ISSN 1392–124X (print), ISSN 2335–884X (online) INFORMATION TECHNOLOGY AND CONTROL, 2014, T. 43, Nr. 3

Robust Intelligent Construction Procedure for Job-Shop Scheduling

Majid Abdolrazzagh-Nezhad[a], Salwani Abdullah[b]

[a]Department of Computer and Software Engineering, University of Birjand, Birjand, Iran
e-mail: abdolrazzagh@birjand.ac.ir

[b]Data Mining and Optimization Research Group, Centre for Artificial Intelligence Technology
Universiti Kebangsaan Malaysia, Bangi, Selangor, Malaysia
e-mail: salwani@ukm.edu.my

crossref http://dx.doi.org/10.5755/j01.itc.43.3.3536

Abstract. This paper proposes a robust intelligent technique to produce the initial population close to the optimal solution for the job-shop scheduling problem (JSSP). The proposed technique is designed by a new heuristic based on an intelligent skip from the primal point of the solution space to a better one that considers a new classification of jobs on machines. This new classification is named mPlates-Jobs. The main advantages of the proposed technique are its capability to produce any size of the initial population, its proximity to the optimal solution, and its capability to observe the best-known solution in the generated initial population for benchmark datasets. The comparison of the experimental results with those of Kuczapski's, Yahyaoui's, Moghaddam and Giffler's, and Thompson's initialization techniques, which are considered the four state-of-the-art initialization techniques, proves the abovementioned advantages. In this study, the proposed intelligent initialization technique can be considered a fast and intelligent heuristic algorithm to solve the JSSP based on the quality of its results.

Keywords: Job-shop scheduling; population-based algorithms; initialization procedures; approximation algorithms; intelligent techniques.

1. Introduction

The job-shop scheduling problem (JSSP) is one of the most difficult non-deterministic polynomial hard combinatorial complexity optimization problems [1, 2]. Since the mid-50s, research on JSSP has continued given its widespread applications in industry, management, transportation, business, and service sectors. Its history is characterized by the proposal of exact methods, such as branch and bound algorithms [3] and integer programming [4]. Although exact algorithms for very small-size instances of combinatorial optimization problems are guaranteed to find an optimal solution in bounded time, an exact algorithm to solve JSSP in polynomial time is unavailable. Current algorithms are applicable only for small-size instances. Thus, researchers focused on heuristic and meta-heuristic algorithms as approximation methods. These algorithms, namely, memetic algorithm [5], genetic algorithm [6, 7], bee colony optimization [8], ant colony optimization [9], particle swarm optimization [10], artificial immune system [11], electromagnetic-like mechanism [12], chemical reaction optimization [13], DNA computing [14], and others [15–17], are mostly population-based algorithms. These algorithms require the production of an initial population to start the exploration and exploitation of the solution space. A high-quality initial population would speed up these algorithms, but the attention of researchers is often focused on other steps of the meta-heuristic algorithm. Thus, studies on the subject are limited.

The initialization of JSSP has been performed using various methods, such as random methods, priority rules, and heuristic algorithms. Based on published literature, most of the previous researchers used random techniques, such as random keys, to produce an initial population ([6], [18], [10], [14], and [19]). However, the quality of the random points produced is far below that of optimal points. Algorithms that use random initialization require more computation time to reach an optimal solution than those that use guided techniques in the initialization section. Priority dispatching rules simulated by Moghaddam and Daneshmand [20] and Canbolat and Gundogar [21] are ranked second in the initialization techniques, according to the number of their applications in previous literature. Considering that priority rules are easy to execute and have low requirements for computational power, they have fascinated many researchers. Giffler and Thompson's (G&T) algorithm

[22] is one of the most important and original executions of priority rules. Kuczapski et al. [7] proposed an efficient initialization procedure to enhance genetic algorithm by generating near-optimal initial populations. Yahyaoui et al. [23] also proposed a heuristic initialization procedure based on integer linear programming; their results are comparable with those of other proposed initialization procedures [24].

This paper proposes a novel intelligent initialization technique to enhance population-based algorithms by generating initial populations close to the optimal solution in an acceptably short computational time. To achieve this goal, a novel heuristic is designed to generate any size of the initial population. A new intelligent skipping strategy (ISS) is embedded in the proposed heuristic to enable skipping from a primal point of solution space to a better one. The ISS is designed by first introducing a new classification of jobs, called mPlates-Jobs, to machines based on the operation number of jobs. Second, a novel set of rules, namely, the activator rules of mPlates (ARs), is designed to activate the mPlates-Jobs. Finally, ISS–ARs is produced based on the ISS and ARs of mPlates.

The rest of the paper is organized as follows: Section 2 describes and formulates JSSP. Section 3 presents the pre-processing of JSSP. Section 4 proposes a generic heuristic to produce an initial population and elaborates on the ISS embedded in the proposed heuristic. Section 5 reports the experimental and comparative results. Section 6 discusses the conclusions and future work.

2. JSSP Description

A JSSP, which is considered in this study, can be formulated as follows: A set of jobs $J = \{J_1, J_2, \dots, J_n\}$ is provided on a set of machines $M = \{M_1, M_2, \dots, M_m\}$. The operations of the jobs $O = \{O_{ij},\ i = 1,2,\dots,n\ \&\ j = 1,2,\dots,m\}$ have to be processed in the machines under three types of constraints, namely, precedence, capacity, and release and due date constraints. Precedence constraints include three limitations: each job should be processed through the sequence of machines in a predetermined order (SOJ); the machine orders among different jobs are unconfined; and no precedence constraints exist among the operations of different jobs. Capacity constraints comprise five restraints: machines are independent of one another; machines cannot remain idle while an operation is waiting for processing; each machine can only handle at most one operation at a time; each job can be processed only once on a given machine; and jobs are independent of one another. Finally, release and due date constraints contain three restrictions. First, no negative starting time is observed. Second, the processing time of operations is given a length. Finally, the processing of each operation must not be interrupted. Therefore, to satisfy these constraints and to achieve the objective of JSSP, the starting time of the processing operation is considered the decision variable of JSSP. A feasible schedule is assigned time slots on the machines for operations by satisfying the constraints of the problem and by finding a sequence of jobs on machines (SJM); the corresponding schedule should minimize the maximum completion time of the last completed operation (makespan/C_{max}) [25] as the standard objective function of JSSP. Therefore, the problem target is in finding a sequence of jobs on machines (SJM) whereby its corresponding schedule satisfies all constraints and, at the same time, minimizes the makespan.

3. Pre-processing of the JSSP

One of the key issues in successfully applying meta-heuristics to JSSP is in encoding a schedule to search space [26], that is, a suitable selection of encoding scheme is extremely important in enhancing the search effectiveness of any meta-heuristic. In addition, all of the encoding schemes proposed for the JSSP can generate an active schedule through the decoder.

3.1. Encoding Solution Space

In this study, a modified version of the preference list-based representation [26] is considered in a matrix format as the encoding scheme. One condition is designed over the encoded points to satisfy the feasibility and escape the loop of consecutive operations. Thus, the sequence of jobs on machines matrix, called SJM, is considered as the encoding scheme that represents the points of the solution space. Each row of the SJM matrix represents a permutation of jobs to be processed in a given machine. Therefore, the entries of SJM are jobs. Each job on each row (machine) has to be viewed only once. This limitation comes from precedence and capacity constraints (i.e., processing of each operation must not be interrupted, and each job can be processed only once on a given machine).

JSSP has $(n!)^m$ different $SJMs$ with m machines and n jobs. Some of these $SJMs$ do not have a feasible schedule because they have at least one loop of operations. If an SJM does not have at least one of the first processed operations of jobs (O_{i1}) with the lowest order (the first to process) on each machine as the source operation and does not have at least one of the last processed operations of jobs (O_{im}) with the highest order (the last to process) on each machine as the sink operation, then at least one loop from operations is created in SJM. There are $\left((n-2)(n-1)!\right)^m$ different $SJMs$ with at least one loop from operations; these types of SJM are rejected.

3.2. Decoder Algorithm

An encoded point is not a schedule, but it has a corresponding active feasible schedule. A feasible schedule that includes no idle time is called active.

Constructing another schedule with at least one operation finishing earlier and that with no operation finishing later is not possible [27]. Therefore, a decoder algorithm should be designed to construct an active feasible schedule that corresponds to a given encoding. The most famous decoder algorithm was proposed by Giffler and Thompson [22] and later improved by many other researchers [10, 19, 28]. In the present study, a new heuristic decoder algorithm, called the switching function, is designed to construct an active feasible schedule that corresponds to a given SJM. The algorithm has a simple structure and guarantees the satisfaction of all constraints. A brief outline of the switching function is as follows:

Step 0: $0 \rightarrow i$ *and* $0 \rightarrow j$.

Step 1: *Consider operations of jobs* (O_{ij}, $i = 1, \dots, n$ *and* $j = 1, \dots, m$) *one by one. If* ($i = n$ *and* $j = m$), *then terminate the progress; else if* ($i \neq n$ *and* $j = m$), *then* $i + 1 \rightarrow i$; *else* $j + 1 \rightarrow j$.

Step 2: $i \rightarrow i'$ *and* $j \rightarrow j'$

Step 3: *If* $O_{i'j'}$ *is not processed, then proceed to Step 4, else return to Step 1.*

Step 4: *Let* Ω' *be the set of operations of* $J_{i'}$ *with smaller index than* j' *that have not been processed. Consider* M_k *as the machine to process* $O_{i'j'}$, *and let* Ω *be the set of operations with jobs (that have not been processed) having lower order to be processed on* M_k *compared with* $J_{i'}$.

Step 5: *If* Ω *and* Ω' *are empty, evaluate* $O_{i'j'}$ *based on its constraints with the earliest possible starting time and proceed to Step 6. else if* Ω *is non-empty and* Ω' *is empty, then consider the operation with the minimum order to process in* Ω *(for example, it is* O_{rs} *) instead of* $O_{i'j'}$ *(i.e.,* $r \rightarrow i'$ *and* $s \rightarrow j'$*), and return to Step 4. else if* Ω *is empty and* Ω' *is non-empty, then consider* $O_{i'j'-1}$ *instead of* $O_{i'j'}$ *(i.e.,* $j' - 1 \rightarrow j'$*), and return to Step 4.*

Step 6: *If* $i = i'$ *and* $j = j'$ *, then return to Step 1, else* $i \rightarrow i'$, $j \rightarrow j'$, *then return to Step 4.*

For a better presentation of the procedure of the switching function, a small instance, including three jobs and three machines, is considered in Table 1.

Table 1. Original data for the small instance

Machine (Processing Time)			
	Operation		
Job	1	2	3
1	3(1)	1(3)	2(6)
2	2(3)	3(5)	1(7)
3	3(5)	2(4)	1(3)

The predetermined sequence of operations (SOJ) of the instance is as follows:

$$SOJ = \begin{bmatrix} 3 & 1 & 2 \\ 2 & 3 & 1 \\ 3 & 2 & 1 \end{bmatrix}. \tag{1}$$

A feasible encoded point of solution space (SJM^1) is arbitrarily (randomly) considered:

$$SJM^1 = \begin{bmatrix} 1 & 3 & 2 \\ 3 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix}. \tag{2}$$

The procedure of the algorithm for generating the corresponding feasible active schedule of SJM^1 is explained in Table 2 and Fig. 1. Table 2 shows that $ST(O_{ij})$ is a starting time of O_{ij}.

4. Initialization Technique

4.1. Heuristic to Generate an Initial Population

In this study, a new heuristic is proposed to generate the initial population. The algorithm of the heuristic is elaborated in Fig. 2, where two main parts are included. The first part, which uses the priority rule (shortest remaining time first, SRTF) generates one primal SJM^1. The literature on constructing initial schedules directly uses the priority rules, but in the present study, these rules are used in generating the primal SJM^1. The rule used to design SJM^1 is the SRTF. The second step includes producing the remaining SJM^i, where $i = 2, \dots, pop\_size$. The second part considers the calculation of the remaining $pop\_size - 1$ of SJMs, the SJM^{i-1} generated earlier, as a source or primal SJM. The ISS is then performed to find an improved SJM on two machines of the primal SJMconsecutively. Before running ISS, a machine (or a row of SJM^{i-1}) is randomly selected from all machines (called α) by randomly drawing a number from among $[1 \dots m]$. The ISS is designed based on a new classification of jobs on machines (mPlates-Jobs) and the AR of mPlates as an activator of mPlates. The ISS aims to find a new SJM with better quality and more difference than the primal SJM^{i-1}. ISS focuses on M_α, which is located in row α of SJM^{i-1}. ISS will then modify the orders of jobs on M_α and the other machines. Next, the SJM^i reconsiders the primal SJM to generate further SJM by randomly selecting another machine, except M_α, and performing ISS on the SJM^i.

The ISS procedure on M_α states that if a new improved SJM cannot be found after R_{expect} times, where R_{expect} relates to the number of jobs, then the value of α is randomly reselected from $[1 \dots m]$ by excluding the current value of α. Next, the ISS is re-run based on the new value of α. Reselecting the value of α is valid depending on the number of machines, and it is signified by R_{repeat}. If the procedure for finding a new improved SJM^i by reselecting the value of α is unsuccessful after R_{repeat} times, SJM^{i-2} is considered

Table 2. Sample of the procedure of the switching function algorithm

Proc.	Operations	Note
Step 1	$1 \rightarrow i, 1 \rightarrow \mathrm{j}$	
Step 2	$1 \rightarrow i', 1 \rightarrow j'$	
Step 3	$SJM(1,1) \rightarrow O_{12}$ $SOJ(O_{12}) = 1{:}\ M_1$	O_{12} is not processed and go to Step 4
Step 4	$\Omega' = \{O_{11}\},\ \Omega = \emptyset$	
Step 5	$O_{11} = SJM(3,1)$ $3 \rightarrow i',\ 1 \rightarrow j'$ $SOJ(O_{11}) = 3{:}\ M_3$	Consider O_{11} instead O_{12} and go to Step 4
Step 4	$\Omega' = \emptyset,\ \Omega = \emptyset$	
Step 5	$ST(O_{11}) = 0$ $ST(O_{11}) = 1$ $SJM(3,1)$ processed	Evaluate O_{11} Figure 1.a
Step 6	Due to $i \neq i'$ So $i \rightarrow i', j \rightarrow j'$ *and* Return to Step 4	$SJM(1,1) \rightarrow O_{12}$ $SOJ(O_{12}) = 1{:}\ M_1$
Step 4	$\Omega' = \emptyset,\ \Omega = \emptyset$	
Step 5	$ST(O_{12}) = 1$ $ST(O_{12}) = 4$ $SJM(1,1)$ processed	Evaluate O_{12} Figure 1.b
Step 6	Duo to $i = i'$ & $j = j'$ So go to Step 1	
Step 1	$2 \rightarrow i, 1 \rightarrow j$	
Step 2	$2 \rightarrow i', 1 \rightarrow j'$	
Step 3	$SJM(2,1) \rightarrow O_{32}$ $SOJ(O_{32}) = 2{:}\ M_2$	O_{32} is not processed and go to Step 4
Step 4	$\Omega' = \{O_{31}\},\ \Omega = \emptyset$	
Step 5	$O_{31} \rightarrow SJM(3,2)$ $3 \rightarrow i',\ 2 \rightarrow j'$ $SOJ(O_{31}) = 3{:}\ M_3$	Consider O_{31} instead O_{32} and go to Step 4
Step 4	$\Omega' = \emptyset,\ \Omega = \emptyset$	
Step 5	$ST(O_{31}) = 1$ $ST(O_{31}) = 6$ $SJM(3,2)$ processed	Evaluate O_{31} Figure 1.c
Step 6	Due to $i \neq i'$ & $j \neq j'$ So $i \rightarrow i', j \rightarrow j'$ & Go to Step 4	$SJM(2,1) \rightarrow O_{32}$ $SOJ(O_{32}) = 2{:}\ M_2$
Step 4	$\Omega' = \emptyset,\ \Omega = \emptyset$	
Step 5	$ST(O_{32}) = 6$ $ST(O_{32}) = 10$ $SJM(2,1)$ processed	Evaluate O_{32} Figure 1.d
Step 6	Duo to $i = i'$ & $j = j'$ So go to Step 1	
Step 1	$3 \rightarrow i, 1 \rightarrow j$	
Step 2	$3 \rightarrow i', 1 \rightarrow j'$	
Step 3	$SJM(3,1) \rightarrow O_{11}$	O_{11} is processed and go to Step 1
⋮	⋮	⋮
Step 1	$3 \rightarrow i, 3 \rightarrow j$	
Step 2	$3 \rightarrow i', 3 \rightarrow j'$	
Step 3	$SJM(3,3) \rightarrow O_{22}$	O_{22} is processed and go to Step 1
Step 1	Terminate the progress	Terminate when $i =$ 3 & $j = 3$

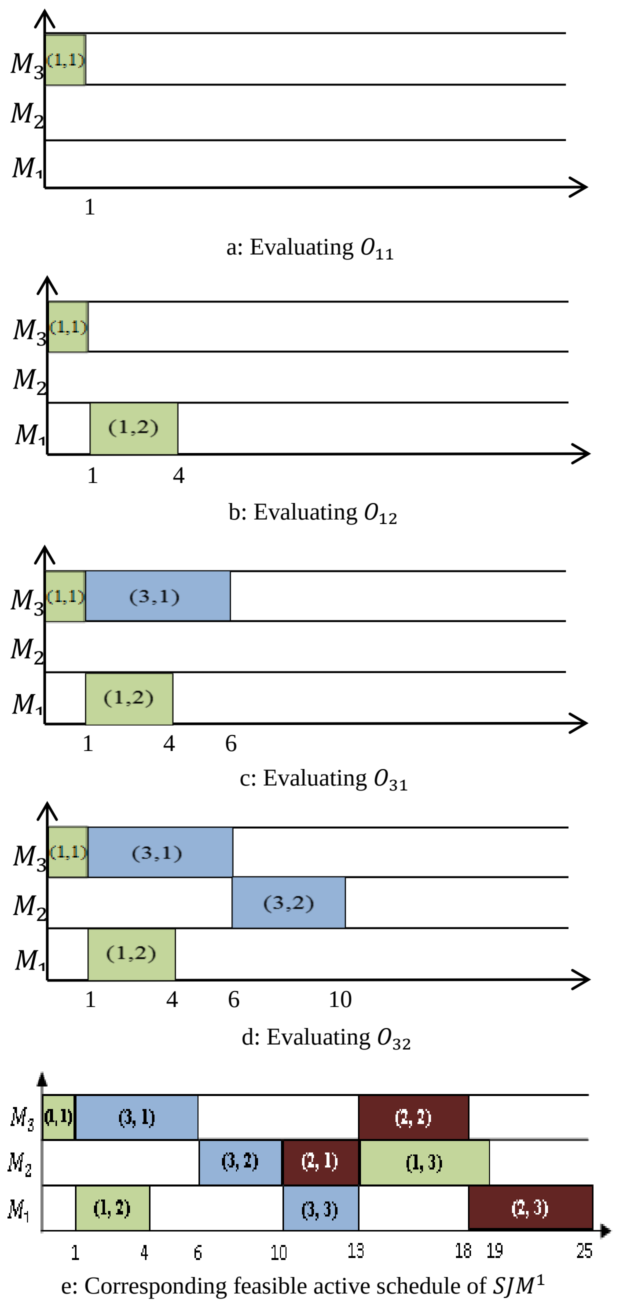

Figure 1. Procedure of the switching function on SJM^1

the primal SJM to generate SJM^i. If i equals 2 ($i = 2$), then the primal SJM is generated by the priority rule, which has not yet been applied in the construction procedure. For example, SJM^1 is produced by SRTF. In addition, unsuccessful R_{repeat} times occur when finding a new improved SJM^2 by reselecting the value of α. Thus, a new primal SJM is generated by the earliest due date. The abovementioned procedure is completed to produce SJM^i, given that the new sequence of jobs can be found on two machines/rows of the primal SJM. The new concept of mPlates-Jobs is first explained in the rest of this section. Next, the ARs of mPlates are designed, and finally, a brief outline of

ISS based on the first generation rules of mPlates (ISS–ARs) is described. The procedure of the heuristic algorithm to produce an initial population is presented in Fig. 2. The items used in Fig. 2 are explained in Table 3.

Table 3. Items used in Figure 1

Items	Description
Θ	Set of machines that is used to generate SJM^i based on the primal SJM.
r_index	It shows that the primal SJM is improved twice on two machines.
k_index	It presents that the primal SJM is improved while ISS–AR runs on α.
primal_index	It presents the index number of SJM that is considered as primal SJM.
$f\begin{pmatrix} exchanged \\ primal\ SJM \end{pmatrix}$	The quality (makespan) of the corresponding active feasible schedule to the exchanged primal SJM, which is obtained from running ISS–AR on α.
$f(primal\ SJM)$	The quality (makespan) of the corresponding active feasible schedule to the primal SJM.
R_{expect}	The upper bound on the number of iterations to find an improved SJM. while ISS–AR runs on α.
R_{repeat}	The upper bound on the number of iterations to find a new improved SJM based on the primal SJM.
SRTF	The priority rule of the shortest remaining time first.
EDD	The priority rule of the earliest due date.
LPT	The priority rule of the longest processing time.

4.2. Plates-Jobs and mPlates-Jobs

Two new job classifications are proposed, namely, Plates-Jobs and mPlates-Jobs, which are based on the concepts and structures of SOJ and SJM, respectively. In these classifications of jobs, the dependent rules, which are presented in the next subsection, create a range of candidates of processing orders for each job in SJM. The range reduces the dimension of the search space; thus, a better and new SJM can be determined in a shorter period. In Plates-Jobs, jobs are classified based on machine number and should be processed in the same operation. The classification of jobs in Plates-Jobs is obtained from SOJ. Each row of Plates-Jobs is assigned to a given operation. Each available machine in a given operation creates one class of jobs, which is named Plate-Jobs, and includes at least one job. Each Plate-Jobs is indexed through the generated machine number. The classification of jobs generated in Plates-

Generating Initial Population Procedure ( )

```
1  for $i = 1 : pop\_size$
2   if ($i$ == 1)
3     Generate $SJM^i$ based on the priority rule (SRTF)
4   else
5    Consider $SJM^{i-1}$ as the primal $SJM$
6     $i - 1 \rightarrow$ primal_index
7    $1 \rightarrow j$
8    $\emptyset \rightarrow \Theta$
9    while ($j \le 2$)
10     $1 \rightarrow r$
11     $0 \rightarrow$ r_index
12     Select a machine randomly from $[1...m]/\Theta \rightarrow \alpha$
13      $\alpha \cup \Theta \rightarrow \Theta$
14      while (($r \le R_{repeat}$)&($j \le 2$))
15        $1 \rightarrow k$
16        $0 \rightarrow$ k_index
17        while (($k \le R_{expect}$)&($j \le 2$))
18          Run the ISS–ARs on $\alpha$ from the primal $SJM$
19          if ($f(exchanged\ primal\ SJM)$
                                  $< f(primal\ SJM)$)
20            The exchanged primal $SJM \rightarrow$ the primal $SJM$
21             $R_{expect} + 1 \rightarrow k$
22             $j + 1 \rightarrow j$
23            $1 \rightarrow$ k_index
24             $R_{repeat} + 1 \rightarrow r$
25            $1 \rightarrow$ r_index
26          else
27             $k + 1 \rightarrow k$
28          end if
29        end while
30        if (k_index=0)
31          Select a machine randomly from $[1...m]/\Theta \rightarrow \alpha$
33          $\alpha \cup \Theta \rightarrow \Theta$
34          $r + 1 \rightarrow r$
35        else
36          $R_{repeat} + 1 \rightarrow r$
37          $1 \rightarrow$ r_index
38        end if
39      end while
40      if (r_index = 0)
41        if (primal_index> 1)
42           $SJM^{primal\ index-1} \rightarrow$ the primal $SJM$
43           primal_index-1 → primal_index
44         else if (primal_index = 1)
45            Generate the primal $SJM$ based on EDD
46            primal_index-1 → primal_index
47         else if (primal_index = 0)
48           Generate the primal $SJM$ based on LPT
49           primal_index-1 → primal_index
50         else
51           drop generating initial population and go to 56
52         end if
53       end if
54     end while
55   end if
56 end for
```

Figure 2. Heuristic algorithm to produce an initial population

Generating Plates-Jobs and mPlates-Jobs Procedure

```
1  for $j = 1\ to\ m$          (Operations)
2    $1 \to beta(j)$
3    $1 \to h(j, beta(j))$
4    for $i = 1\ to\ n$          (Jobs)
5      if $(i == 1)$
6        $h(j, beta(j)) \to t$
7        $SOJ(O_{ij}) \to Plates\{j, beta(j)\}\{1,1\}$
8        $J_i \to Plates\{j, beta(j)\}\{2, t\}$
9        $1 + t \to h(j, beta(j))$
10       $1 + beta(j) \to beta(j)$
11     else
12       $0 \to index$
13       $1 \to k$
14       while $(k \le beta(j) - 1)$
15         if $(Plates\{j, k\}\{1,1\} == SOJ(O_{ij}))$
16           $1 + index \to index$
17           $k \to alpha$
18           $k + beta(j) + 1 \to k$
19         else
20           $k + 1 \to k$
21         end if
22       end while
23       if $(index == 0)$
24         $h(j, beta(j)) \to t$
25         $SOJ(O_{ij}) \to Plates\{j, beta(j)\}\{1,1\}$
26         $J_i \to Plates\{j, beta(j)\}\{2, t\}$
27         $1 + t \to h(j, beta(j))$
28         $1 + beta(j) \to beta(j)$
29       else
30         $h(j, alpha) \to t$
31         $SOJ(O_{ij}) \to Plates\{j, alpha\}\{1,1\}$
32         $J_i \to Plates\{j, alpha\}\{2, t\}$
33         $1 + t \to h(j, alpha)$
34       end if
35     end if
36   end for
37 end for
38 for $j = 1\ to\ m$          (Machines)
39   $1 \to phi(j)$
40   $1 \to r\_order$
41   for $k = 1\ to\ m$          (Operations)
42     $1 \to i$
43     while $(i \le beta(k) - 1)$
44       if $(Plates\{k, i\}\{1,1\} == M_j)$
45         $O_k \to mPlates\{j, phi(j)\}\{1,1\}$
46         for $t = 1\ to\ h(k, i) - 1$
47           $Plates\{k, i\}\{2, t\} \to mPlates\{j, phi(j)\}\{2, t\}$
48           $r\_order \to mPlates\{j, phi(j)\}\{2, r\_order\}$
49           $r\_order + 1 \to r\_order$
50         end for
51         $phi(j) + 1 \to phi(j)$
52         $i + beta(k) \to i$
53       else
54         $i + 1 \to i$
55       end if
56     end while
57   end for
58 end for
```

Figure 3. Algorithm to produce Plates-Jobs and mPlates-Jobs

Jobs is reorganized based on the operation number applied over the same machine and is called mPlates-Jobs. Each row of mPlates-Jobs is assigned to a given machine. Fig. 3 shows the algorithm that produces Plates-Jobs and mPlates-Jobs. Lines 1 to 37 of the algorithm generate Plates-Jobs, and lines 38 to 58 produce mPlates-Jobs. For a clear presentation of the proposed new concepts, a benchmark instance of six-job six-machine (Ft06) by Fisher and Thompson [25] is considered:

Table 4. Original data for Ft06

	Machine (Processing Time)					
	Operation					
Job	**1**	**2**	**3**	**4**	**5**	**6**
1	3(1)	1(3)	2(6)	4(7)	6(3)	5(6)
2	2(8)	3(5)	5(10)	6(10)	1(10)	4(4)
3	3(5)	4(4)	6(8)	1(9)	2(1)	5(7)
4	2(5)	1(5)	3(5)	4(3)	5(8)	6(9)
5	3(9)	2(3)	5(5)	6(4)	1(3)	4(1)
6	2(3)	4(3)	6(9)	1(10)	5(4)	3(1)

Based on Table 4, SOJ of Ft06 can be easily presented as follows:

$$SOJ = \begin{bmatrix} 3 & 1 & 2 & 4 & 6 & 5 \\ 2 & 3 & 5 & 6 & 1 & 4 \\ 3 & 4 & 6 & 1 & 2 & 5 \\ 2 & 1 & 3 & 4 & 5 & 6 \\ 3 & 2 & 5 & 6 & 1 & 4 \\ 2 & 4 & 6 & 1 & 5 & 3 \end{bmatrix}. \qquad (3)$$

The first column/operation of SOJ to be considered has two machines (M_2 *and* M_3), and the second has four machines (M_1, M_2, M_3, and M_4). Thus, O_{i1} has two Plates-Jobs: one indexed with M_2 that includes three jobs (J_2, J_4 , and J_6) and another indexed with M_3, which has three other jobs (J_1, J_3 , and J_5). O_{i2} has four Plates-Jobs: the first belongs to M_1 and consists of two jobs (J_1 and J_4); the second depends on M_2 and has only one job (J_5); the third depends on M_3 and has one job (J_2); and the last belongs to M_4 and consists of two jobs (J_3 and J_6). Other Plates-Jobs are created with different job numbers for the remaining operations/columns of SOJ, as presented in Table 5. For example, O_{i5} has four Plates-Jobs: the first consists of one job (J_1); the second, two jobs ($J_2$and J_5); the third, one job (J_3); and the last, two jobs (J_4 and J_6).

The Plates-Jobs of Ft06 (Table 5) are individually reorganized, based on machine numbers, to create mPlates-Jobs. In the first step, M_1 is selected. M_1 has three Plates-Jobs on O_{i2} (J_1 and J_4), O_{i4} (J_3 and J_6), and O_{i5} (J_2 and J_5) because these jobs are indexed by M_1. These jobs are moved to the first row of mPlates-Jobs. This movement is repeated for the rest of the machines and their Plates-Jobs (Table 6).

Table 5. Plates-Jobs based on SOJ

Operation	Plates-Jobs			
O_{i1}	M_3: J_1, J_3, J_5	M_2: J_2, J_4, J_6		
O_{i2}	M_1: J_1, J_4	M_3: J_2	M_4: J_3, J_6	M_2: J_5
O_{i3}	M_2: J_1	M_5 : J_2, J_5	M_6 : J_3, J_6	M_3 : J_4
O_{i4}	M_4: J_1, J_4	M_6 : J_2, J_5	M_1 : J_3, J_6	
O_{i5}	M_6: J_1	M_1 : J_2, J_5	M_2 : J_3	M_5 : J_4, J_6
O_{i6}	M_5: J_1,J_3	M_4: J_2, J_5	M_6: J_4	M_3: J_6

For example, to create mPlates-Jobs related to M_4, Plates-Jobs should be found indexed by M_4 over operations on Plates-Jobs (Table 5). Three Plates-Jobs, namely, (O_{i2}: J_3, J_6), (O_{i4}: J_1, J_4) and (O_{i6}: J_2, J_5) are available to fill up the fourth row of mPlates-Jobs (see Table 6).

Table 6. mPlates-Jobs by reorganizing the Plates-Jobs based on the machine numbers

Machine	mPlates-Jobs			
M_1	O_{i2}: J_1, J_4	O_{i4}: J_3, J_6	O_{i5}: J_2, J_5	
M_2	O_{i1}: J_2, J_4, J_6	O_{i2}: J_5	O_{i3}: J_1	O_{i5}: J_3
M_3	O_{i1}: J_1, J_3, J_5	O_{i2}: J_2	O_{i3}: J_4	O_{i6}: J_6
M_4	O_{i2}: J_3, J_6	O_{i4}: J_1, J_4	O_{i6}: J_2, J_5	
M_5	O_{i3}: J_2, J_5	O_{i5}: J_4, J_6	O_{i6}: $J_1 J_3$	
M_6	O_{i3}: J_3, J_6	O_{i4}: J_2, J_5	O_{i5}: J_1	O_{i6}: J_4

The number of rows and their elements on the lower floor of mPlates-Jobs are similar to those on SJM. For example, mPlate-Jobs1 on M_2 has three jobs (J_2, J_4, and J_6). Three empty cells are found on its lower floor that can be loaded by any permutation of these three jobs, which are exactly similar to those of the first three cells of SJM on M_2/row2. Each mPlate-Jobs represents a limitation on the exchange of the processing orders of its jobs on SJM.

The empty lower floors of mPlates-Jobs are randomly filled up by the available jobs on each mPlate. An example is shown in Table 7.

The SJM randomly produced from the lower floors of mPlates-Jobs in Table 7 is presented as follows:

Table 7. Loaded mPlates-Jobs by a random permutation of available jobs in each mPlate-Jobs

Machine	mPlates-Jobs			
M_1	O_{i2}: J_1, J_4	O_{i4}: J_3, J_6	O_{i5}: J_2, J_5	
	J_1 J_4	J_6 J_3	J_2 J_5	
M_2	O_{i1}: J_2, J_4, J_6	O_{i2}: J_5	O_{i3}: J_1	O_{i5}: J_3
	J_2 J_6 J_4	J_5	J_1	J_3
M_3	O_{i1}: J_1, J_3, J_5	O_{i2}: J_2	O_{i3}: J_4	O_{i6}: J_6
	J_3 J_1 J_5	J_2	J_4	J_6
M_4	O_{i2}: J_3, J_6	O_{i4}: J_1, J_4	O_{i6}: J_2, J_5	
	J_3 J_6	J_4 J_1	J_2 J_5	
M_5	O_{i3}: J_2, J_5	O_{i5}: J_4, J_6	O_{i6}: $J_1 J_3$	
	J_2 J_5	J_6 J_4	J_3 J_1	
M_6	O_{i3}: J_3, J_6	O_{i4}: J_2, J_5	O_{i5}: J_1	O_{i6}: J_4
	J_6 J_3	J_2 J_5	J_1	J_4

$$SJM = \begin{bmatrix} 1 & 4 & 6 & 3 & 2 & 5 \\ 2 & 6 & 4 & 5 & 1 & 3 \\ 3 & 1 & 5 & 2 & 4 & 6 \\ 3 & 6 & 4 & 1 & 2 & 5 \\ 2 & 5 & 6 & 4 & 3 & 1 \\ 6 & 3 & 2 & 5 & 1 & 4 \end{bmatrix}. \qquad (4)$$

The makespan of the SJM based on Eq. (4) is 60, whereas the optimal makespan of Ft06 is 55. SJM (Eq. (4)) is randomly generated based on available real processing orders for candidate jobs on their mPlates. This finding shows the capability and efficiency of the proposed classification of jobs (mPlates-Jobs).

4.3. The Activators Rules of mPlates (ARs)

An mPlates-Jobs is designed as a classification of jobs over machines based on SOJ and when matched with the structure of SJM. This classification is generated just once in the proposed intelligent initialization technique and possesses a fixed structure. Three rules are proposed to activate and to make up training on the mPlates-Jobs. These rules, which are regarded as the ARs of mPlates, are as follows:

Rule 1: *Two consecutive mPlates before and two consecutive mPlates after for each mPlate that belongs to consecutive operations on a given machine (if it exists) should combine together and generate a range of candidate orders based on three-layer priority, namely, central layer (CL), first layer (FL), and second layer (SL).*

Rule 2: *For jobs on a given machine in SJM based on the abovementioned rule, a range of candidates to the three-layer priority will exist in which the probability of the orders of the CL is higher than those of the other layers and can be selected between 55% and 75%. For the next layer (FL), the probability is* *between 25% and 40%, and for the last layer (SL), it is between 5% and 10%.*

Rule 3: *If an mPlate has only one job but does not have any consecutive mPlates before or after it with consecutive operations, the nearest mPlate before and after is considered as the FL by a probability between 20% and 30%; the probability of selecting the order of the CL is between 70% and 80%.*

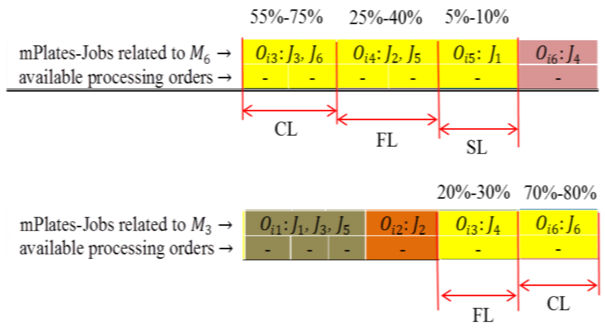

Figure 4. Implementations of the ARs of mPlates on M_3 and M_6 for J_6

The range of candidate orders for each job on a given machine becomes limited based on the ARs. In this range, each order has a different selection probability. For example, mPlates-Jobs related to M_6 and M_3 are considered to explain the ARs (Fig. 4). The mPlates-Jobs related to M_6 include four mPlates (O_{i3}, O_{i4}, O_{i5}, and O_{i6}), which consecutively operate. With regard to Rules 1 and 2, the range of candidate orders for J_3 or J_6 on M_6 (the sixth row of SJM) is (1, 2, 3, 4, 5) such that orders 1 and 2 have 55% to 75% chances of sitting J_3 or J_6 on them. The chances of orders 3 and 4 to select are 25% to 40%, and order 5 has 5% to 10% chance of selecting J_3 or J_6. According to Rule 3, J_6 can obtain order 5 with 20% to 30% probability, and it can sit on order 6 with 70% to 80% chances. However, the mPlates of O_{i3} and O_{i6} are not consecutive. Reducing the search space of the solution space by using ARs depends on the dimensions of the dataset (number of jobs and machines) and the complexity of the process. For datasets with larger dimensions, more reduction in the search space from the solution space will occur. For instance, the search space of La19 [29] with 10 jobs:10 machines has to reduce 3.9594e + 64 points to 1.8092e + 40 based on the ARs.

4.4. ISS Based on the ARs

The robust intelligent initialization technique is completed by designing an ISS based on the ARs. The technique is called ISS–ARs. For ISS–ARs, one condition with two equations (Eqs. (5) and (6)) is considered as the threshold for performing ARs. If the jobs on a given machine satisfy the thresholds for performing ARs, their processing orders are exchanged in relation to ARs. Finally, the processing orders of the jobs, which have moved, on the other machines are modified. A brief outline of ISS–ARs is as follows:

Step 1: *Focus on a given machine/row (M_α) of the primal SJM, recognize the current orders of jobs on M_α, and identify the mPlates-Jobs, which can be combined based on Rule 1.*

Step 2: *Classify the list candidate orders of jobs and the probability that the jobs will be selected based on Rule 2 or Rule 3 in three or two layers.*

Step 3: *Focus on each of the mPlates of the primal SJM ($mPlate(M_\alpha, k)$), and for each job (J_r) on $mPlate(M_\alpha, k)$ with the c_order ρ, randomly select a new order based on the probability of the candidate orders for J_r as a target order τ. If J_r and its consecutive jobs can satisfy Condition 1 and Eq. (5) or Eq. (6), then J_r is moved to the target order τ.*

To choose the target order randomly, one layer from CL, FL, and SL should be selected based on their probability. Subsequently, one order from the selected layer is kept random.

Step 4: *Simultaneously, move jobs from their current order to a new order (Step 3). The movement should be made with certainty that the order of these jobs in their other operations has settled on other machines with the best value. Otherwise, exchange the order during its mPlate, if jobs satisfy Condition 1 and Eq. (5).*

Steps 3 and 4 are repeated for all mPlates-Jobs on M_α and are created as an exchanged version of the primal SJM. To check the quality of the exchanged version of the primal SJM after Steps 2 to 4, and to determine whether it has improved or not, the switching function should be run over the exchanged version of the primal SJM. The initialization procedure that works based on this strategy is called ISS–ARs.

Condition 1: Suppose that the c_order of J_r with operation (O_{rs}) is ρ on the primal SJM and the nearest r_order based on its mPlate equals τ. To match the c_order of J_r with its nearest r_order, J_r should be moved by one unit to the left or right depending on the value of τ. If $\rho > \tau$, jobs with c_order $\rho - 2, \rho - 1$, and $\rho + 1$ are J_p with operation O_{pq}, J_i with operation O_{ij}, and J_a with operation O_{ab}, respectively. If these operations satisfy Eq. (5), then the orders of J_r and J_i should be swapped, and the makespan of the exchanged primal SJM should be recalculated. Subsequently, the procedure should be repeated until the current order of J_r reaches τ.

$$\begin{cases} \theta \leq CT(O_{rs+1}) \\ and \\ max\left(CT(O_{ij-1}), \theta\right) + PT\left(O_{ij}\right) \leq CT(O_{ab}) \end{cases} \tag{5}$$

In the abovementioned equation, $\theta = max\left(CT(O_{rs-1}), CT\left(O_{pq}\right)\right) + PT(O_{rs})$ and CT and PT are completion time and processing time, respectively.. If no J_a is found, then $CT(O_{ab})$ equals the makespan of the primal SJM. If J_p is not available, then $CT(O_{pq})$ equals zero. Conversely, if $\rho < \tau$, jobs with c_orders $\rho - 1$, $\rho + 1$, and $\rho + 2$ are J_p with operation O_{pq}, J_i with operation O_{ij}, and J_a with operation O_{ab}, respectively. If these operations satisfy Eq. (6), then the c_orders of J_r and J_i should be swapped, the makespan of the exchanged primal SJM should be recalculated, and then the procedure should be repeated until the current order of J_r reaches τ.

$$\begin{cases} \delta \le CT(O_{ij+1}) \\ and \\ max(CT(O_{rs-1}), \delta) + PT(O_{rs}) \le CT(O_{ab}) \end{cases} \tag{6}$$

In the abovementioned equation, $\delta = max\left(CT(O_{ij-1}), CT(O_{pq})\right) + PT(O_{ij})$. If J_a does not exist, then $CT(O_{ab})$ equals the makespan of the primal SJM. If J_p is not available, then $CT(O_{pq})$ equals zero. To illustrate the execution of the ISS–ARs on a given machine clearly, a numerical sample is presented as follows: SJM_2^{Ft06} in Eq. (7) with a makespan of 72 is supposed as a primal SJM for Ft06 and is focused on M_1:

$$SJM_2^{Ft06} = \begin{bmatrix} 1 & 4 & 5 & 6 & 3 & 2 \\ 6 & 4 & 2 & 1 & 5 & 3 \\ 1 & 3 & 5 & 2 & 4 & 6 \\ 6 & 3 & 1 & 4 & 5 & 2 \\ 5 & 2 & 1 & 4 & 6 & 3 \\ 6 & 3 & 1 & 5 & 2 & 4 \end{bmatrix} \tag{7}$$

The current orders (c_order) of jobs on M_1 should be recognized in relation to Step 1, and the mPlates, which can be combined based on Rule 1 of ARs, should be identified. M_1 has three mPlates-Jobs: $mPlate(M_1, 1)$, which includes J_1 and J_4 and indexed by O_{i1}; $mPlate(M_1, 2)$, which consists of J_3 and J_6 and is referred to as O_{i4}; and $mPlate(M_1, 3)$, which is composed of J_2 and J_5 and cited by O_{i5}. Given $mPlate(M_1, 2)$ and $mPlate(M_1, 3)$, jobs are indexed by consecutive operations, and they can be combined as the FL. Meanwhile, $mPlate(M_1, 1)$, which remains as one layer, is the CL. The c_order layer for each mPlate and the list of candidate orders with their probabilities for jobs based on Rules 2 and 3 of ARs, which are in accordance with Step 2, are shown in Table 8.

In Step 3, the first $mPlate(M_1, 1)$ includes J_1 and J_4. Next, order1 and order2 are randomly selected for J_4 and J_1, respectively. J_4 and J_1 can be swapped with each other (J_4 can be moved to order 1) because these conditions satisfy Eq. (6). Thus, the exchanged sequence of jobs on M_1 is equal to (4 1 5 6 3 2). Moreover, given the movement of J_4 to the right side, the current orders of J_4 on other machines should be simultaneously controlled. If the condition of Eq. (5) is satisfied, then the current orders should be exchanged, and the makespan of the exchanged primal SJM should be recalculated based on Step 4. J_1 should not be moved because its target and current orders are similar. Newly exchanged primal SJM, a result of performing Steps 3 and 4 on $mPlate(M_1, 1)$, has six swaps and is presented in SJM_3^{Ft06} (Eq. (8)) with a makespan of 72. The results of the movement to other operations of J_4 on another machine based on Step 4 are presented in Table 9.

$$SJM_3^{Ft06} = \begin{bmatrix} 4 & 1 & 5 & 6 & 3 & 2 \\ 4 & 6 & 2 & 1 & 5 & 3 \\ 1 & 3 & 4 & 5 & 2 & 6 \\ 6 & 3 & 4 & 1 & 5 & 2 \\ 5 & 2 & 4 & 1 & 6 & 3 \\ 6 & 3 & 1 & 5 & 2 & 4 \end{bmatrix} \tag{8}$$

Table 8. List of candidate orders for jobs on M_1

Job	mPlate	mPlates in each layer	candidate orders	probability of candidate orders	current order of jobs	operation of jobs
J_1	1	CL: 1	1, 2	100%	1	O_{12}
		FL: -	-	-		
		SL: -	-	-		
J_2	3	CL: 3	5, 6	85%	6	O_{25}
		FL: 2	3, 4	15%		
		SL: -	-	-		
J_3	2	CL: 2	3, 4	85%	5	O_{34}
		FL: 3	5, 6	15%		
		SL: -	-	-		
J_4	1	CL: 1	1, 2	100%	2	O_{42}
		FL: -	-	-		
		SL: -	-	-		
J_5	3	CL: 3	5, 6	85%	3	O_{55}
		FL: 2	3, 4	15%		
		SL: -	-	-		
J_6	2	CL: 2	3, 4	85%	4	O_{64}
		FL: 3	5, 6	15%		
		SL: -	-	-		

Table 9. Results of movement of J_4 based on Step 4

Operation	Machine	Current order	Candidate order	Eq. (5)	makespan
O_{41}	M_2	2	1	Satisfied	72
O_{43}	M_3	5	4	Satisfied	72
		4	3	Satisfied	72
		3	2	Unsatisfied	-
O_{44}	M_4	4	3	Satisfied	72
		3	2	Unsatisfied	-
O_{45}	M_5	4	3	Satisfied	72
		3	2	Unsatisfied	-
O_{46}	M_6	6	5	Unsatisfied	-

The movements of jobs on $mPlate(M_1, 1)$ are thus completed. As such, $mPlate(M_1, 2)$ is considered the next mPlate. Steps 3 and 4 are repeated for jobs on $mPlate(M_1, 2)$, whereas Step 3, executed on $mPlate(M_1, 2)$, begins on J_3. The target order of J_3 is chosen from two layers. CL includes order3 and order4 with 85% probability, and FL consists of order5 and order6 with 15% probability (Table 8). Therefore, based on the probability of the orders, order3 is randomly selected for J_3. The c_order of J_3 is order5. If J_3 and the jobs in the right side can satisfy Eq. (5), then first, J_3 should be moved to order4 and the next one should be moved to order3. Thus, J_3 and J_6 are swapped with each other (J_3 is moved to order4) once they satisfy Eq. (5). Subsequently, the makespan value is recalculated. Therefore, the exchanged sequence of jobs on M_1 equals (4 1 5 3 6 2). Next, the order of J_3 on other machines should be checked based on Step 4. The results are presented in Table 10.

Table 10. Results of movement of J_3 based on Step 4

Operation	Machine	Current order	Candidate order	Eq.(5)	makespan
O_{31}	M_3	2	1	Satisfied	72
O_{32}	M_4	2	1	Satisfied	72
O_{33}	M_6	2	1	Satisfied	72
O_{35}	M_2	6	5	Unsatisfied	-
O_{36}	M_5	6	5	Satisfied	72
		5	4	Satisfied	72
		4	3	Unsatisfied	-

Step 3 is repeated over J_3 to check the capability of J_3 to move to order 3. J_3 and the jobs in the right side (J_5) satisfy Eq. (5) and therefore can be swapped with each other. The exchanged sequence of jobs on M_1 equals (4 1 3 5 6 2). The makespan of the exchanged SJM is 68. Next, the order of J_3 on other machines is checked based on Step 4. The results are shown in Table 11.

Table 11. Results of movement of J_3 based on Step 4

Operation	Machine	Current order	Candidate order	Eq. (5)	makespan
O_{31}	M_3	1	-	-	-
O_{32}	M_4	1	-	-	-
O_{33}	M_6	1	-	-	-
O_{35}	M_2	6	5	Unsatisfied	-
O_{36}	M_5	4	3	Satisfied	68
		3	2	Unsatisfied	-

The implementations of Steps 3 and 4 on J_3 are then completed, and J_3 reaches the target order. J_6 is then considered the next job of $mPlate(M_1, 2)$, and order 4 is randomly selected as the target order for J_6. The current order of J_6 is order5 on the last exchanged version of the primal SJM. The results of checking the order of J_6 on other machines in Step 4 are presented in Table 12. The implementations of Steps 3 and 4 over jobs on $mPlate(M_1, 2)$ and SJM_4^{Ft06} are completed, and the exchanged version of SJM with a makespan of 67 is available based on Eq. (9).

$$SJM_4^{Ft06} = \begin{bmatrix} 4 & 1 & 3 & 6 & 5 & 2 \\ 4 & 6 & 2 & 1 & 5 & 3 \\ 3 & 1 & 4 & 5 & 6 & 2 \\ 6 & 3 & 4 & 1 & 5 & 2 \\ 5 & 2 & 3 & 4 & 6 & 1 \\ 6 & 3 & 1 & 5 & 2 & 4 \end{bmatrix} \tag{9}$$

Table 12. Results of movement of J_6 based on Step 4

Operation	Machine	Current order	Candidate order	Eq. (5)	makespan
O_{61}	M_2	2	1	Unsatisfied	-
O_{62}	M_4	2	1	Satisfied	68
O_{63}	M_6	2	1	Satisfied	68
O_{65}	M_5	6	5	Satisfied	67
		5	4	Unsatisfied	-
O_{66}	M_3	6	5	Satisfied	67
		5	4	Unsatisfied	-

Finally, $mPlate(M_1, 3)$ and its jobs (J_2 and J_5) are considered in Step 2. Orders 5 and 6 are selected as the target orders of J_2 and J_5, respectively. Given that the current and target orders of J_2 and J_5 in the last exchanged version of the primal SJM (SJM_4^{Ft06}) are similar, the procedure of ISS–ARs is completed for all jobs on M_1. The comparison of SJM_2^{Ft06} as the primal SJM with SJM_4^{Ft06} as the exchanged SJM produces 16 variations (44.4% exchanges based on the overall exchanges), although 19 movements (52.8% exchanges) occur during the performance of ISS–ARs on M_1. Meanwhile, ISS–ARs should be simultaneously run on two machines to generate one point (SJM) of the initial population. Thus, to generate a new point of the initial population from SJM_2^{Ft06} as the primal SJM, ISS–ARs should be performed on other machines except M_1.

5. Experimental Results

ISS–ARs is tested on benchmark datasets available in the OR library [30] and Willem [24] to evaluate the proposed intelligent construction procedure. The dataset details and parameters, which are used to execute ISS–ARs, include programming, processor, population size, R_{expect}, R_{repeat}, and others. These details and parameters are presented in Table 13. The experimental results of ISS–ARs are presented in Table 14, based on the makespan as the objective function (Ran. C_{max}). G_n is the number of generations required to reach the solution, and T is the CPU time to produce the initial population. The experimental results are also compared with the best-known solution (BKS), simulated G&T algorithm [22], heuristic initialization (IPG) proposed in [7], GA with IPG (IPG + GA), and without IPG as evaluated in [7].

Table 13. Experimental details and parameters

Programming	Matlab9
Processor	Intel Core2 Due P8600 2.4 GHz
Population size	Twice the number of jobs (2 n)
R_{expect}	80% of the number of jobs (0.8 n)
R_{repeat}	Number of machines (m)
Number of runs of ISS–ARs over datasets	10 times
Main criterion in the experiment	Makespan as quality of generated points
Second calculated measure	CPU time as computational time to produce initial population

Table 14. Experimental results and their comparison with the results of G&T, GA, and Kuczapski's algorithm [7]

Instance	**BKS**	**ISS–ARs**			**G&T**	**GA**		**IPG + GA**		**IPG**	
		Ran. C_{max}	**G_n**	**T**	**C_{max}**	**C_{max}**	**G_n**	**C_{max}**	**G_n**	**C_{max}**	**G_n**
Ft06	55	71–57	≤ 12	0.49	61	55	112	55	124	55	23
Ft10	930	1509–1046	≤ 20	6.65	1228	1051	178	1007	187	1043	27
Ft20	1165	1739–1206	≤ 40	14.79	1565	1295	200	1223	151	1230	29
La01	666	892–666	≤ 20	2.26	772	676	154	668	195	701	36
La02	655	878–655	≤ 20	2.12	899	697	26	677	149	704	26
La03	597	1001–597	≤ 20	2.09	771	628	141	640	179	653	31
La26	1218	2023–1325	≤ 40	65.34	1433	1479	200	1316	166	1348	35
La27	1235	1806–1395	≤ 40	66.92	1593	1556	186	1426	224	1460	32
La28	1216	1928–1357	≤ 40	65.60	1557	1506	238	1403	206	1460	35
La29	1157	1934–1351	≤ 40	66.58	1496	1481	208	1385	287	1449	38
La30	1355	2110–1512	≤ 40	65.09	1614	1595	210	1492	269	1560	35
La36	1268	1752–1447	≤ 30	15.85	1546	1500	191	1434	221	1543	38
La37	1397	2120–1579	≤ 30	15.13	1579	1623	195	1554	194	1580	29
La38	1196	1830–1381	≤ 30	16.50	1466	1442	185	1338	202	1370	33
La39	1233	1767–1401	≤ 30	15.89	1532	1460	220	1397	177	1417	34
La40	1222	1844–1361	≤ 30	14.92	1539	1438	96	1288	170	1297	35

Table 15. Experimental results and their comparison with the results of Yahyaoui's algorithm [23]

Ins.	**ISS–ARs**			**CSANN**	**FM**			**SM**			**YTPM**		
	C_{max}	**G_n**	**T**	**C_{max}**	**C_{max}**	**G_n**	**T**	**C_{max}**	**G_n**	**T**	**C_{max}**	**G_n**	**T**
Ft06	57	≤ 12	0.49	55	85.2	472	156.4	129.1	2725	578.4	67.4	393	81.89
Willem	94	≤ 20	5.06	95	142	1596	34562	256	4562	45869	107	1358	27994

An interesting point in Table 14 is the observation of the BKS among the initial populations generated by ISS–ARs for La01, La02, and La03. Although ISS–ARs is designed to generate the initial population, the results can challenge GA based on the results presented in Table 14.

Table 15 includes the simulation results by Yahyaoui et al. [23] and compares the best point produced in the initial population generated by ISS–ARs. Three initialization methods are proposed for the neural network to solve JSSP [23]. The authors considered two initialization methods based on random techniques, namely, first method (FM) and second method (SM), and proposed a new heuristic initialization method, namely "the third new proposed method" (YTPM). They also considered the results of a constraint satisfaction adaptive neural network (CSANN) to evaluate their results. Table 15 shows that ISS–ARs succeeded in generating a point better than BKS for Willem's dataset [24], which belonged to Yang and Wang [31].

Finally, Table 16 shows the experimental results of ISS–ARs that are compared with those of the simulating priority rules proposed by Moghadam and Daneshmand-Mehr [20]: first in first out (FIFO), last in first out (LIFO), low-value function (LVF), and high-value function (HVF).

The advantages of ISS–ARs are clearly proven by the quality of solutions and the lower computational times obtained through the comparisons presented in Tables 14 to 16.

Table 16. Comparison of the experimental results with simulated priority rules by Moghaddam [20]

Instance	FIFO	LIFO	HVF	LVF	ISS–ARs
Ft06	61	69	68	69	57
Ft10	1184	1283	1240	1370	1046
Ft20	1645	1291	1656	1336	1206

6. Conclusion

A novel intelligent initialization technique is proposed to generate an initial population close to the optimal solution. The technique is based on an ISS from a primal point to a better one. The ISS is designed based on a new classification of jobs, called mPlates-Jobs, which considers a predetermined precedence constraint (SOJ). A set of rules, namely, ARs, is proposed to activate and formulate the training of the mPlates-Jobs.

ISS–ARs can produce any size of initial population. Based on the experimental results and the comparison with other available valid methods in Tables 14 to 16, ISS–ARs generate an initial population in a significantly short computation time. The best point of the produced initial population is close to an optimal solution. The important result that shows the advantage of the ISS–ARs is the observation of the BKS among the initial populations of La01, La02, and La03 in Table 14. Another important result is the successful generation of a point that is better than the BKS of Willem's dataset [24] among the initial population, as shown in Table 15. The ISS–AR of this study is a fast, intelligent heuristic algorithm for solving the JSSP, based on the quality of the experimental results.

This study has created the perfect foundation through which a more effective procedure to produce the initial population with a shorter computation time and better quality than ISS–AR can be designed. The implementation of ISS–ARs in other cases of JSSPs and other scheduling problems is the recommended second phase of future studies.

References

[1] **B. J. Lageweg, J. K. Lenstra, A. Rinnooy Kan.** Job-shop scheduling by implicit enumeration. *Management Science*, 1977, 441-450.

[2] **D. Applegate, W. Cook.** A computational study of the job shop scheduling problem. *ORSA Journal on Computing*, 1991, Vol. 3, 149-156.

[3] **P. Brucker, B. Jurisch, B. Sievers.** A Branch and Bound Algorithm for the Job-Shop Scheduling Problem. *Discrete Appl. Math.*, 1994, Vol. 49, 107-127.

[4] **M. L. Fisher**. Optimal solution of scheduling problems using Lagrange multipliers: Part I. *Operation Research*, 1973, Vol. 21, 1114-1127.

[5] **H. C. Cheng, T. C. Chiang, L. C. Fu.** Multiobjective job shop scheduling using memetic algorithm and shifting bottleneck procedure. In: *Proceedings of Symposium on Computational Intelligence in Scheduling*, 2009, 15-21.

[6] **K. P. Dahal, G. M. Burt, J. R. McDonald, A. Moyes.** A case study of scheduling storage tanks using a hybrid genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 2001, Vol. 5, No. 3, 283-294.

[7] **A. M. Kuczapski, M. V. Micea, L. A. Maniu, V. I. Cretu.** Efficient Generation of Near Optimal Initial Populations to Enhance Genetic Algorithms for Job Shop Scheduling. *Information Technology and Control*, 2010, Vol. 39, No. 1, 32-37.

[8] **L. Wang, G. Zhou, Y. Xu, S. Wang, M. Liu**. An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 2012, Vol. 60, No. 1, 303-315.

[9] **L. Deng, V. Lin, M. Chen.** Hybrid ant colony optimization for the resource-constrained project scheduling problem. *Journal of Systems Engineering and Electronics*, 2010. Vol. 21, No. 1, 67-71.

[10] **G. Hong-Wei, S. Liang, L. Yan-Chun, Q. Feng.** An Effective PSO and AIS-Based Hybrid Intelligent Algorithm for Job-Shop Scheduling. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 2008, Vol. 38, No. 2, 358-368.

[11] **R. Zhang, C. Wu.** A hybrid immune simulated annealing algorithm for the job shop scheduling problem. *Applied Soft Computing*, 2010, Vol. 10, No. 1, 79-89.

[12] **V. Roshanaei, A. K. G. Balagh, M. M. S. Esfahani, B. Vahdani.** A mixed-integer linear programming model along with an electromagnetism-like algorithm for scheduling job shop production system with sequence-dependent set-up times. *The International Journal of Advanced Manufacturing Technology*, 2010, Vol. 47, No. 5, 783-793.

[13] **J. Q. Li, Q. Pan.** Chemical-reaction optimization for flexible job-shop scheduling problems with maintenance activity. *Applied Soft Computing*, 2012, Vol. 12, No. 9, 2896-2912.

[14] **Y. Zhixiang, C. Jianzhong, Y. Yan, M. Ying.** Job shop scheduling problem based on DNA computing. *Journal of Systems Engineering and Electronics*, 2006, Vol. 17, No. 3, 654-659.

[15] **Z. Jie, L. Xiaoping**, An Effective Meta-Heuristic for No-Wait Job Shops to Minimize Makespan. *IEEE Transactions on Automation Science and Engineering*, 2012, Vol.9, No.1, 189-198.

[16] **S. Rong-Lei, L. Han-Xiong, X. Youlun.** Performance-oriented integrated control of production scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 2006, Vol. 36, No. 4, 554-562.

[17] **S. S. Walker, R. W. Brennan, D. H. Norrie.** Holonic job shop scheduling using a multiagent system. *Intelligent Systems*, 2005, Vol. 20, No.1, 50-57.

[18] **K. Krishna, K. Ganeshan, D. J. Ram.** Distributed simulated annealing algorithms for job shop scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 1995, Vol. 25, No. 7, 1102-1109.

[19] **D. Y. Sha, H. H. Lin.** A Multi-Objective PSO for Job Shop Scheduling Problems. *Int. J. of Expert System with Application*, 2010, Vol.37, 1065-1070.

[20] **R. T. Moghaddam, M. Daneshmand-Mehr.** A Computer Simulation Model for Job Shop Scheduling Problems Minimizing Makespan. *Computers & Industrial Engineering*, 2005, Vol. 48, 811-823.

[21] **Y. B. Canbolat, E. Gundogar.** Fuzzy priority rule for job shop scheduling. *J. Intell. Manuf*, 2004, Vol.15, No. 4, 527-533.

[22] **B. Giffler, G. L. Thompson.** Algorithms for solving production-scheduling problems. *Operations Research*, 1960, Vol. 8, No. 4, 487-503.

[23] **A. Yahyaoui, N. Fnaiech, F. Fnaiech.** A Suitable Initialization Procedure for Speeding a Neural Network Job-Shop Scheduling. *IEEE Transactions on Industrial Electronics*, 2011, Vol. 58, No. 3, 1052-1060.

[24] **T. M. Willems, L. W. Brandts.** Implementing Heuristics as an Optimization Criterion in Neural Networks for Job Shop Scheduling. *Int. J. Manuf.*, 1995, Vol. 6, 377-387.

[25] **H. Fisher, G. L. Thompson.** Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. Eds. J.F. Muth and G.L. Thompson. 1963, Industrial Scheduling, Prentice Hall, Englewood Cliffs: New Jersey. 225-251.

[26] **R. Cheng, M. Gen, Y. Tsujimura.** A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. *Computers & industrial engineering*, 1996, Vol. 30, No. 4, 983-997.

[27] **M. L. Pinedo.** Scheduling, Theory, Algorithm and Systems. 3rd. ed., *Springer: New York,* 2008.

[28] **V. Sels, K. Craeymeersch, M. Vanhoucke.** A hybrid single and dual population search procedure for the job shop scheduling problem. *European Journal of Operational Research*, 2011, Vol. 215, No. 3, 512-523.

[29] **S. Lawrence.** Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement), in *Graduate School of Industrial Administration*., Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.

[30] **J. E. Beasley.** OR-Library. Available from: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html.

[31] **S. Yang, D. Wang.** Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling. *IEEE Transactions on Neural Networks*, 2000, Vol. 11, No. 2, 474-486.

Received February 2013.