. Res.* **2021**, *29*, 291–306. [CrossRef]
9. Callahan, J.R. The Nothing Hot Delay Problems in the Production of Steel. Ph.D. Dissertation, Department of Mechanical & Industrial Engineering, Toronto University, Toronto, ON, Canada, 1971.
10. Tang, J.F.; Song, J.W. Discrete particle swarm optimisation combined with no-wait algorithm in stages for scheduling mill roller annealing process. *Int. J. Comput. Integr. Manuf.* **2010**, *23*, 979–991. [CrossRef]
11. Höhn, W.; Jacobs, T.; Megow, N. On Eulerian extensions and their application to no-wait flowshop scheduling. *J. Sched.* **2012**, *15*, 295–309. [CrossRef]
12. Yuan, H.W.; Jing, Y.W.; Huang, J.P.; Ren, T. Optimal research and numerical simulation for scheduling No-Wait Flow Shop in steel production. *J. Appl. Math.* **2013**, *2013*, 498282. [CrossRef]
13. Reklaitis, G.V. Review of scheduling of process operations. *AIChE Symp. Ser.* **1982**, *78*, 119–133.
14. Hall, N.G.; Sriskandarajah, C. A survey of machine scheduling problems with blocking and no-wait in process. *Oper. Res.* **1996**, *44*, 510–525. [CrossRef]
15. Babor, M.; Senge, J.; Rosell, C.M.; Rodrigo, D.; Hitzmann, B. Optimization of No-Wait Flowshop Scheduling Problem in Bakery Production with Modified PSO, NEH and SA. *Processes* **2021**, *9*, 2044. [CrossRef]
16. Hsu, V.N.; de Matta, R.; Lee, C.Y. Scheduling patients in an ambulatory surgical center. *Nav. Res. Logist.* **2003**, *50*, 218–238. [CrossRef]
17. Mannino, C.; Mascis, A. Optimal real-time traffic control in metro stations. *Oper. Res.* **2009**, *57*, 1026–1039. [CrossRef]
18. Tomazella, C.P.; Nagano, M.S. A comprehensive review of Branch-and-Bound algorithms: Guidelines and directions for further research on the flowshop scheduling problem. *Expert Syst. Appl.* **2020**, *158*, 113556. [CrossRef]
19. Graham, R.; Lawler, E.; Lenstra, J.; Kan, A. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Math.* **1979**, *5*, 287–326.
20. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A guide to the Theory of NP-Completeness*; W.H. Freeman and Company: New York, NY, USA, 1979.
21. Papadimitriou, C.H.; Kanellakis, P.C. Flowshop scheduling with limited temporary storage. *J. ACM* **1980**, *27*, 533–549. [CrossRef]
22. Bonney, M.C.; Gundry, S.W. Solutions to the constrained flowshop sequencing problem. *J. Oper. Res. Soc.* **1976**, *27*, 869–883. [CrossRef]
23. King, J.R.; Spachis, A.S. Heuristics for flowshop scheduling. *Int. J. Prod. Res.* **1980**, *18*, 343–357. [CrossRef]
24. Gangadharan, R.; Rajendran, C. Heuristic algorithms for scheduling in the no-wait flowshop. *Int. J. Prod. Econ.* **1993**, *32*, 285–290. [CrossRef]
25. Laha, D.; Chakraborty, U.K. A constructive heuristic for minimizing makespan in no-wait flow shop scheduling. *Int. J. Adv. Manuf. Technol.* **2009**, *41*, 97–109. [CrossRef]
26. Ying, K.C.; Lu, C.C.; Lin, S.W. Improved exact methods for solving no-wait flowshop scheduling problems with due date constraints. *IEEE Access* **2018**, *6*, 30702–30713. [CrossRef]
27. Land, A.H.; Doig, A.G. An automatic method of solving discrete programming problems. *Econometrica* **1960**, *28*, 497–520. [CrossRef]
28. Pei, Z.; Zhang, X.F.; Zheng, L.; Wan, M.Z. A column generation-based approach for proportionate flexible two-stage no-wait job shop scheduling. *Int. J. Prod. Res.* **2020**, *58*, 487–508. [CrossRef]
29. Allahverdi, A. A survey of scheduling problems with no-wait in process. *Eur. J. Oper. Res.* **2016**, *255*, 665–686. [CrossRef]
30. Pan, Q.K.; Wang, L.; Tasgetiren, M.F.; Zhao, B.H. A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion. *Int. J. Adv. Manuf. Technol.* **2008**, *38*, 337–347. [CrossRef]
31. Engin, O.; Guclu, A. A new hybrid ant colony optimization algorithm for solving the no-wait flow shop scheduling problems. *Appl. Soft. Comput.* **2018**, *72*, 166–176. [CrossRef]

32. Zhao, F.Q.; Liu, H.; Zhang, Y.; Ma, W.M.; Zhang, C. A discrete water wave optimization algorithm for no-wait flow shop scheduling problem. *Expert Syst. Appl.* **2018**, *91*, 347–363. [CrossRef]
33. Zhao, F.Q.; Qin, S.; Yang, G.Q.; Ma, W.M.; Zhang, C.; Song, H.B. A factorial based particle swarm optimization with a population adaptation mechanism for the no-wait flow shop scheduling problem with the makespan objective. *Expert Syst. Appl.* **2019**, *126*, 41–53. [CrossRef]
34. Zhao, F.Q.; Zhang, L.X.; Liu, H.; Zhang, Y.; Ma, W.M.; Zhang, C.; Song, H.B. An improved water wave optimization algorithm with the single wave mechanism for the no-wait flow-shop scheduling problem. *Eng. Optimiz.* **2019**, *51*, 1727–1742. [CrossRef]
35. Zhao, F.Q.; Qin, S.; Zhang, Y.; Ma, W.M.; Zhang, C.; Song, H.B. A hybrid biogeography-based optimization with variable neighborhood search mechanism for no-wait flow shop scheduling problem. *Expert Syst. Appl.* **2019**, *126*, 321–339. [CrossRef]
36. Zhu, H.H.; Qi, X.M.; Chen, F.L.; He, X.; Chen, L.F.; Zhang, Z.Y. Quantum-inspired cuckoo co-search algorithm for no-wait flow shop scheduling. *Appl. Intell.* **2019**, *49*, 791–803. [CrossRef]
37. Zhang, S.J.; Gu, X.S.; Zhou, F.N. An improved discrete migrating birds optimization algorithm for the no-wait flow shop scheduling problem. *IEEE Access* **2020**, *8*, 99380–99392. [CrossRef]
38. Lai, R.S.; Gao, B.; Lin, W.G. Solving no-wait flow shop scheduling problem based on discrete wolf pack algorithm. *Sci. Program.* **2021**, *2021*, 4731012. [CrossRef]
39. Burke, E.K.; Kendall, G.; Soubeiga, E. A tabu-search hyperheuristic for timetabling and rostering. *J. Heuristics* **2003**, *9*, 451–470. [CrossRef]
40. Della Croce, F.; Salassa, F. A variable neighborhood search based matheuristic for nurse rostering problems. *Ann. Oper. Res.* **2014**, *218*, 185–199. [CrossRef]
41. Kramer, R.; Subramanian, A.; Vidal, T.; Cabral, L.D.F. A matheuristic approach for the Pollution-Routing Problem. *Eur. J. Oper. Res.* **2015**, *243*, 523–539. [CrossRef]
42. Hong, J.; Moon, K.; Lee, K.; Lee, K.; Pinedo, M.L. An iterated greedy matheuristic for scheduling in steelmaking-continuous casting process. *Int. J. Prod. Res.* **2021**, *60*, 623–643. [CrossRef]
43. Lin, S.W.; Ying, K.C. Optimization of makespan for no-wait flowshop scheduling problems using efficient matheuristics. *Omega-Int. J. Manag. Sci.* **2016**, *64*, 115–125. [CrossRef]
44. Bagchi, T.P.; Gupta, J.N.; Sriskandarajah, C. A review of TSP based approaches for flowshop scheduling. *Eur. J. Oper. Res.* **2006**, *169*, 816–854. [CrossRef]
45. Helsgaun, K. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *Eur. J. Oper. Res.* **2000**, *126*, 106–130. [CrossRef]
46. Gouveia, L.; Pires, J.M. The asymmetric travelling salesman problem and a reformulation of the Miller-Tucker-Zemlin constraints. *Eur. J. Oper. Res.* **1999**, *112*, 134–146. [CrossRef]
47. Campuzano, G.; Obreque, C.; Aguayo, M.M. Accelerating the Miller-Tucker-Zemlin model for the asymmetric traveling salesman problem. *Expert Syst. Appl.* **2020**, *148*, 113229. [CrossRef]