Applied Soft Computing 24 (2014) 63–77

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem

Jun-qing Li [a,b], Quan-ke Pan [a,b,*], Fa-tao Wang [b,c]

[a] State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, PR China
[b] College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[c] School of Economics and Management, Beijing University of Posts and Telecommunications, Beijing 100876, PR China

ARTICLE INFO

Article history:
Received 26 June 2013
Received in revised form 3 July 2014
Accepted 5 July 2014
Available online 12 July 2014

Keywords:
Hybrid flow shop scheduling problem
Chemical-reaction optimization
Estimation of distribution
Variable neighborhood search

ABSTRACT

This paper proposes a hybrid variable neighborhood search (HVNS) algorithm that combines the chemical-reaction optimization (CRO) and the estimation of distribution (EDA), for solving the hybrid flow shop (HFS) scheduling problems. The objective is to minimize the maximum completion time. In the proposed algorithm, a well-designed decoding mechanism is presented to schedule jobs with more flexibility. Meanwhile, considering the problem structure, eight neighborhood structures are developed. A kinetic energy sensitive neighborhood change approach is proposed to extract global information and avoid being stuck at the local optima. In addition, contrary to the fixed neighborhood set in traditional VNS, a dynamic neighborhood set update mechanism is utilized to exploit the potential search space. Finally, for the population of local optima solutions, an effective EDA-based global search approach is investigated to direct the search process to promising regions. The proposed algorithm is tested on sets of well-known benchmark instances. Through the analysis of experimental results, the high performance of the proposed HVNS algorithm is shown in comparison with four efficient algorithms from the literature.

Introduction

In modern manufacturing and production systems, production scheduling plays an important role in increasing production efficiency and profit. In recent years, the flow shop scheduling problem (FSSP) has been investigated by more and more researchers because of its important role in a realistic production systems. The hybrid flow shop (HFS) scheduling problem, as one branch of the classical FSSP, is more complex because of the addition of machine selection. In 1988, Gupta proved that HFS is a NP-hard problem [1]. Recent and comprehensive reviews on HFS can be found in [2,3], which illustrate the published HFS literature before 2010. It can be concluded from the two literature reviews that: (1) there are more than 200 current papers discussing the problem; (2) more and more algorithms, including exact algorithms, dispatching rules, heuristics, and meta-heuristics, have been used for solving HFS; and (3) HFS has played an important role in present production systems. For instance, in most production systems with a flexible scheduling environment, such as the oil food, paper, textile, chemical and cosmetic industries, HFS plays a key role and becomes an important factor that can lead to improvements in production efficiency.

The branch and bound (B&B) algorithm proposed for solving the HFS was published in 1970 (Rao [4]). After that, many published papers have discussed the HFS with many different algorithms. We can classify these algorithms by the stage size of the considered problems. There are three types of problem, that is, two-stage, three-stage, and *m*-stage. The two-stage problem is the HFS with two consecutive stages, while the *m*-stage problem is the *m* stage series.

To solve the two-stage HFS, in 1988 Gupta studied an HFS for which there is only one machine at the second stage [1]. In 1991, the B&B approach was introduced by Brash and Hunsucker [5]. After that, many different types of approaches have been developed. In 1994, Gupta solved the two-stage HFS problem with separable setup and removal times [6]. In 2003, Lin and Liao [7] studied the two-stage HFS problem with setup time and dedicated machines. Lee and Kim [8] introduced the B&B approach to solve the two-stage HFS with any number of identical parallel machines at the second stage. Haouari et al. [9] considered the problem without any machine number constraints at any stage.

* Corresponding author at: State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, PR China. Tel.: +8618606356701.
E-mail addresses: lijunqing.cn@gmail.com, p2p_jql@126.com (J.-q. Li), panquanke@mail.neu.edu.cn (Q.-k. Pan).

http://dx.doi.org/10.1016/j.asoc.2014.07.005
1568-4946/

To solve the three-stage HFS, in 1998 Riane et al. [10] developed an efficient heuristic to minimize makespan in a three-stage HFS problem. Jin et al. [11] modeled a real printed circuit board manufacturing system by a HFS with three stages, and solved the problem by GA. Carlier and Neron [12] proposed an exact algorithm for solving the multi-processor flow shop. The benchmark problems generated by them were used by many studies as test problems. In 2004, Babayan and He [13] presented an agent-based approach for a three-stage HFS with identical parallel machines.

To solve the HFS with *m*-stages, in 1998 Portmann et al. [14] introduced an enhanced version of the branch and bound algorithm crossed with GA. In 2001, Neron et al. [15] proposed an algorithm using energetic reasoning and global operations for an HFS with up to five stages. The HFS with multiprocessor task problems was studied by Oguz and Ercan [16] with a GA approach. Ruiz and Maroto [17] developed a GA for HFS with sequence dependent setup times and machine eligibility. Janiak et al. [18] applied three constructive algorithms and three meta-heuristics based on Tabu Search (TS) and SA, to solve the HFS with cost-related criteria. Niu et al. [19] developed a quantum-inspired immune algorithm for HFS. In 2010, Kahraman et al. [20] investigated a parallel greedy algorithm for solving the multistage HFS. Engin et al. [21] developed an efficient GA for an HFS with multiprocessor task problems. In 2012, Liao et al. [22] proposed an approach using PSO and bottleneck heuristic for the problem. The other meta-heuristics were also used for solving HFS with multi-stages, such as ant colony optimization (ACO) [23,24] and artificial immune systems (AIS) [25,26].

Variable Neighborhood Search (VNS) is a recent and effective meta-heuristic for solving both continuous and global optimization problems. By systematic changes of the neighborhood structures within the search, VNS is capable of escaping from the local optima [27–43]. By simulating the behavior of molecules in chemical reactions, an efficient chemical-reaction optimization (CRO) algorithm was proposed by Lam and Li [44] to optimize combinatorial problems. In recent years, CRO has been applied to solve many continuous and discrete optimization problems [44–46]. The estimation of distribution algorithm (EDA) was introduced by Mühlenbein and Paaß [47]. In EDA, global statistical information was extracted from the current population. Meanwhile, a probabilistic model was constructed that represents the global information and characterizes the distribution of promising solutions in the search region [47–50].

In this study, we make a hybridization of the VNS, CRO, and EDA. To enhance the exploration ability of the proposed algorithm, we assign a certain *KE* value for each individual solution. A kinetic-energy-based neighborhood change procedure is introduced that can avoid being stuck at the local optima and thus further increase the exploration ability. In addition, after collecting a group of local optima solutions by a VNS process, an EDA-based global search process is performed on the local optima population to search global optimal solutions, which can further enhance the exploitation and exploration ability of the proposed algorithm. The rest of this paper is organized as follows: The Problem Formulation deals with the job/task allocation details together with the stage by stage work flow. Next The Related Algorithms deal with how VNS was construed by others. Then the technical approach is briefly discussed in the The Proposed HVNS algorithm. Further, The Experimental Results provide a fair account of results derived and compare them with other algorithms in the literature to demonstrate the performance of HVNS. Finally, Conclusions provide a concise description of solving the hybrid flow shop scheduling problem of our work.

Problem formulation

In an HFS, there are *n* jobs to be processed on *m* machines in a predefined order. The *m* machines are grouped into *k* stages in series. In each stage *i*, there are m_i identical machines in parallel, where $m_i \geq 1$, and there are at least two parallel machines in one stage. Each job should visit each stage following the same production flow: stage_1, stage_2, ..., stage_k. When a job arrives at a stage *i*, it can select exactly one machine from m_i available identical machines. The assumptions for the considered HFS in this study are given as follow.

- Each machine in the same stage can process only one job at a time;
- Each job can be operated by only one machine at a time;
- All of the jobs and machines are available at time zero;
- Preemption is not allowed, that is, any job cannot be interrupted before the completion of its current operation;
- Setup times are negligible and problem data are deterministic and known in advance;
- Overlap of operations is not allowed, that is, each operation of the same job should not start until the previous operation has completed its work;
- There is an unlimited intermediate buffer between two successive stages.

The aim of HFS is to assign an optimal machine for each job at each stage, and schedule each job on each machine to minimize the makespan or the maximum completion time. The mathematical model can be found in [2,17,22].

Related algorithms

The canonical VNS

In 1997, VNS was introduced by Mladenovic and Hansen [27] for solving the traveling salesman problem (TSP). After that, VNS has been applied by more and more researchers to solve continuous and global optimization problems. In 2010, Hansen et al. [28] gave a comprehensive survey of VNS. The survey literature shows that since it was first proposed in 1997, VNS has seen rapid development and enhancement in two aspects, i.e., methods and applications.

Very recently, VNS has been the focus of substantial research. The recent developments with VNS are also taken in two fields: (1) designing improved VNS by combining other dispatching rules, heuristics, meta-heuristics, and local search methods and (2) applying VNS for new applications. In 2009, Wang and Tang [29] developed a population-based VNS for the single machine total weighted tardiness problem. By combining VNS with GA, Wen et al. [30] presented a hybrid genetic based VNS for task scheduling in a heterogeneous multiprocessor system. Stenger et al. [31] investigated a routing problem arising in the last-mile delivery of small packages by using an improved VNS that embeds cyclic-exchange neighborhoods and an adaptive mechanism. Yazdani et al. [32] developed a parallel VNS for the flexible job-shop scheduling problem (FJSP). Recently, more and more literature has addressed solving scheduling problems using VNS. For parallel machine scheduling problems, Behnamian et al. [33] proposed a hybrid algorithm using ACO, SA and VNS; Driessel et al. [34] developed several variants of VNS; in 2012, Bilyk and Mönch [35] investigated VNS for planning and scheduling of jobs on unrelated parallel machines. For HFS problems utilizing VNS, Naderi et al. [36] proposed a VNS that uses advanced neighborhood search structures for flexible flow line problems with sequence dependent setup time and preventive maintenance activity; Tavakkoli-Moghaddam et al. [37] presented a memetic algorithm for HFS with processor blocking. The other recent applications of VNS include vertex separation problems [38], resource allocation problems [39], TSP [40], location

routing [41], multilevel lot-sizing problems [42], and facility layout problems [43].

The basic VNS employs a set of pre-defined neighborhoods. By sequentially exploring these neighborhoods, local optima solutions in different neighborhood structure can be obtained, and thus, better solutions can be reached through this process. In VNS, one solution in the current neighborhood is selected as the incumbent solution. Then, local search is performed on the selected solution to generate several neighboring solutions. By comparing the neighboring solutions with the incumbent one, the current solution will be replaced by the best solution found thus far or remain its state if no better solution is found. Then, the neighborhood will turn to the first one if a better solution is found or turn to the next one if the incumbent solution remains itself. By systematically changing the current neighborhood, VNS directs the search to a promising field, and thus, global optimal solutions will be found. The main steps of the basic VNS are given as follows [27]:

Step 1. Set system parameters that include the set of neighborhood structure N_k, $k = 1, \ldots, k_{\max}$, and an initial solution x;
Step 2. Repeat the following steps until the stop condition is satisfied.
Step 3. Generate a solution x^T at random from the kth neighborhood of x;
Step 4. Apply some local search methods with x' as the initial solution; denote x'' as the best obtained solution;
Step 5. If the obtained solution x'' is better than the incumbent, replace it with the former and continue the search with the first neighborhood structure, that is, set $k = 1$; otherwise, set $k = k + 1$.

The canonical CRO

In 2010, Lam and Li [44] introduced CRO, which loosely mimics what happens to molecules in a chemical reaction system and tries to capture the energy in the reaction process. The molecules represent the solution for the considered problem, while the change in molecular structure is tantamount to switching to another feasible solution. Each molecule possesses two types of energies, i.e., potential energy (*PE*) and kinetic energy (*KE*). *PE* corresponds to the objective function of a molecule, while the *KE* of a molecule symbolized its ability to escape from a local minimum. In the basic CRO, there are four elementary reactions, i.e., the on-wall ineffective collision, decomposition, inter-molecular ineffective collision, and synthesis. Meanwhile, CRO has a buffer to collect the kinetic energy produced by on-wall ineffective collisions and to help the molecules to escape from the local optima.

The main steps of the basic CRO are given as follows [44–46]:

Step 1. Set system parameters;
Step 2. Perform the following steps until the stop condition is satisfied.
Step 3. If the inter-molecular condition is satisfied, perform step 4; otherwise, perform step 5.
Step 4. Randomly select two or more molecules from the current population; if the synthesis condition is satisfied, perform synthesis operation on the selected two molecules; otherwise, perform inter-molecular collision on them;
Step 5. Randomly select one molecule; if the decomposition condition is satisfied, perform decomposition operation on the selected individual; otherwise, perform on-wall collision on it.

The canonical EDA

In 1996, Mühlenbein and Paaß [47] proposed a new evolutionary algorithm, called the estimation of distribution algorithm (EDA). Unlike other meta-heuristics, such as the genetic algorithm (GA) and particle swarm optimization (PSO), EDA uses a probabilistic model to utilize global statistical information to generate new offspring, which is very different from crossover or mutation operators.

The main steps of the basic EDA are given as follows [47–50]:

Step 1. Set system parameters.
Step 2. Randomly generate a population of initial solutions.
Step 3. Evaluate each solution, and select good individuals with respect to their fitness.
Step 4. Generate a new distribution of probability based on the selected individuals.
Step 5. Generate new offspring from the estimated distribution.

Step 6. If a stop condition is satisfied, then stop the algorithm; otherwise, go back to step 3.

The proposed algorithm

Solution representation

In the proposed algorithm, each solution is represented by a string of integers that are commonly used in most of the literature for HFS [2,17,50]. Each integer in the string corresponds to a job number. Thus, the length of the string equals to n. Consider an example problem with three stages and five jobs, the parallel machine in each stage and the processing time of each operation is given in Table 1. Suppose that one solution is represented by {2, 4, 3, 1, 5}, which means that in the first stage, then the scheduling sequence is J_2, J_4, J_3, J_1, and J_5.

Decoding

It is notable that the solution encoding given above contains no machine selection (routing) information in each stage. Therefore, we should consider both machine selection and operation scheduling in the decoding process. The common method is as follows [2,17,50]: (1) in the first stage, schedule each job according to their sequence in the solution representation, and assign each job to the first available machine; (2) in the following stages, assign the first available machine for the arriving job.

In this study, to decode the solution representation in all of the stages, except the first one, we introduce two decoding methods as follows.

- Solution-Dependent Rule, denoted by SDR. In the SDR rule, the concurrently arriving jobs will be scheduled according to their occurrence order in the solution representation. It should be noted that, the SDR rule is commonly used in the current literature [2,17,50]. However, by using the SDR rule, the concurrently arriving jobs will be scheduled in a relatively deterministic order at each stage. For the given example in Solution representation, when the two jobs J_4 and J_1 arrive at the second stage at the same time, that is, with the same completion time at the first stage, if only one available machine is waiting for scheduling them, then J_4 will be scheduled first according to the SDR rule;
- Random Rule, denoted by RR. In the RR rule, the jobs will be scheduled in a random order. The detailed steps are given as follows. First, at each stage, except the first, sort each job into a set R_s in a random way. Second, when more than one job arrives at a stage concurrently and there are not enough available machines for them, then schedule them according to their order in R_s. For the given example in Solution representation, when J_4 and J_1 arrive at the second stage concurrently while only one machine is waiting for them, then J_4 and J_1 will be scheduled in a random way.

Table 1
Processing time table.

	Stage1 (two parallel machines)		Stage2	Stage3 (two parallel machines)	
	M1	M2	M3	M4	M5
J1	7	7	3	3	3
J2	1	1	2	3	3
J3	3	3	5	8	8
J4	3	3	3	6	6
J5	6	6	8	3	3

It should be noted that the proposed RR method considers the following realistic production characteristics: (1) there is usually more than one job arriving concurrently at the same stage; (2) the number of available machines is usually less than the number of concurrent arriving jobs. By using the decoding heuristics discussed above, the detailed decoding process is given as follows:

Step 1: In the first stage, schedule each job one by one according to their sequence in the solution representation. For the above example solution listed in Solution representation, in the first stage, the job scheduling sequence is J_2, J_4, J_3, J_1, and J_5. When one job is to be scheduled, it will select the earliest available machine. If there are several available machines with the same available time, the job will randomly select one.
Step 2: In the other stages, each job will be scheduled as soon as it completes its previous operation. The machine selection strategy is the same as in the first stage, that is, to select the earliest available machine for the arrived operation. If more than one parallel available machine is available, then randomly select one from them. When several jobs arrive concurrently at the same time with too few available machines at the stage, we use the RR rule.

Fig. 1 gives the Gantt chart for the above example solution listed in Solution representation. In Fig. 1, each operation is represented by a pair of number, i.e., the job number and the stage number. For example, in Fig. 1, the first operation scheduled on the machine M_1 is denoted by (2, 1), which denotes that the job J_2 in the first stage is scheduled first on M_1. In the second stage, there is only one machine, and thus, each job is scheduled one by one on M_3. The last completed job is J_5 on M_4, with the completion time 25. Therefore, the maximum completion time (makespan) of the given solution is 25.

Neighborhood structure

In this study, considering the problem structure, eight neighborhood structures are proposed, which are given as follows:

- Swap two positions structure, denoted by N_1. (1) Randomly select two job numbers in the solution representation; (2) Swap the two selected jobs in the scheduling string.
- Insertion structure, denoted by N_2. (1) Randomly select two positions, r_1 and r_2, in the solution representation, where $r_1 < r_2$; (2) Insert the job at the position r_2 before r_1 in the scheduling string.
- Reverse the elements between two positions structure, denoted by N_3. (1) Randomly select two positions, r_1 and r_2, in the solution representation. (2) Reverse the job numbers between the positions r_1 and r_2 in the scheduling string.
- Swap elements structure, denoted by N_4, with an example given in Fig. 2(a). (1) Randomly select two positions, r_1 and r_2, in the solution representation, where $r_1 < r_2$; (2) Swap each pair of elements between the positions r_1 and r_2 in the scheduling string. The detailed steps are as follows: (a) let $p = r_1$, and $q = r_2$; (b) swap the two elements p and q in the solution representation; let $p = r_1 + 1$, and $q = r_2 - 1$; (3) repeat step (b) until $p \geq q$.

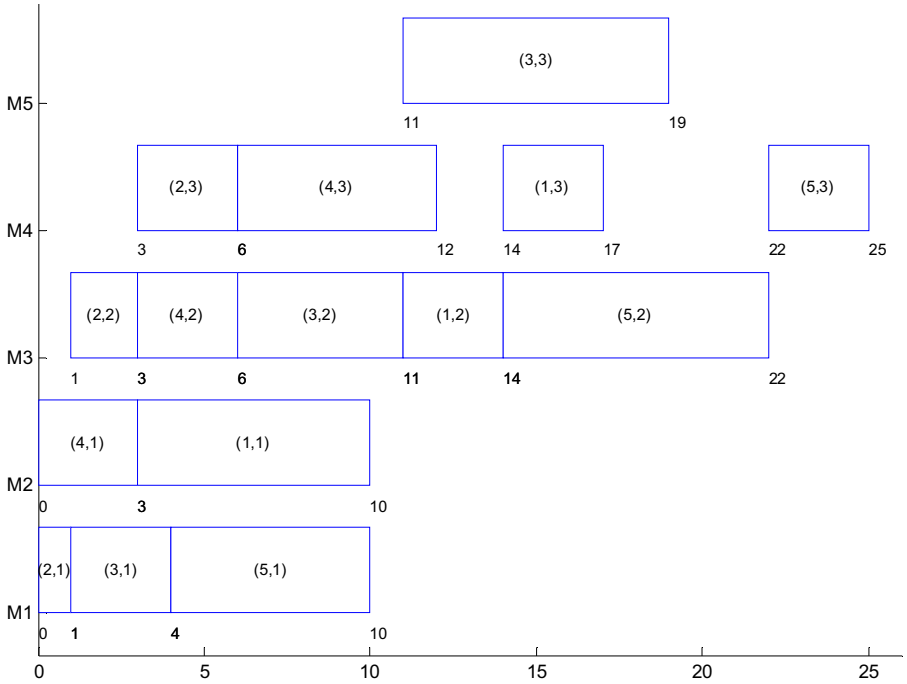
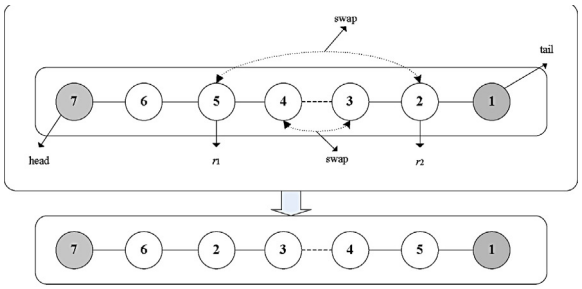
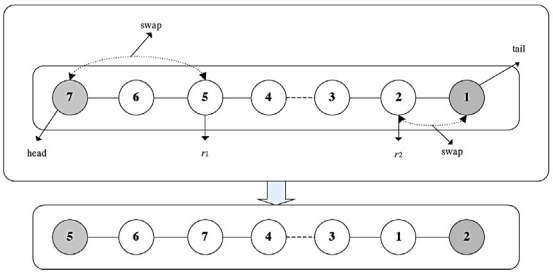


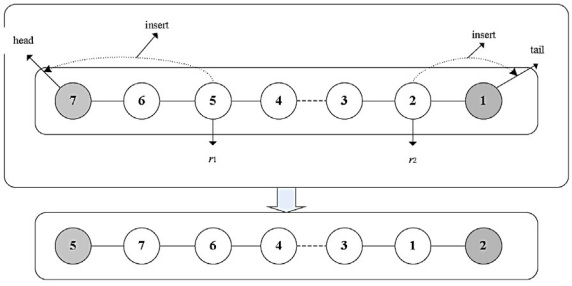
Fig. 1. Gantt chart for the example solution.



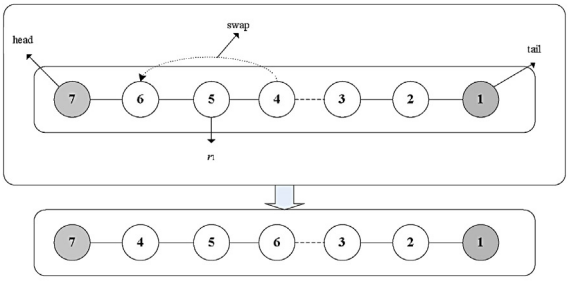
(a) N_4



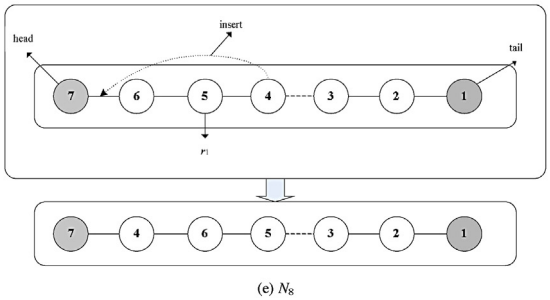
(b) N_5



(c) N_6



(d) N_7



(e) N_8

Fig. 2. Eight neighborhoods.

- Swap the elements with the first and the last element structure, denoted by N_5, with an example given in Fig. 2(b). (1) Randomly select two positions, r_1 and r_2, in the solution representation, where $r_1 < r_2$; (2) swap the element at the positions r_1 with the head element, and swap the element at the positions r_2 with the tail element.
- Insert elements before the first and after the last structure, denoted by N_6, with an example given in Fig. 2(c). (1) Randomly select two positions, r_1 and r_2, in the solution representation, where $r_1 < r_2$; (2) Insert the element at the positions r_1 before the head position, and insert the element at the positions r_2 after the tail position.
- Swap elements at one point structure, denoted by N_7, with an example given in Fig. 2(d). (1) Randomly select one position, r_1, in the solution representation, except the head position and the tail position. (2) Swap the two elements before and after the positions r_1.
- Insert at one point structure, denoted by N_8, with an example given in Fig. 2(e). (1) Randomly select one position, r_1, in the solution representation, except the head position and the tail position; (2) Insert the element after the position r_1 at the position just before r_1.

KE based neighborhood change procedure

We assign a certain *KE* value for each individual solution and introduce a kinetic-energy-based neighborhood change procedure (*KE_NeighborhoodChange*), which updates the incumbent solution and the *KE* value for the current solution. By using the kinetic energy sensitive procedure, the proposed algorithm can avoid being stuck at the local optima, and thus, increase the exploitation and exploration ability.

In VNS, if a better neighboring solution is found, the neighborhood index k will be set to the first position, which makes the next neighborhood to be accessed potentially different from the current one. A disadvantage is that the search process will jump to other search regions even when it finds a better solution in the current neighborhood. In contrast to the traditional VNS, the neighborhood set is updated dynamically in the proposed algorithm by inserting the promising neighborhood into the first position. That is, if a better solution is found in the current neighborhood, the corresponding neighborhood will be accessed again in the following search process until no better solution is obtained. Therefore, the first neighborhood has a higher probability of being accessed. The above process makes the algorithm completely exploit the promising search region until local optima are reached.

The detailed steps of the proposed procedure are given in Fig. 3.

EDA-based global search

In this study, after collecting a group of local optima solutions by the VNS process, an EDA-based global search process will be performed on the local optima population to search for global optimal solutions. The probabilistic model is the main issue for EDA and the performance of the algorithm is strongly dependent on it; therefore, the best choice of model is crucial. In general, two key issues should be considered for the probabilistic model of EDA, i.e., the building of the model, and the update of the model.

Building of the probabilistic model

To construct a probabilistic model for the current population solutions, we first select B_{size} solutions with relatively low fitness values. Let t be the generation index, $n_{ij}(t)$ be the number of times of appearance of job i before or in the position j in the subset of the selected B_{size} solutions in generation t. The value of $n_{ij}(t)$ refers to the importance of the order of the jobs in the scheduling sequence,

Procedure *KE_NeighborhoodChange*(x, x'', k, KE_x, $KE_{lossRate}$, Γ)

Input: the best neighboring solution x''; the incumbent solution x; neighborhood index k; *KE* value for x KE_x; loss rate $KE_{lossRate}$; neighborhood set Γ.

Output: the updated states, include x, k, KE_x, and Γ.

Begin

(1) if $f(x'') < f(x) + KE_x$, then

$x \leftarrow x''$;

$k \leftarrow 1$;

$KE_x \leftarrow KE_x \times (1 - KE_{lossRate})$;

Insert the neighborhood $\Gamma(k)$ into the first position in Γ.

(2) else

(3) $k \leftarrow k + 1$;

End

Fig. 3. Pseudo code of *KE_NeighborhoodChange*.

which indicates the probability of the global statistical information. Let $p_{ij}(t)$ be the probability for job i to appear before or at the position j, which means that from the global statistical point, job i should be scheduled no later than position j. Let $(p_{i1}(t), p_{i2}(t), \ldots, p_{in}(t))$ be the probabilistic model for each element of job i in generation t. Then, we can obtain the completed probabilistic model as follows.

$$P(t) = \begin{bmatrix} p_{11}(t) & p_{12}(t) & \ldots & p_{1n}(t) \\ p_{21}(t) & p_{22}(t) & \ldots & p_{2n}(t) \\ \ldots & \ldots & \ldots & \ldots \\ p_{n1}(t) & p_{n2}(t) & .. & p_{nn}(t) \end{bmatrix} \quad (1)$$

The construction process for $p_{ij}(t)$ is given as follows, similar to [49].

$$p_{ij}(0) = \frac{\sum_{k=1}^{B_{size}} \chi_{ij}^{k}(0)}{i \cdot B_{size}} \quad (2)$$

The update process for $p_{ij}(t)$ is given as follows, similar to [49].

$$p_{ij}(t+1) = \alpha \times \frac{\sum_{k=1}^{B_{size}} \chi_{ij}^{k}(t+1)}{i \cdot B_{size}} + (1-\alpha) \times p_{ij}(t) \quad (3)$$

where, α is the study rate, which is range between 0 and 1;

$$\chi_{ij}^{k}(t) = \begin{cases} 1 & \text{If job } i \text{ appears before or in the position } j \text{ in solution } k \\ 0 & \text{Otherwise} \end{cases} \quad (4)$$

Procedure of the global search

The EDA based global search process is given in Fig. 4.

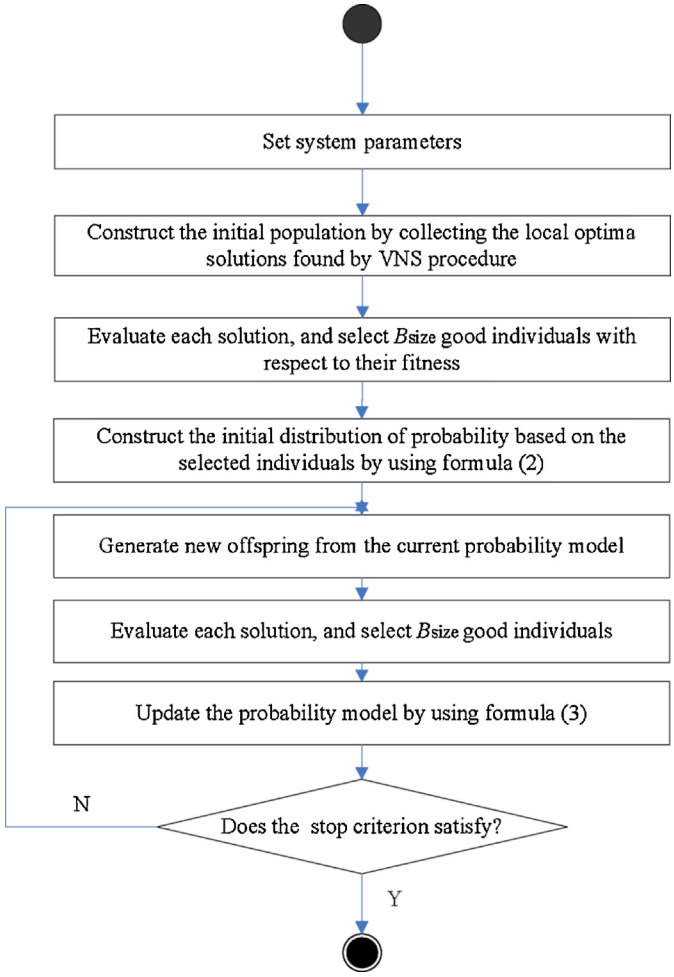


Fig. 4. Flow chart of the EDA-based global search.

The framework of HVNS

The detailed steps of the proposed HVNS algorithm are as follows:

***Step* 1:** Initialization phase.

Step 1.1 Set the population size P_{size}.

Step 1.2 Initialize the population in a random way. That is, randomly sequence each job in the solution representation for each initial individual.

***Step* 2**: Evaluate each solution in the population. Select the best solution as the current incumbent individual.

***Step* 3**: If the stopping criterion is satisfied, output the best solution; otherwise, perform steps 4–6.

***Step* 4:** VNS local search phase.

Step 4.1 Let $t = 1$, and repeat steps 4.2–4.7 until $t > t_{max}$.

Step 4.2 Let $k = 1$, and repeat steps 4.3–4.6 until $k > k_{max}$.

Step 4.3 Generate an initial neighboring solution x' around the incumbent solution in the kth neighborhood.
Step 4.4 Local search phase: Generate N_s neighboring solutions, select the best one as x''.
Step 4.5 Neighborhood change phase: By using the proposed neighborhood change procedure *KE_NeighborhoodChange* as illustrated in Fig. 3, update the current incumbent solution and the value of k.
Step 4.6 Store every local optima solution for each neighborhood structure into a set *LO*.
Step 4.7 Let $t = t + 1$.
***Step* 5**: EDA based global search phase.
Step 5.1 From the local optima solutions set *LO*, which is generated in step 4.6, select B_{size} best solutions to construct the initial population for the EDA based global search procedure.
Step 5.2 Construct the initial probabilistic model based on the selected initial population.
Step 5.3 Generate B_{size} new solutions based on the initial population and the initial probabilistic model.
Step 5.4 If the stop condition satisfies, go to step 6; otherwise, perform steps 5.5–5.6.
Step 5.5 Update the probabilistic model by using formula (3).
Step 5.6 Generate B_{size} new solutions by the new estimation distribution model. Evaluate each solution and go back to step 5.4.
***Step* 6**: Go back to step 3.

Experimental results

This section discusses the computational experiments used to evaluate the performance of the proposed algorithm. Our algorithm was implemented in C++ on an Intel Core i5 3.3 GHz PC with 4 GB of memory. The compared algorithms include PSO (Liao et al. [22]), AIS (Engin and Doyen [25]), ACO (Alaykyran et al. [24]), and B&B (Neron et al. [15]).

To present a detailed comparison with the present algorithms, we also select Carlier and Neron's benchmark problems [15] as the test instances. In the given 77 benchmark problems, there are 41 10-job problems and 36 15-job problems. To test the proposed algorithm on large-scale HFS problems, we select the ten harder problems in [22]. Each benchmark problem is characterized by three numbers and three letters. The three numbers are the number of jobs, number of stages, and problem structure index. The problem structure index makes the benchmark with the same problem scale different from each other [15,22]. The three letters have the following meanings: *j* represents the job; *c* represents the production stage; and the third letter illustrates the parallel machine layout structure, which is given as follows [22,25].

(a) There is one machine at the middle stage and three machines at the other stages.
(b) There is one machine at the first stage and three machines at the other stages.
(c) There are two machines at the middle stage and three machines at the other stages.
(d) There are three machines at each stage.

Table 2
Combinations of the parameter values.

Parameter	Level			
	1	2	3	4
P_{size}	5	10	20	50
t_s	10	20	50	100
t_{max}	10	20	50	100

Parameters setting

To set the system parameter efficiently, we first conduct a preliminary experiment, where several typical values were set for each parameter by simply fixing others, and find three parameters (P_{size}, N_s, and t_{max}) with a significant impact. The other parameters with less significant impact are set as follows: (1) the maximum neighborhood structure number k_{max}: 8; (2) similar to reference [50], the parameters for EDA based global search are as follows: the number of best solutions $B_{size} = 5$, and the study rate for update process of the probabilistic model $\alpha = 0.3$; (3) similar to reference [44], the parameters for *KE* based local search are as follows: the lost rate for kinetic energy *KELossRate* = 0.5, and the initial *KE* value for each individual = 1,000,000; and (4) the maximum computational time is set to 100 s, or until the lowest bound (LB) is reached.

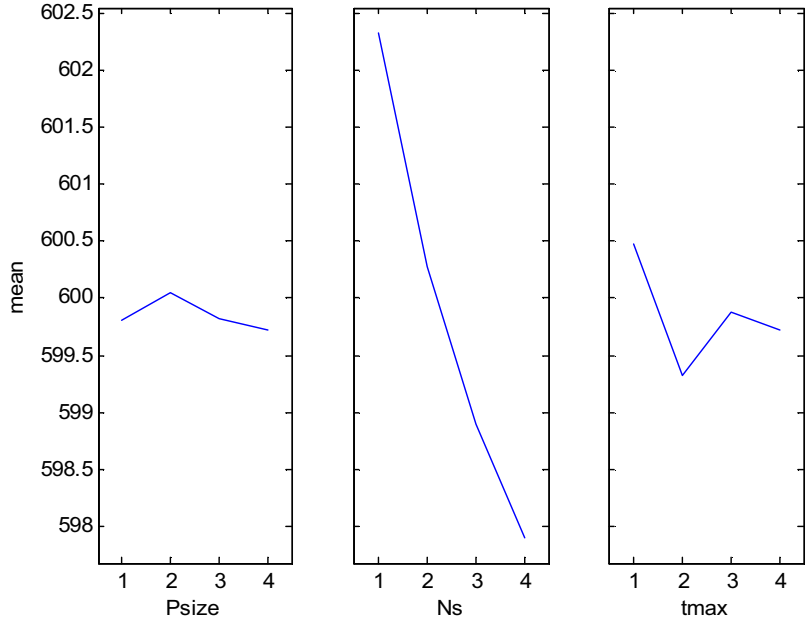


Fig. 5. Factor level trend.

For the determination of the three significant parameters, we conduct a detailed experiment based on the Taguchi method of DOE [51] method. The levels of the three parameters are given in Table 2. The tested instances are the ten harder problems, from "j30c5e1" to "j30c5e10". Because the three parameters are set with four factor levels, an orthogonal array $L_{16}(4^3)$ is selected. For each parameter combination, the proposed algorithm is independently run 20 times, and then, the average makespan value obtained by the proposed algorithm is collected as the response variable (RV). Fig. 5 reports the factor level trend for the three parameters. It can be seen from Fig. 5 that the proposed algorithm has a better performance under the three parameters with the following values: the initial population size $P_{size} = 50$, the number of neighboring solutions during the local search: $N_s = 100$, and the maximum number of iterations for the VNS search process: $t_{max} = 20$.

Effectiveness of different decoding heuristics

In this section, we investigate the effectiveness of decoding heuristics presented in Decoding. Let D_NH represents the decoding process without the RR rule, which is commonly used in most literature [2,17,50]. D_H corresponds to the decoding process utilizing the RR rule, which is used in the proposed HVNS algorithm. We create both D_NH and D_H in Visual C++6.0 and use the same library functions and objective evaluation functions. The comparison is based on the 10 harder instances, from "j30c5e1" to "j30c5e10". The performance measure is relative percentage increase (RPI) calculated as follows:

$$\mathrm{RPI}(F) = \frac{C^c_{max} - C_{best}}{C_{best}} \times 100 \quad (5)$$

where, C^c_{max} is the makespan obtained by the compared heuristic, and C_{best} is the best value collected by the two compared heuristics. The two algorithms are independently run 20 times in the same environment. Then, the RPI results, averaged across the 20 replications for each instance, are collected for comparison in Table 3.

It can be seen from Table 3 that the proposed D_H performs much better than the D_NH heuristic because D_NH generates an overall average RPI of 0.26%, whereas D_H produces a zero RPI value for each of the ten instances. To determine whether the observed difference from Table 3 is indeed significantly different, we carry out a single-factor analysis of variance (ANOVA) where the type of decoding heuristic is considered to be a factor. The

Table 3
Comparison results for the RPI values of different decoding heuristic.

Instance	D_NH	D_H
j30c5e1	0.86	0.00
j30c5e2	0.00	0.00
j30c5e3	0.17	0.00
j30c5e4	0.53	0.00
j30c5e5	0.00	0.00
j30c5e6	0.00	0.00
j30c5e7	0.00	0.00
j30c5e8	0.00	0.00
j30c5e9	0.31	0.00
j30c5e10	0.69	0.00
Mean	0.26	0.00

Table 4
Comparison results for the RPI values of the four versions of VNS.

Instance	HVNS	VNS-I	VNS-II	VNS-III
j10c5c1	0.00	0.00	0.00	0.00
j10c5c2	0.00	1.35	1.35	0.00
j10c5c3	1.41	1.41	1.41	1.41
j10c5c4	0.00	1.52	0.00	0.00
j10c5c5	0.00	0.00	0.00	0.00
j10c5c6	0.00	0.00	0.00	0.00
j10c5d1	0.00	0.00	0.00	0.00
j10c5d2	0.00	1.37	1.37	1.37
j10c5d3	0.00	0.00	0.00	0.00
j10c5d4	0.00	0.00	0.00	0.00
j10c5d5	0.00	1.52	0.00	1.52
j10c5d6	0.00	0.00	0.00	0.00
j15c5c1	0.00	0.00	0.00	1.18
j15c5c2	0.00	1.11	1.11	1.11
j15c5c3	0.00	0.00	0.00	0.00
j15c5c4	0.00	0.00	0.00	0.00
j15c5c5	1.37	4.11	2.74	4.11
j15c5c6	0.00	0.00	0.00	0.00
j15c5d1	0.00	0.00	0.00	0.00
j15c5d2	2.44	3.66	2.44	3.66
j15c5d3	6.49	9.09	7.79	7.79
j15c5d4	37.70	39.34	39.34	39.34
j15c5d5	17.91	20.90	19.40	19.40
j15c5d6	2.53	3.80	2.53	3.80
Mean	2.911	3.715	3.312	3.529

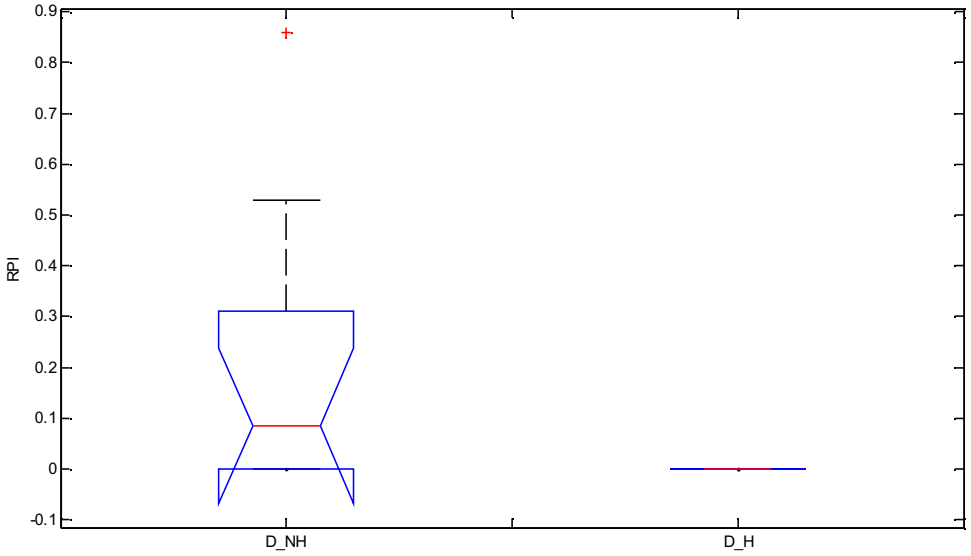


Fig. 6. Means and 95% LSD interval for D_NH and D_H (p-value: 0.0324).

Table 5
Comparison results for 10-jobs problems.

Instance	LB	HVNS		PSO		AIS		ACO		B&B		RPI				
		C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	HVNS	PSO	AIS	ACO	B&B
j10c5a2	88	88	0.004	88	0.002	88	1	88	–	88	13	0.00	0.00	0.00	0.00	0.00
j10c5a3	117	117	0.004	117	0.002	117	1	117	–	117	7	0.00	0.00	0.00	0.00	0.00
j10c5a4	121	121	0.002	121	0.003	121	1	121	–	121	6	0.00	0.00	0.00	0.00	0.00
j10c5a5	122	122	0.003	122	0.013	122	1	124	–	122	11	0.00	0.00	0.00	1.64	0.00
j10c5a6	110	110	0.024	110	0.174	110	4	110	–	110	6	0.00	0.00	0.00	0.00	0.00
j10c5b1	130	130	0.003	130	0.003	130	1	131	–	130	13	0.00	0.00	0.00	0.77	0.00
j10c5b2	107	107	0.003	107	0.003	107	1	107	–	107	6	0.00	0.00	0.00	0.00	0.00
j10c5b3	109	109	0.003	109	0.012	109	1	109	–	109	9	0.00	0.00	0.00	0.00	0.00
j10c5b4	122	122	0.004	122	0.025	122	2	124	–	122	6	0.00	0.00	0.00	1.64	0.00
j10c5b5	153	153	0.003	153	0.001	153	1	153	–	153	6	0.00	0.00	0.00	0.00	0.00
j10c5b6	115	115	0.002	115	0.001	115	1	115	–	115	11	0.00	0.00	0.00	0.00	0.00
j10c5c1	68	68	0.792	68	0.332	68	32	68	–	68	28	0.00	0.00	0.00	0.00	0.00
j10c5c2	74	74	0.752	74	0.535	74	4	76	–	74	19	0.00	0.00	0.00	2.70	0.00
j10c5c3	71	72	0.141(a)	71	36.997	72	a	72	–	71	240	1.41	0.00	1.41	1.41	0.00
j10c5c4	66	66	0.631	66	0.215	66	3	66	–	66	1017	0.00	0.00	0.00	0.00	0.00
j10c5c5	78	78	0.129	78	0.122	78	14	78	–	78	42	0.00	0.00	0.00	0.00	0.00
j10c5c6	69	69	0.226	69	0.405	69	12	69	–	69	4865	0.00	0.00	0.00	0.00	0.00
j10c5d1	66	66	0.148	66	0.185	66	5	–	–	66	6490	0.00	0.00	0.00	–	0.00
j10c5d2	73	73	0.142	73	1.158	73	31	–	–	73	2617	0.00	0.00	0.00	–	0.00
j10c5d3	64	64	0.157	64	0.098	64	15	–	–	64	481	0.00	0.00	0.00	–	0.00
j10c5d4	70	70	0.675	70	0.337	70	5	–	–	70	393	0.00	0.00	0.00	–	0.00
j10c5d5	66	66	1.382	66	0.515	66	1446	–	–	66	1627	0.00	0.00	0.00	–	0.00
j10c5d6	62	62	0.135	62	0.383	62	8	–	–	62	6861	0.00	0.00	0.00	–	0.00
j10c10a1	139	139	0.227	139	0.055	139	1	–	–	139	41	0.00	0.00	0.00	–	0.00
j10c10a2	158	158	0.354	158	0.87	158	18	–	–	158	21	0.00	0.00	0.00	–	0.00
j10c10a3	148	148	0.225	148	0.017	148	1	–	–	148	58	0.00	0.00	0.00	–	0.00
j10c10a4	149	149	0.007	149	0.085	149	2	–	–	149	21	0.00	0.00	0.00	–	0.00
j10c10a5	148	148	0.006	148	0.102	148	1	–	–	148	36	0.00	0.00	0.00	–	0.00
j10c10a6	146	146	0.264	146	0.239	146	4	–	–	146	20	0.00	0.00	0.00	–	0.00
j10c10b1	163	163	0.006	163	0.013	163	1	163	–	163	36	0.00	0.00	0.00	0.00	0.00
j10c10b2	157	157	0.104	157	0.221	157	1	157	–	157	66	0.00	0.00	0.00	0.00	0.00
j10c10b3	169	169	0.006	169	0.014	169	1	169	–	169	19	0.00	0.00	0.00	0.00	0.00
j10c10b4	159	159	0.068	159	0.021	159	1	159	–	159	20	0.00	0.00	0.00	0.00	0.00
j10c10b5	165	165	0.005	165	0.037	165	1	165	–	165	33	0.00	0.00	0.00	0.00	0.00
j10c10b6	165	165	0.005	165	0.056	165	1	165	–	165	34	0.00	0.00	0.00	0.00	0.00
j10c10c1	113	115	0.571(a)	115	a	115	a	118	–	127	c	1.77	1.77	1.77	4.42	12.39
j10c10c2	116	117	0.448(a)	117	a	119	a	117	–	116	1100	0.86	0.86	2.59	0.86	0.00
j10c10c3	98	116	0.698(a)	116	a	116	a	108	–	133	c	18.37	18.37	18.37	10.20	35.71
j10c10c4	103	120	0.461(a)	120	a	120	a	112	–	135	c	16.50	16.50	16.50	8.74	31.07
j10c10c5	121	125	2.125(a)	125	a	126	a	126	–	145	c	3.31	3.31	4.13	4.13	19.83
j10c10c6	97	106	0.343(a)	106	a	106	a	102	–	112	c	9.28	9.28	9.28	5.15	15.46
Mean			0.275		1.236		47.735		–		719.4	1.26	1.22	1.32	1.44	2.79

Problems with bold font are harder problems.
(a) means LB is not reached.

Table 6
Comparison results for 15-job problems.

Instance	LB	HVNS		PSO		AIS		ACO		B&B		RPI				
		C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	C_{max}	CPU	HVNS	PSO	AIS	ACO	B&B
j15c5a1	178	178	0.006	178	0.06	178	1	178	–	178	18	0.00	0.00	0.00	0.00	0.00
j15c5a2	165	165	0.005	165	0.005	165	1	165	–	165	35	0.00	0.00	0.00	0.00	0.00
j15c5a3	130	130	0.005	130	0.006	130	1	132	–	130	34	0.00	0.00	0.00	1.54	0.00
j15c5a4	156	156	0.005	156	0.013	156	2	156	–	156	21	0.00	0.00	0.00	0.00	0.00
j15c5a5	164	164	0.003	164	0.004	164	1	166	–	164	34	0.00	0.00	0.00	1.22	0.00
j15c5a6	178	178	0.005	178	0.006	178	1	178	–	178	38	0.00	0.00	0.00	0.00	0.00
j15c5b1	170	170	0.005	170	0.003	170	1	170	–	170	16	0.00	0.00	0.00	0.00	0.00
j15c5b2	152	152	0.003	152	0.005	152	1	152	–	152	25	0.00	0.00	0.00	0.00	0.00
j15c5b3	157	157	0.006	157	0.03	157	1	157	–	157	15	0.00	0.00	0.00	0.00	0.00
j15c5b4	147	147	0.007	147	0	147	1	149	–	147	37	0.00	0.00	0.00	1.36	0.00
j15c5b5	166	166	0.088	166	0.086	166	2	166	–	166	20	0.00	0.00	0.00	0.00	0.00
j15c5b6	175	175	0.004	175	0.016	175	1	176	–	175	23	0.00	0.00	0.00	0.57	0.00
j15c5c1	85	85	1.128	85	4.205	85	774	85	–	85	2131	0.00	0.00	0.00	0.00	0.00
j15c5c2	90	90	0.491	90	1198	91	a	90	–	90	184	0.00	0.00	1.11	0.00	0.00
j15c5c3	87	87	0.502	87	2.398	87	16	87	–	87	202	0.00	0.00	0.00	0.00	0.00
j15c5c4	89	89	0.569	89	2.208	89	317	89	––	90	c	0.00	0.00	0.00	0.00	1.12
j15c5c5	73	74	1.968(a)	74	a	74	a	73	–	84	c	1.37	1.37	1.37	0.00	15.07
j15c5c6	91	91	0.180	91	0.191	91	19	91	–	91	57	0.00	0.00	0.00	0.00	0.00
j15c5d1	167	167	0.004	167	0	167	1	167	–	167	24	0.00	0.00	0.00	0.00	0.00
j15c5d2	82	84	0.915(a)	84	a	84	a	86	–	85	c	2.44	2.44	2.44	4.88	3.66
j15c5d3	77	82	1.356(a)	82	a	83	a	83	–	96	c	6.49	6.49	7.79	7.79	24.68
j15c5d4	61	84	1.325(a)	84	a	84	a	84	–	101	c	37.70	37.70	37.70	37.70	65.57
j15c5d5	67	79	0.585(a)	79	a	80	a	80	–	97	c	17.91	17.91	19.40	19.40	44.78
j15c5d6	79	81	1.123(a)	81	a	81	a	79	–	87	c	2.53	2.53	2.53	0.00	10.13
j15c10a1	236	236	0.008	236	0.018	236	1	236	–	236	40	0.00	0.00	0.00	0.00	0.00
j15c10a2	200	200	0.339	200	0.214	200	30	200	–	200	154	0.00	0.00	0.00	0.00	0.00
j15c10a3	198	198	0.313	198	0.171	198	4	198	–	198	45	0.00	0.00	0.00	0.00	0.00
j15c10a4	225	225	0.212	225	0.072	225	12	228	–	225	78	0.00	0.00	0.00	1.33	0.00
j15c10a5	182	182	0.008	182	0.509	182	2	182	–	183	c	0.00	0.00	0.00	0.00	0.55
j15c10a6	200	200	0.024	200	0.468	200	2	200	–	200	44	0.00	0.00	0.00	0.00	0.00
j15c10b1	222	222	0.009	222	0.017	222	3	222	–	222	70	0.00	0.00	0.00	0.00	0.00
j15c10b2	187	187	0.008	187	0.012	187	1	188	–	187	80	0.00	0.00	0.00	0.53	0.00
j15c10b3	222	222	0.008	222	0.007	222	1	224	–	222	80	0.00	0.00	0.00	0.90	0.00
j15c10b4	221	221	0.008	221	0.007	221	1	221	–	221	84	0.00	0.00	0.00	0.00	0.00
j15c10b5	200	200	0.045	200	0.135	200	1	–	–	200	84	0.00	0.00	0.00	–	0.00
j15c10b6	219	219	0.009	219	0.006	219	1	–	–	219	67	0.00	0.00	0.00	–	0.00
Mean	–	–	0.313	–	40.296	–	41.379	–	–	–	133.571	1.90	1.90	2.01	2.27	4.60

Problems with bold font are harder problems.
(a) means LB is not reached.

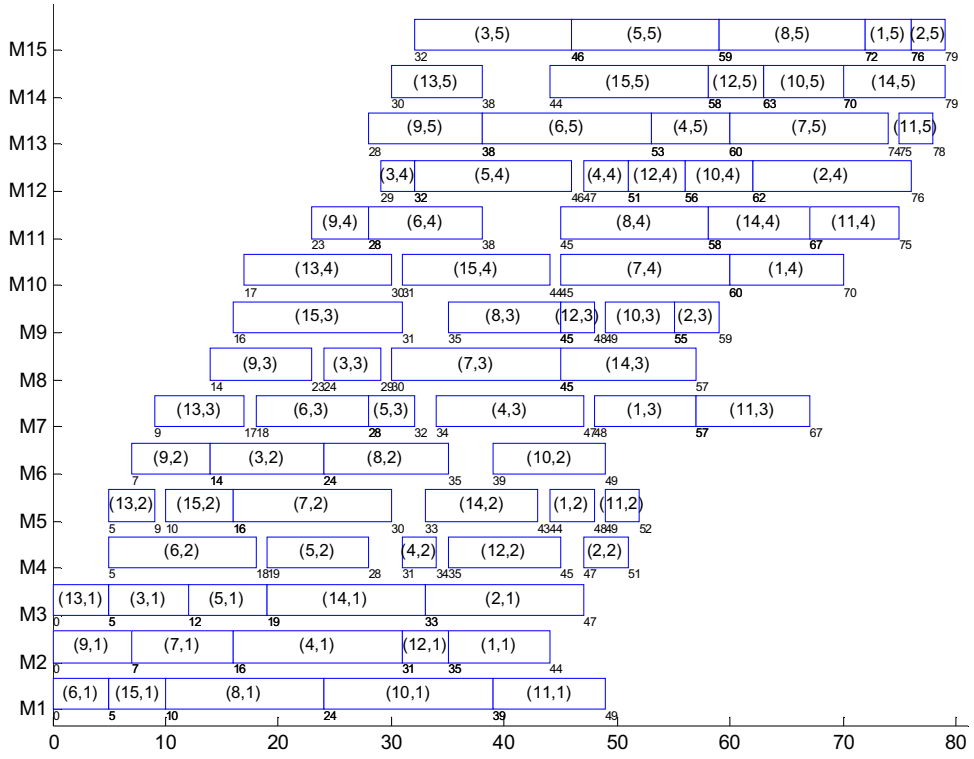


Fig. 7. Gantt chart for the obtained best solution for j15c5d5 (makespan = 79).

resulting p-value equals 0.0324, which is close to zero and suggests that there is a statistically significant difference between the two decoding heuristics at a 95% confidence level. Fig. 6 gives the means with the least significant difference or LSD intervals (at the 95% confidence level). It can be observed from Fig. 6 that our decoding heuristic results in a better performance.

Effectiveness of different versions of VNS

In HVNS, we proposed a KE-based neighborhood change approach and an EDA-based global search procedure. To verify the efficiency of the two approaches, we realize several versions of VNS, namely, the canonical VNS (VNS-I), VNS with KE-based neighborhood change approach (VNS-II), VNS with EDA-based global search procedure (VNS-III), and VNS with the two approaches (HVNS). To make detailed comparisons between the four algorithms, we select the 24 harder problems from the Carlier and Neron's benchmark problems, i.e., "j10c5c1" to "j10c5d6", and "j15c5c1" to "j15c5d6". The computational results are reported in Table 4.

It can be concluded from Table 4 that, for solving the harder problems, HVNS shows the best performance considering the makespan objective, the following algorithms are VNS-II, VNS-III, and VNS-I. That is, considering the search ability, HVNS is better than the other three algorithms; VNS-II shows better performance than VNS-III and VNS-I; and the worst among the four compared algorithms is the canonical VNS (VNS-I). The comparison results verify the efficiency of the two approaches embedded in the proposed algorithm.

Comparisons of HVNS and other presented algorithms

Comparisons of the 10-job problems

In Carlier and Neron's benchmark problems, there are 41 instances with 10-job scale. These problems can be classified into two categories, i.e., five stages and ten stages. The 10 job-5 stage instances range from "j10c5a2" to "j10c5d6", while the 10 job-10 stage instances range from "j10c10a1" to "j10c10c6".

The computational results for solving the 10-job problems are summarized in Table 5. There are 17 columns in the comparison table. The first column gives the tested problem name. The column labeled "LB" displays the lowest bound for each instance. Then, the following two columns tell the computational results due to the proposed HVNS algorithm, where, the third column gives the makespan and the fourth column reports the computational CPU times for each problem. The next two column list the computational results obtained by PSO algorithm. The makespan and the computational time collected by AIS algorithm are displayed in 7th and 8th column, respectively. The next two columns report the test results for ACO, while the following two columns give the results due to the B&B algorithm. Following the computational results for each compared algorithms, the deviation values for HVNS, PSO, AIS, ACO, and B&B, are reported in the last five columns.

It can be concluded from Table 5 that: (1) for solving the 10 job-5 stage problem with the first two kinds of machine layout, i.e., "j10c5a2" to "j10c5b6", all of the five compared algorithms obtain the LB value for each benchmark. The proposed HVNS can reach the LB in very short computational times for each instance, which shows the convergence ability of the proposed algorithm; (2) for solving the following hard twelve problems, i.e., "j10c5c1" to "j10c5d6", the proposed algorithm obtains eleven LB values, except for the benchmark "j10c5c3". However, the C_{max} (equals to 72) obtained by HVNS is slightly bigger than the given LB value (equals to 71), with a very short computational time; (3) for solving the following 12 10 job-10 stage benchmarks, HVNS reaches the LB for each problem with a very fast computational speed; (4) for solving the last six problems, i.e., "j10c10c1" to "j10c10c6", the proposed algorithm obtains near-optimal results for each instance. The computational times collected by HVNS for solving the six problems are also very short, which is competitive to other compared algorithms; (5) considering the deviation results, PSO gains

Table 7
Comparison results for the ten harder problems.

Instance	BEST	C_{max}			RPI			Time (second)		
		HVNS	PSO	AIS	HVNS	PSO	AIS	HVNS	PSO	AIS
j30c5e1	464.00	464.00	471.00	479.00	0.00	1.51	3.23	29.16	96.16	99.44
j30c5e2	616.00	616.00	616.00	619.00	0.00	0.00	0.49	11.69	55.28	80.24
j30c5e3	595.00	595.00	602.00	614.00	0.00	1.18	3.19	27.33	64.56	116.7
j30c5e4	566.00	566.00	575.00	582.00	0.00	1.59	2.83	37.00	86.98	108.63
j30c5e5	601.00	601.00	605.00	610.00	0.00	0.67	1.50	21.32	79.84	101.19
j30c5e6	603.00	603.00	605.00	620.00	0.00	0.33	2.82	33.39	0.996	100.47
j30c5e7	626.00	626.00	629.00	635.00	0.00	0.48	1.44	27.37	87.18	93.56
j30c5e8	674.00	674.00	678.00	686.00	0.00	0.59	1.78	31.40	97.67	100.68
j30c5e9	643.00	643.00	651.00	662.00	0.00	1.24	2.95	33.81	83.8	100.75
j30c5e10	577.00	577.00	594.00	604.00	0.00	2.95	4.68	29.72	77.46	89.29
Mean	596.50	596.50	602.60	611.10	0.00	1.05	2.49	28.22	73.00	99.1

the best among the five compared algorithms, the results obtained by HVNS is slightly worse than PSO. It should be noted that, in solving the given 41 10-job problems, HVNS obtained the same C_{max} values with PSO, except for the instance "j10c5c3". However, considering the average computational times consumed by the two algorithms, i.e., PSO and HVNS, the proposed algorithm shows better performance than PSO; and (6) for solving the given 10-job scale problems, the proposed HVNS algorithm obtains 34 optimal values of the 41 problems (83%), except seven instances with smaller average percentage deviation values. It can be concluded that the proposed algorithm is competitive to PSO, and shows better performance than the other three compared algorithm, i.e., AIS, ACO, and B&B in solving the given 10-job scale problems.

Comparisons of the 15-job problems

In Carlier and Neron's benchmark problems, there are 36 instances with 15-job scale. These problems can be classified into two categories, i.e., five stages and 10 stages. The 15 job-5 stage instances range from "j15c5a1" to "j15c5d6", while the 15 job-10 stage instances range from "j15c10a1" to "j15c10b6".

The computational results for solving the 15-job problems are summarized in Table 6. There are also 17 columns in the comparison table, with the same meanings as in Table 5. It can be concluded from Table 6 that: (1) for solving the 15 job-5 stage with the first two kinds of machine layout, i.e., "j15c5a1" to "j15c5b6", all of the compared algorithms obtain the LB value for each benchmark. The proposed HVNS can reach the LB in very short computational times, which shows the convergence ability of the proposed algorithm; (2) for solving the following hard 12 problems, i.e., "j15c5c1" to "j15c5d6", the proposed algorithm obtains six LB values within very short process duration. For the remained six harder problems, our proposed algorithm can obtain near optimal values with a very fast speed; (3) for solving the following 12 15 job-10 stage benchmarks, HVNS reaches the LB for each problem with a very fast computational speed, which is competitive to other compared algorithms; (4) considering the deviation results, PSO and HVNS gain the same results for solving the given 15-job problems. It should be noted that, considering the computational time, the proposed algorithm shows obvious better performance than PSO, AIS, ACO, and B&B; and (5) for solving the 36 larger scale problems, the proposed HVNS algorithm obtains 30 LB values of the 36 problems (83%), except six instances with smaller average percentage deviation values. It can be concluded that the proposed algorithm shows the best among the five compared algorithms for solving the given 15-job problems.

Fig. 7 gives the best solution obtained by HVNS for solving the benchmark "j15c5d5", and the resulted makespan equals to 79, which is better than the corresponding results obtained by AIS, ACO and B&B.

Comparisons of the 10 harder problems

In order to test the proposed algorithm on large-scale HFS problems, we select the 10 harder problems, i.e., "j30c5e1" to "j30c5e10". The compared algorithms include PSO, AIS, and the proposed HVNS algorithm. All of these three compared algorithms run

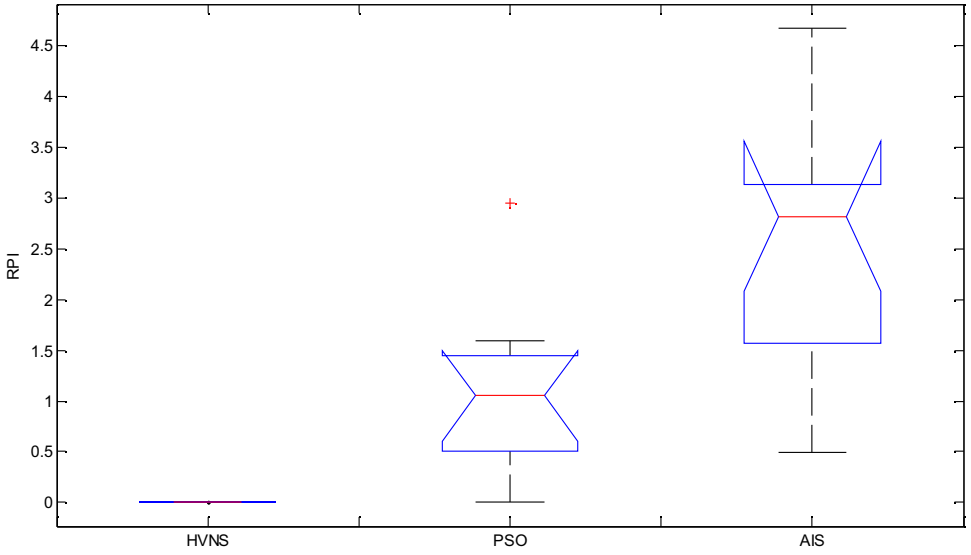


Fig. 8. Means and 95% LSD interval for HVS, PSO, and AIS.

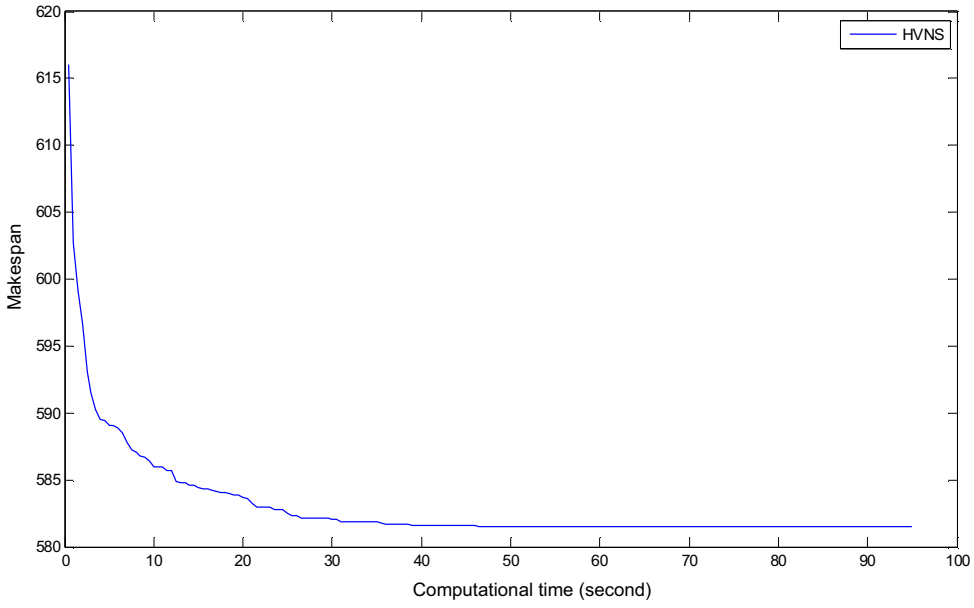


Fig. 9. Decreasing of the makespan obtained by HVNS for j130c5e10 instance.

20 times for each benchmark problem, and the best fitness values (makespan) are collected in Table 7. The average computational times consumed by each compared algorithm are also listed in the table. In Table 7, the first column gives the benchmark problem. Then, the following column presents the best makespan collected by the three compared algorithm. The next following columns list the minimum makespan values obtained by HVNS, PSO, and AIS, respectively. The RPI values of the minimum makespan for the compared algorithms are displayed in the following three columns, respectively. The last three columns presents the computational time (time unit: second) consumed by the three compared algorithms, respectively.

It can be concluded from the compared results that: (1) for solving the ten harder problems, HVNS shows the best performance; (2) the computational times consumed by HVNS for each benchmark are obvious shorter than the other compared algorithms, which verify the efficiency of the proposed algorithm; and (3) the proposed algorithm obtains the best values for all of the instances, which are obvious better than the other compared algorithms. For example, for solving "j30c5e10", the minimum fitness value obtained by HVNS equals to 577, while the result collected by PSO is 594, and 604 for AIS.

To check whether the observed difference from Table 7 is indeed statistically different, we also carry out a single-factor analysis of

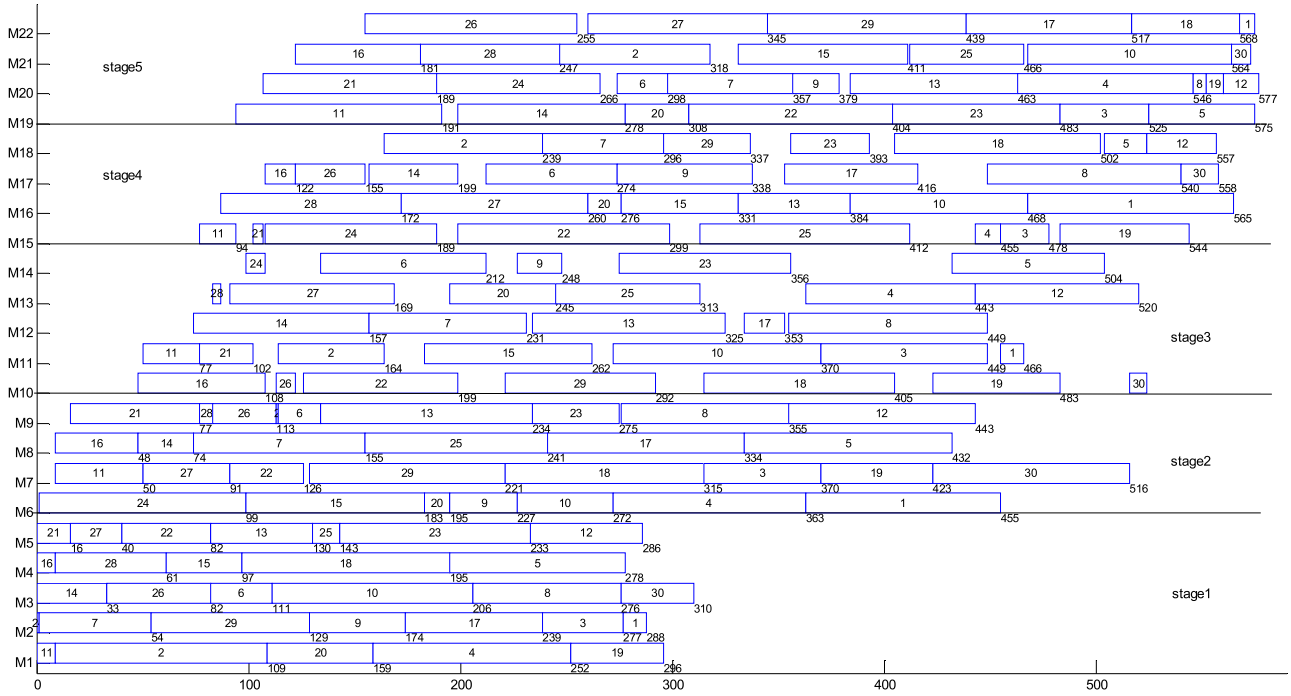


Fig. 10. Gantt chart of the obtained best solution for j30c5e10 (makespan = 577).

variance (ANOVA). Fig. 8 gives the means with the least significant difference or LSD intervals (at the 95% confidence level), which suggests that there is a statistically significant difference between the proposed HVNS algorithm and the other compared algorithms. It can be observed from Table 7 and Fig. 8 that HVNS results in a better performance for solving the given ten large-scale instances.

Fig. 9 gives the convergence curve of the average makespan for the "j30c5e10" instance over 20 runs. It can be seen that the proposed algorithm improves the average makespan very fast. Fig. 10 gives the Gantt chart of the best solution for "j30c5e10" obtained by HVNS.

Conclusions

In this study, a hybrid VNS algorithm combining the CRO and EDA is proposed for solving the hybrid flow shop scheduling problem. A detailed encoding and decoding mechanism is developed for the problem. The main contributions of the proposed HVNS are as follows:

(1) To enhance the exploitation ability of the proposed algorithm, a dynamic neighborhood set update mechanism is embedded in the HVNS;
(2) To avoid being stuck at the local optima and jump out of the valley, a kinetic-energy-based neighborhood change approach is proposed;
(3) To utilize the global statistical information from local optima solutions found by the VNS procedure and direct the search to promising region, an EDA-based global search method is investigated in the HVNS.

The Carlier and Neron's benchmark problems with different problem scales are tested to make detailed comparisons between HVNS and other efficient algorithms from the literature. Experimental results show the robustness and efficiency of the proposed algorithm. Objective for further research are as follows:

(1) To improve the local search ability of the algorithm and enhance the balance between the exploitation and exploration to increase the simulation result reliability;
(2) To consider the problem structure, design efficient neighborhood structures and increase the global search ability of the algorithm;
(3) To apply the proposed algorithm to solving other potential applications, such as the HFS with preventive maintenance activities, HFS with setup time, and fuzzy HFS problems.

Acknowledgements

This research is partially supported by National Science Foundation of China 61174187 and 61104179, Program for New Century Excellent Talents in University (NCET-13-0106), Specialized Research Fund for the Doctoral Program of Higher Education (20130042110035), Science Foundation of Liaoning Province in China (2013020016), Basic scientific research foundation of Northeast University under Grant N110208001 and N130508001, Starting foundation of Northeast University under Grant 29321006, and IAPI Fundamental Research Funds (2013ZCX02). The authors would like to thank Jasques Carlier and Emmanuel Néron for their benchmark problems. We are also grateful to Ching-Jong Liao, Evi Tjandradjaja, and Tsui-Ping Chung for their benchmark problems.

References

[1] J.N.D. Gupta, Two-stage, hybrid flow shop scheduling problem, J. Oper. Res. Soc. 39 (1988) 359–364.
[2] R. Ruiz, J.A. Vázquez Rodríguez, The hybrid flow shop scheduling problem, Eur. J. Oper. Res. 205 (2010) 1–18.
[3] I. Ribas, R. Leisten, J.M. Framinan, Review and classification of hybrid flow shop scheduling problems from a production systems and a solutions procedure perspective, Comput. Oper. Res. 37 (2010) 1439–1454.
[4] T.B.K. Rao, Sequencing in the order a, b, with multiplicity of machines for a single operation, OPSEARCH: J. Oper. Res. Soc. India 7 (1970) 135–144.
[5] S.A. Brah, J.L. Hunsucker, Branch and bound algorithm for the flow-shop with multiple processors, Eur. J. Oper. Res. 51 (1991) 88–99.
[6] J.N.D. Gupta, E.A. Tunc, Scheduling a two-stage hybrid flowshop with separable setup and removal times, Eur. J. Oper. Res. 77 (1994) 415–428.
[7] H.T. Lin, C.J. Liao, A case study in a two-stage hybrid flow shop with setup time and dedicated machines, Int. J. Prod. Econ. 86 (2003) 133–143.
[8] G.C. Lee, Y.D. Kim, A branch-and-bound algorithm for a two-stage hybrid flow-shop scheduling problem minimizing total tardiness, Int. J. Prod. Res. 42 (2004) 4731–4743.
[9] M. Haouari, L. Hidri, A. Gharbi, Optimal scheduling of a two-stage hybrid flow shop, Math. Methods Oper. Res. 64 (2006) 107–124.
[10] F. Riane, A. Artiba, S.E. Elmaghraby, A hybrid three-stage flowshop problem: efficient heuristics to minimize makespan, Eur. J. Oper. Res. 109 (1998) 321–329.
[11] Z.H. Jin, K. Ohno, T. Ito, S.E. Elmaghraby, Scheduling hybrid flowshops in printed circuit board assembly lines, Prod. Oper. Manag. 11 (2002) 216–230.
[12] J. Carlier, E. Neron, An exact method for solving the multi-processor flowshop, RAIRO Oper. Res. 34 (2000) 1–25.
[13] A. Babayan, D. He, Solving the n-job three-stage flexible flowshop scheduling problem using an agent-based approach, Int. J. Prod. Res. 42 (2004) 777–799.
[14] M.C. Portmann, A. Vignier, D. Dardilhac, D. Dezalay, Branch and bound crossed with GA to solve hybrid flowshops, Eur. J. Oper. Res. 107 (1998) 389–400.
[15] E. Neron, P. Baptiste, J.N.D. Gupta, Solving hybrid flow shop problem using energetic reasoning and global operations, Omega-Int. J. Manag. Sci. 29 (2001) 501–511.
[16] C. Oguz, M. Ercan, A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks, J. Scheduling 8 (2005) 323–351.
[17] R. Ruiz, C. Maroto, A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility, Eur. J. Oper. Res. 169 (2006) 781–800.
[18] A. Janiak, E. Kozan, M. Lichtenstein, C. Oguz, Metaheuristic approaches to the hybrid flow shop scheduling problem with a cost-related criterion, Int. J. Prod. Econ. 105 (2007) 407–424.
[19] Q. Niu, T. Zhou, S. Ma, A quantum-inspired immune algorithm for hybrid flow shop with makespan criterion, J. Univers. Comput. Sci. 15 (2009) 765–785.
[20] C. Kahraman, O. Enginb, I. Kaya, R.E. Öztürk, Multiprocessor task scheduling in multistage hybrid flow-shops: a parallel greedy algorithm approach, Appl. Soft Comput. 10 (2010) 1293–1300.
[21] O. Engin, G. Ceran, M.K. Yilmaz, An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems, Appl. Soft Comput. 11 (2011) 3056–3065.
[22] C.J. Liao, E. Tjandradjaja, T.P. Chung, An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem, Appl. Soft Comput. 12 (2012) 1755–1764.
[23] K.C. Ying, S.W. Lin, Multiprocessor task scheduling in multistage hybrid flow-shops: an ant colony system approach, Int. J. Prod. Res. 44 (2006) 3161–3177.
[24] K. Alaykyran, O. Engin, A. Doyen, Using ant colony optimization to solve hybrid flow shop scheduling problems, Int. J. Adv. Manuf. Technol. 35 (2007) 541–550.
[25] O. Engin, A. Doyen, A new approach to solve hybrid flow shop scheduling problems by artificial immune system, Future Gener. Comput. Syst. 20 (2004) 1083–1095.
[26] M. Zandieh, S.M.T. Fatemi Ghomi, S.M. Moattar Husseini, An immune algorithm approach to hybrid flow shops scheduling with sequence dependent setup times, Appl. Math. Comput. 180 (2006) 111–127.
[27] N. Mladenovic, P. Hansen, Variable neighborhood search, Comput. Oper. Res. 24 (1997) 1097–1100.
[28] P. Hansen, N. Mladenović, J.A. Moreno Pérez, Variable neighbourhood search: methods and applications, Ann. Oper. Res. 175 (2010) 367–407.
[29] X.P. Wang, L.X. Tang, A population-based variable neighborhood search for the single machine total weighted tardiness problem, Comput. Oper. Res. 36 (2009) 2105–2110.
[30] Y. Wen, H. Xu, J.D. Yang, A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system, Inf. Sci. 181 (2011) 567–581.
[31] A. Stenger, D. Vigo, S. Enz, M. Schwind, An adaptive variable neighborhood search algorithm for a vehicle routing problem arising in small package shipping, Transport. Sci. 47 (2013) 64–80.
[32] M. Yazdani, M. Zandieh, M. Amiri, Flexible job-shop scheduling with parallel variable neighborhood search algorithm, Expert Syst. Appl. 37 (2010) 678–687.
[33] J. Behnamian, M. Zandieh, S.M.T. Fatemi Ghomi, Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm, Expert Syst. Appl. 36 (2009) 9637–9644.
[34] R. Driessel, L. Mönch, Variable neighborhood search approaches for scheduling jobs on parallel machines with sequence-dependent setup times, precedence constraints, and ready times, Comput. Ind. Eng. 61 (2011) 336–345.
[35] A. Bilyk, L. Mönch, A variable neighborhood search approach for planning and scheduling of jobs on unrelated parallel machines, J. Intell. Manuf. 23 (2012) 1621–1635.

[36] B. Naderi, M. Zandieh, S.M.T. Fatemi Ghomi, A study on integrating sequence dependent setup time flexible flow lines and preventive maintenance scheduling, J. Intell. Manuf. 20 (2009) 683–694.
[37] R. Tavakkoli-Moghaddam, N. Safaei, F. Sassani, A memetic algorithm for the flexible flow line scheduling problem with processor blocking, Comput. Oper. Res. 36 (2009) 402–414.
[38] A. Duartea, L.F. Escuderob, R. Marti, N. Mladenović, J.J. Pantrigo, J. Sanchez-Oro, Variable neighborhood search for the Vertex Separation Problem, Comput. Oper. Res. 39 (2012) 3247–3255.
[39] Y.C. Liang, C.Y. Chuang, Variable neighborhood search for multi-objective resource allocation problems, Robot. Comput. Integr. Manuf. 29 (2013) 73–78.
[40] N. Mladenović, D. Urošević, S. Hanafi, A. Ilić, A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem, Eur. J. Oper. Res. 220 (2012) 270–285.
[41] B. Jarboui, H. Derbel, S. Hanafi, N. Mladenović, Variable neighborhood search for location routing, Comput. Oper. Res. 40 (2013) 47–57.
[42] Y.Y. Xiao, I. Kaku, Q.H. Zhao, R.Q. Zhang, A reduced variable neighborhood search algorithm for uncapacitated multilevel lot-sizing problems, Eur. J. Oper. Res. 214 (2011) 223–231.
[43] M. Abedzadeh, M. Mazinani, N. Moradinasab, E. Roghanian, Parallel variable neighborhood search for solving fuzzy multi-objective dynamic facility layout problem, Int. J. Adv. Manuf. Technol. 65 (2013) 197–211.
[44] A.Y.S. Lam, V.O.K. Li, Chemical-reaction-inspired metaheuristic for optimization, IEEE Trans. Evol. Comput. 14 (2010) 381–399.
[45] A.Y.S. Lam, V.O.K. Li, J.J.Q. Yu, Real-Coded Chemical Reaction Optimization, IEEE Trans. Evol. Comput. 16 (2012) 339–353.
[46] J.Q. Li, Q.K. Pan, Chemical-reaction optimization for flexible job-shop scheduling problems with maintenance activity, Appl. Soft Comput. 12 (2012) 2896–2912.
[47] H. Mühlenbein, G. Paaß, From recombination of genes to the estimation of distribution. Binary parameters, in: Lecture notes in computer science 1411: parallel problem solving from nature, PPSN, IV, 1996, pp. 178–187.
[48] B. Jarboui, M. Eddaly, P. Siarry, An estimation of distribution algorithm for minimizing the total flowtime in permutation flow shop scheduling problems, Comput. Oper. Res. 36 (2009) 2638–2646.
[49] S.Y. Wang, L. Wang, Y. Xu, G. Zhou, An estimation of distribution algorithm for solving hybrid flow-shop scheduling problem, Acta Automatica Sin. 38 (2012) 437–443.
[50] Q.K. Pan, R. Ruiz, An estimation of distribution algorithm for lot-streaming flow shop problems with setup times, OMEGA-Int. J. Manag. Sci. 40 (2012) 166–180.
[51] D.C. Montgomery, Design and Analysis of Experiments, John Wiley & Sons, Arizona, 2005.