Applied Soft Computing 13 (2013) 3259–3272

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

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

A hybrid harmony search algorithm for the flexible job shop scheduling problem

Yuan Yuan, Hua Xu*, Jiadong Yang

State Key Laboratory of Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, PR China

ARTICLE INFO

Article history:
Received 1 May 2012
Received in revised form 5 January 2013
Accepted 4 February 2013
Available online 5 March 2013

Keywords:
Scheduling
Flexible job shop
Harmony search
Local search
Neighborhood structure
Makespan

ABSTRACT

In this paper, a novel hybrid harmony search (HHS) algorithm based on the integrated approach, is proposed for solving the flexible job shop scheduling problem (FJSP) with the criterion to minimize makespan. First of all, to make the harmony search (HS) algorithm adaptive to the FJSP, the converting techniques are developed to convert the continuous harmony vector to a kind of discrete two-vector code for the FJSP. Secondly, the harmony vector is mapped into a feasible active schedule through effectively decoding the transformed two-vector code, which could largely reduce the search space. Thirdly, a resultful initialization scheme combining heuristic and random strategies is introduced to make the initial harmony memory (HM) occur with certain quality and diversity. Furthermore, a local search procedure is embedded in the HS algorithm to enhance the local exploitation ability, whereas HS is employed to perform exploration by evolving harmony vectors in the HM. To speed up the local search process, the improved neighborhood structure based on common critical operations is presented in detail. Empirical results on various benchmark instances validate the effectiveness and efficiency of our proposed algorithm. Our work also indicates that a well designed HS-based method is a competitive alternative for addressing the FJSP.

1. Introduction

Scheduling, as a decision-making process, plays an important role in most manufacturing and production systems as well as in most information processing environments [1]. The classical job shop scheduling problem (JSP) is one of the most important and difficult problems in this field and has received an enormous amount of attention in the research literature [2–5]. In JSP, a set of jobs must be processed on a set of machines. Each job consists of a sequence of consecutive operations which have the order of precedence. Each operation needs to be performed on exactly one specified machine. Machines are continuously available at time-zero and can process one operation at a time without interruption. The decision concerns how to sequence the operations on each machine, so that a given performance indicator is optimized. The time required to complete all the jobs, namely makespan, is a typical performance indicator for JSP.

The flexible job shop scheduling problem (FJSP) is a generalization of classical JSP, in which operations are allowed to be processed by any machine from a given set, rather than one specified machine. Compared with classical JSP, the FJSP is closer to a real production environment and has more practical applicability. But, it is also more difficult than classical JSP because of its additional decision to assign each operation to the appropriate machine. JSP is well known to be NP-hard [6]. FJSP is therefore NP-hard too.

In recent years, various meta-heuristics have been extensively applied to solve the challenging FJSP because of its computational complexity. The harmony search (HS) algorithm developed by Geem et al. [7] is one of the latest population-based evolutionary meta-heuristics. Different from the traditional GA, the HS generates a new vector after considering all of the existing vectors, whereas the GA only consider the two parent vectors. Numerical comparisons [8–10] showed that the evolution in the HS algorithm is faster than GA. Due to its simplicity, few parameters, and easy implementation, the HS algorithm has captured much attention and has been successfully applied to a wide range of real-world problems [11–14]. However, because of its continuous nature, the research on the HS for scheduling problems is still considerably limited. Especially, as far as we are aware, there is no detailed work that describes the use of HS algorithm to deal with the FJSP. In this paper, following existing successful applications of the HS, a hybrid harmony search (HHS) algorithm based on the integrated approach is proposed to solve the FJSP with makespan criterion. Unlike basic HS,

* Corresponding author.
E-mail address: xuhua@tsinghua.edu.cn (H. Xu).

1568-4946/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2013.02.013

the proposed HHS incorporates HS and a local search to achieve a balance between the global exploration and local exploitation of the search space, which is crucial to the success of meta-heuristics [15–17]. In the proposed HHS, we represent the harmonies as real vectors and use the converting techniques to map these continuous vectors to a feasible active schedule for the FJSP. Moreover, a resultful initialization scheme combining heuristic and random strategies is presented to generate the initial harmony memory (HM) with certain quality and diversity. Then, the optimization process is executed by providing cooperation between the global search based HS and the local search. To speed up the local search procedure, we introduce the concept of common critical operations to improve the type of neighborhoods based on the critical path. Computational results and comparisons demonstrate the effectiveness and efficiency of the proposed algorithm for solving the FJSP with makespan criterion.

The remainder of this paper is organized as follows. In Section 2, the FJSP is described and an illustrative problem instance is given. Section 3 introduces some existing related research on the FJSP and HS. In Section 4, the proposed algorithm is illustrated in detail. The extensive computational study on our algorithm is provided in Section 5. Section 6 discusses some features of the proposed algorithm. Finally, conclusions and suggestions for future research are drawn in Section 7.

2. Problem description

The FJSP can be defined as follows. There are a set of q independent jobs $J = \{J_1, J_2, \ldots, J_q\}$ and a set of r machines $M = \{M_1, M_2, \ldots, M_r\}$. Each job J_i consists of a sequence of precedence constrained operations $O_{i,1}, O_{i,2}, \ldots, O_{i,n_i}$. The job J_i is completed only when all its operations are executed in a given order, which can be represented as $O_{i,1} \rightarrow O_{i,2} \rightarrow \ldots \rightarrow O_{i,n_i}$. Each operation $O_{i,j}$, i.e. the jth operation of job J_i, can be executed on any among a subset $M_{i,j}$ of compatible machines. The FJSP can be further classified into P-FJSP and T-FJSP according to the relation between the set $M_{i,j}$ and M. If there exists a proper subset $M_{i,j} \subset M$, then it has partial flexibility, it is partial FJSP (P-FJSP). If $M_{i,j} = M$ for any operation, then it has total flexibility, it is total FJSP (T-FJSP). The processing time of each operation is machine dependent. We define $p_{i,j,k}$ as the processing time of $O_{i,j}$ on machine M_k. The scheduling problem is to assign each operation to an appropriate machine (routing problem) and to determine a sequence of operations on all the machines (sequencing problem). The objective is to find a schedule which minimize the makespan. The makespan means the time needed to complete all the jobs and can be defined as $C_{\max} = \max_{1 \le i \le q}(C_i)$, where C_i is the completion time of job J_i. Obviously, C_i equals to the completion time of operation O_{i,n_i} which is the last operation of job J_i.

Moreover, the following assumptions are made in this study: all the machines are available at time 0; all the jobs are released at time 0; each machine can process only one operation at a time; each operation must be completed without interruption once it starts; the order of operations for each job is predefined and cannot be modified; the setting up time of machines and transfer time of operations are negligible.

For illustrating the algorithm more explicitly, a sample problem instance of P-FJSP is shown in Table 1, where rows correspond to operations and columns correspond to machines. Each entry of the input table denotes the processing time of that operation on the corresponding machine. In this table, the tag "–" means that a machine cannot execute the corresponding operation.

Finally, some symbols used mostly through this paper are summarized in Table 2.

Table 1
Processing time table of an instance of P-FJSP.

Job	Operation	M_1	M_2	M_3
J_1	$O_{1,1}$	2	–	3
	$O_{1,2}$	4	1	3
J_2	$O_{2,1}$	–	5	3
	$O_{2,2}$	6	2	4
	$O_{2,3}$	3	–	–
J_3	$O_{3,1}$	1	5	2
	$O_{3,2}$	3	–	1

Table 2
List of symbols.

Symbol	Description
J	Set of all jobs
M	Set of all machines
q	Total number of jobs
r	Total number of machines
J_i	The ith job
n_i	Number of operations of job J_i
M_k	The kth machine
$O_{i,j}$	The jth operation of job J_i
$M_{i,j}$	Set of alternative machines of operation $O_{i,j}$
$p_{i,j,k}$	Processing time of $O_{i,j}$ on machine M_k
$C_{\max}$	Time needed to complete all jobs
l	Total number of all operations
$f(X)$	Objective function value of harmony vector X
X_i	The ith harmony vector in the harmony memory
$x_i(j)$	The jth decision variable of harmony vector X_i
$x_{\min}(j)$	Lower bound for the decision variable $x_i(j)$
$x_{\max}(j)$	Upper bound for the decision variable $x_i(j)$
n	Dimension of harmony vector
HMS	Harmony memory size
$HMCR$	Harmony memory considering rate
PAR	Pitch adjusting rate
NI	Number of improvisations
$loop_{\max}$	Max iterations of local search
$X^{(1)}$	The first half part of harmony vector
$X^{(2)}$	The second half part of harmony vector
X_{best}	The best harmony vector in the harmony memory
X_{worst}	The worst harmony vector in the harmony memory
X_{new}	New harmony vector obtained through improvisation in harmony search
X'_{new}	Harmony vector obtained after local search
$s(j)$	Number of alternative machines of operation j
$s_{i,j}$	Start time of operation $O_{i,j}$ in the schedule
$c_{i,j}$	Completion time of operation $O_{i,j}$ in the schedule
ET_k	Current end time for machine M_k
G	Schedule represented by the disjunctive graph
G'	Disjunctive graph yielded by deleting a common critical operation in G
$ES'_{i,j}$	The earliest starting time for the operation $O_{i,j}$ in G'
$LS'_{i,j}$	The latest starting time for the operation $O_{i,j}$ in G'
$LC'_{i,j}$	The latest completion time for the operation $O_{i,j}$ in G'
δ	Bound factor
MA	Machine assignment vector
$u(j)$	The jth decision variable of machine assignment vector
OS	Operation sequence vector
$v(j)$	The jth decision variable of operation sequence vector
π	Operation ID permutation sequenced in the scheduling order

3. Related work

3.1. Flexible job shop scheduling

The initial study of FJSP can trace back to 1990, Bruker and Schlie [18] proposed a polynomial algorithm for solving the FJSP with two jobs. But in the general form, FJSP is strongly NP-hard. Due to the NP-hard property, exact algorithms are not effective for solving FJSP especially for problems on a large scale. So, over the past two decades, the metaheuristic methods have become very popular with the FJSP, such as simulated annealing (SA) [19], tabu search

(TS) [20–22], genetic algorithm (GA) [23–27], particle swarm optimization (PSO) [28], biogeography-based optimization (BBO) [29] and hybrid techniques [30–34]. These methods are applied in order to find the satisfying schedule in acceptable computation time and can be classified into two main categories: hierarchical approach and integrated approach.

The hierarchical approach reduces the difficulty of FJSP by decomposing it into two sub-problems: routing sub-problem and sequencing sub-problem, then treats them separately. The idea is natural for solving FJSP, because when the machine assignment for each operation is fixed, the remained sequencing sub-problem turns into the JSP. Brandimarte [35] was the first to adopt the hierarchical approach for the FJSP. He solved the routing sub-problem using some dispatching rules and then solved the resulting JSP using a TS heuristic. Tung et al. [36] presented a similar algorithm for the scheduling in a flexible manufacturing system. Kacem et al. [37] proposed a GA algorithm controlled by the assigned model which is generated by the approach of localization. Xia and Wu [30] presented an effective hybrid method for the multi-objective FJSP. They made use of particle swarm optimization (PSO) to assign operations on machines and SA algorithm to sequence operations on each machine. Fattahi et al. [31] developed four kinds of hierarchical approaches based on SA and TS heuristics for solving the FJSP.

The integrated approach is used by considering assignment and sequencing at the same time. The past study indicated that it can achieve a better performance than the hierarchical approach in general, but it is usually more difficult to design. Hurink et al. [20] presented a TS heuristic in which reassignment and rescheduling are regarded as two different types of moves. Dauzère-Pérès and Paulli [21] developed a TS procedure based on a new neighborhood structure that makes no distinction between reassigning and resequencing. Mastrolilli and Gambardella [22] further improved their TS techniques and proposed two neighborhood functions for the FJSP. Chen et al. [23] pronounced an effective GA to solve the FJSP. In their method, the chromosome representation is divided into two parts, the first one denotes a concrete allocation of operations to each machine and the second one describes the sequence of operations on each machine. Jia et al. [24] proposed a modified GA which is able to solve distribute scheduling problems and FJSP. Zhang and Gen [25] pronounced a multistage operation-based GA to deal with the problem from the point view of dynamic programming. Pezzella et al. [26] presented a GA in which a mix of different strategies for generating the initial population, selecting the individuals for reproduction, and reproducing new individuals are integrated. Gao et al. [32] combined GA with variable neighborhood descent (VND) procedure for solving the multi-objective FJSP. Recently, more metaheuristic methods founded on the integrated approach were developed to solve the FJSP. Bagheri et al. [38] presented an artificial immune algorithm (AIA) using some resultful rules. Yazdani et al. [39] proposed a parallel variable neighborhood search (PVNS) algorithm based on the application of multiple independent searches. Xing et al. [40] developed a knowledge-based ant colony optimization (KBACO) algorithm that provides an effective integration between the ant colony optimization (ACO) model and the knowledge model. Li et al. [33] proposed a hybrid TS with an efficient neighborhood structure for the FJSP. Wang et al. [41,42] applied the artificial bee colony (ABC) algorithm and estimation of distribution algorithm (EDA) to the FJSP, both of which stress the balance between global exploration and local exploitation.

3.2. Harmony search algorithm

The harmony search (HS) [7] algorithm was originally devised for optimizing various continuous nonlinear functions. Its basic idea was inspired by the musical improvisation process where musicians improvise their instruments' pitches searching for a perfect state of harmony. In the HS, there exists several core analogies between music improvisation and engineering optimization: each instrument vs. each decision variable; the pitch of an instrument vs. the value of a decision variable; a harmony produced by all the instruments vs. a solution for the optimization problem; the aesthetic quality of a harmony vs. the objective function value of a solution. Based on these analogies, a harmony is represented by an n-dimension real vector and the work mechanism of HS can be simply described as the following. First, an initial population of harmony vectors are randomly generated and sorted in a harmony memory (HM). Then a new candidate harmony is generated from the HM by using a memory consideration, a pitch adjustment, and a random selection. Finally, the HM is updated by comparing the new candidate harmony with the existing worst harmony vector in the current HM. The above process is repeated until the termination criterion is satisfied. More detailed information about HS can be referred to in Refs. [7,8].

3.3. The differences between HHS and TSPCB [33]

Recently, Li et al. [33] presented a hybrid TS called TSPCB for the FJSP, where the local search is also based on critical operations. In this subsection, we would like to have a comparison with this method to identify the novelty of the proposed HHS. This comparison is conducted from the aspects of two-vector code representation, initialization, and critical operation based local search.

First of all, for the two-vector code representation, the HHS and TSPCB both adopt the classical operation-based representation proposed in [43] for the operation sequence component. However, for the machine assignment vector MA = $\{u(1), u(2), \ldots, u(l)\}$, $u(j)$ represents the operation j chooses the $u(j)$th operation in its alternative machine set in the HHS, while in the TSPCB represents the operation j selects machine $M_{u(j)}$. Our representation is obviously more convenient and flexible, especially for problems which have partial flexibility (P-FJSP). Because the possible values of $u(j)$ only depend on the number of alternative machines of operation j, which is denoted as $s(j)$. And $u(j) \in \{1, 2, \ldots, s(j)\}$, which is the set of consecutive integers. The consecutiveness also helps to the conversion to real values within continuous ranges in Eq. (1). Secondly, in the initialization phase, to guarantee the initial HM with certain quality and diversity, we generate one harmony vector by a heuristic approach and the remainder are all produced randomly. The heuristic approach adopts approach by localization (AL) proposed in [37] to assign each operation to a suitable machine, while the known dispatching rule, most work remaining (MWR), is used to determine how to sequence the operations on the machines. The TSPCB concentrates more on the initialization. Many rules used in [26] are incorporated, and the vectors are produced through the adoption of a mix of these rules. Last but not the least, although the local search in the HHS and in the TSPCB are both based on the critical operations, their working mechanisms are rather different. One of the biggest difference lies in that the TSPCB regards machine assignment and operation sequence as two independent and successive operators. When producing a neighboring solution in the TSPCB, it first changes the machine assignment according to some rules while the operation sequence is fixed. Then in the operation sequence part, it adopts several neighborhoods that are similar to some neighborhoods originally designed for the JSP, which do not alter the machine assignment. However, the machine assignment and the operation sequence are closely related in the FJSP, they together decide a solution. One machine assignment may good for one operation sequence, but quite bad for another. So to adjust one component by fixing another component is not very reasonable. In addition, in the operation sequence, the neighborhoods which are borrowed by the JSP is not very problem specific. Because these

operators cannot guarantee the obtained neighborhood is no worse than the current solution. The local search method proposed in the HHS seems more reasonable and problem specific. The machine assignment and operation sequence is not separated in the operator of generating a neighborhood. We first identify all the common critical operations, because the solution could be only be improved by moving these operations. Then we try to move these operations one by one until one operation is moved successfully. When moving one operation, we delete the operation from the current schedule, which can be seen as a kind of relaxation. Take the makespan of the current schedule as the required makespan, the available time intervals on all the machines could be got after deletion. Then, we scan the machines one by one until we can find an interval on certain machine for the operation to insert. Once inserted, the neighborhood is immediately obtained. From the above, we can see that the move could change the machine assignment and operation sequence at the same time. And the success insertion can assure the neighborhood is no worse than the current solution.

To summarize, the strategies used in the HHS are novel in comparison with the TSPCB, especially in the local search phase.

4. The proposed algorithm

This section will present a detailed description of the proposed HHS algorithm, including the whole algorithmic flow and crucial procedures. To facilitate the elaboration, some basic notations are first given. Let $X_i = \{x_i(1), x_i(2), \ldots, x_i(n)\}$ represent the ith harmony vector in the HM and $x_i(j) \in [x_{\min}(j), x_{\max}(j)]$, where $x_{\min}(j)$ and $x_{\max}(j)$ are the lower bound and upper bound for the position value of each dimension j, respectively. The HM consists of harmony vectors in the memory which can be represented as $HM = \{X_1, X_2, \ldots, X_{HMS}\}$, where HMS denotes the harmony memory size. The best and worst harmony vectors in the HM are separately labeled as X_{best} and X_{worst}. It is noteworthy that the objective function value $f(X)$ for a given harmony vector X denotes makespan in our algorithm, so a smaller $f(X)$ means a better harmony.

4.1. Framework

The framework of the proposed HHS algorithm is based on HS and its algorithmic flow is depicted in Fig. 1. In the optimization process, parameters and the stopping criterion are first set. Then, the HM is initialized by combining heuristics and a randomized procedure. After the initialization, the harmony vectors in the HM are evaluated through being mapped to a feasible active schedule for the FJSP, in the meantime, the best and worst harmony vector are labeled. Following the evaluation, a new harmony memory is improvised from the HM by means of a memory consideration, a pitch adjustment, and a random selection. Thereafter, a local search algorithm is performed to improve the harmony vector generated in the improvisation phase, which is different from the basic HS. Finally, the HM is updated by comparing the harmony vector improved by the local search with the worst harmony vector in the HM. The evolution process is repeated until the stopping criterion is satisfied.

4.2. Representation of the harmony vector

In our proposed algorithm, a harmony vector, $X = \{x(1), x(2), \ldots, x(n)\}$ is still represented as an n-dimensional real vector. But the dimension n must satisfy certain constraint according to the problem. Let l be the number of all the operations in the FJSP to solve, then $n = 2l$. The first half part of the harmony vector $X^{(1)} = \{x(1), x(2), \ldots, x(l)\}$ describes the information of machine assignment for each operation, while the last half part of the harmony vector

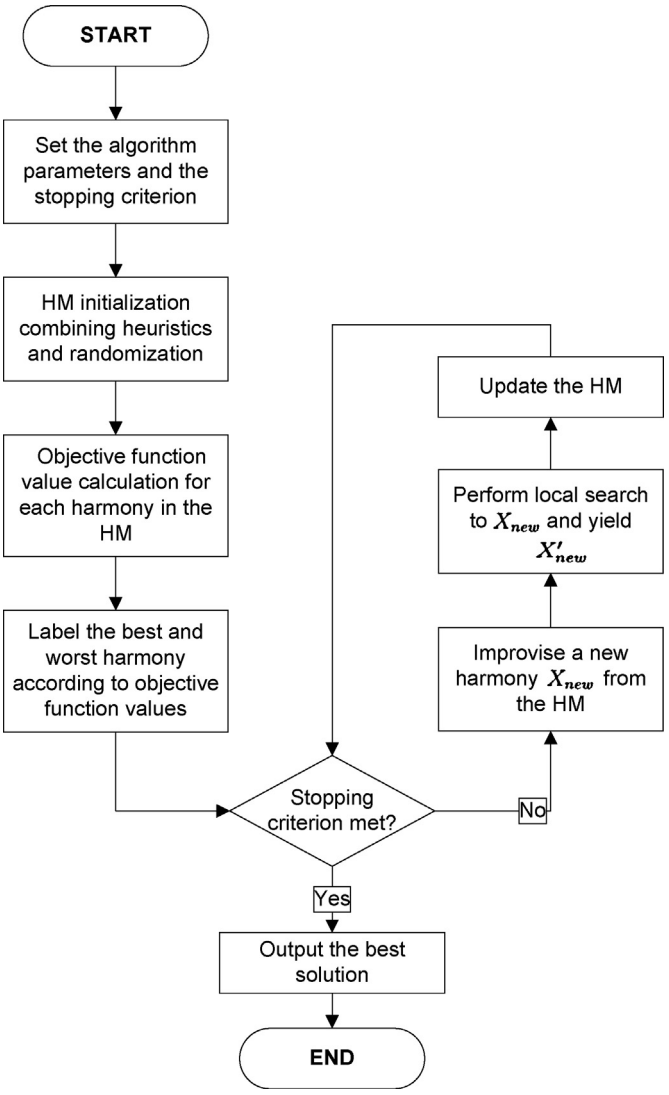

Fig. 1. Algorithmic flow of the proposed HHS algorithm.

$X^{(2)} = \{x(l+1), x(l+2), \ldots, x(2l)\}$ presents the information of operations sequencing on all the machines. This design can correspond well to the two-vector code for the FJSP that will be illustrated in Section 4.3.1. What's more, to deal with the problem conveniently, the intervals $[x_{\min}(j), x_{\max}(j)], j = 1, 2, \ldots, n$, are all set as $[-\delta, \delta], \delta > 0$ in the proposed method.

4.3. Evaluation of the harmony vector

In this subsection, we will address the key problem throughout the proposed HHS, that is, when given a harmony vector $X = \{x(1), x(2), \ldots, x(n)\}$, how to evaluate the objective function value $f(X)$. To realize this, the harmony vector X, represented as a real number vector, is first converted to a kind of two-vector code. Then the two-vector code is decoded to a feasible and active schedule for the FJSP. $f(X)$ is given the value of makespan corresponding to the schedule.

4.3.1. Two-vector code

The two-vector code in this paper consists of two vectors: machine assignment vector and operation sequence vector, corresponding to two subproblems in the FJSP.

Before explaining the two vectors in detail, a fixed ID for each operation is first given in accordance with the job number and operation order within the job. This numbering scheme is illustrated in

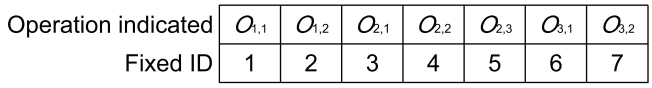

Fig. 2. Illustration of numbering scheme for operations.

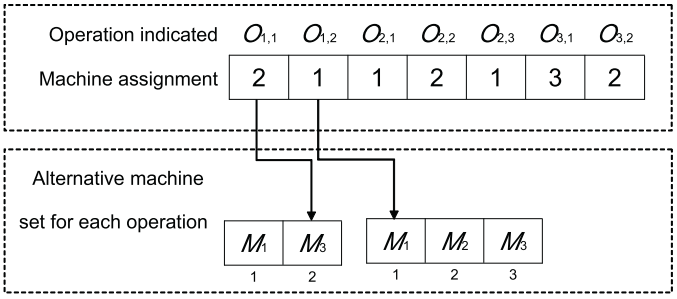

Fig. 3. Illustration of the machine assignment vector.

Fig. 2 for the instance shown in Table 1. After numbered, the operation can also be referred to by the fixed ID, for example, operation 6 has the same reference with the operation $O_{3,1}$ as shown in Fig. 2.

The machine assignment vector, MA = {$u(1), u(2), \ldots, u(l)$}, is an array of l integer values, where l is the total number of operations in the FJSP. In the vector, $u(j)$, $1 \le j \le l$, represents the operation j chooses the $u(j)$th machine in its alternative machine set. For the problem in Table 1, a possible machine assignment vector is shown in Fig. 3 and its meaning is also revealed. For example, $u(1) = 2$ indicates that the operation $O_{1,1}$ chooses the 2nd machine in its alternative machine set, that is machine M_3.

As for the operation sequence vector, OS = {$v(1), v(2), \ldots, v(l)$}, the ID permutation of all the operations is the most natural representation. But because of the existence of the precedence constraints between operations, this representation not always defines a feasible schedule for the FJSP. So, we adopt the operation-based representation [43] which names all the operations of a job with the same symbol and then interprets them according to the order of occurrence in the sequence of a given code. Unlike the permutation representation, this code can be always decoded to a feasible schedule. By means of this encoding method, the index i of the job J_i appears exactly n_i times in the operation sequence vector and represents n_i operations $O_{i,1}$, $O_{i,2}$, …, O_{i,n_i} in turn. For the instance in Table 1, a possible operation sequence vector is shown in Fig. 4 and can be translated into a unique list of ordered operations: $O_{2,1} \succ O_{1,1} \succ O_{2,2} \succ O_{1,2} \succ O_{3,1} \succ O_{2,3} \succ O_{3,2}$. Operation $O_{2,1}$ has the highest priority and is scheduled first, then the operation $O_{1,1}$, and so on.

4.3.2. Converting the harmony vector to the two-vector code

The conversion of a given harmony vector $X = \{x(1), x(2), \ldots, x(l), x(l+1), x(l+2), \ldots, x(2l)\}$, where $-\delta \le x(j) \le \delta$, $j = 1, 2, \ldots, 2l$, is divided into two separate parts. In the first part, we convert the $X^{(1)} = \{x(1), x(2), \ldots, x(l)\}$ to the machine assignment vector MA = {$u(1), u(2), \ldots, u(l)$}. While in the second part, $X^{(2)} = \{x(l+1), x(l+2), \ldots, x(2l)\}$ is converted to the operation sequence vector OS = {$v(1), v(2), \ldots, v(l)$}.

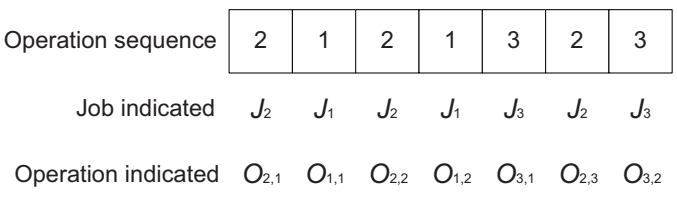

Fig. 4. Illustration of the operation sequence vector.

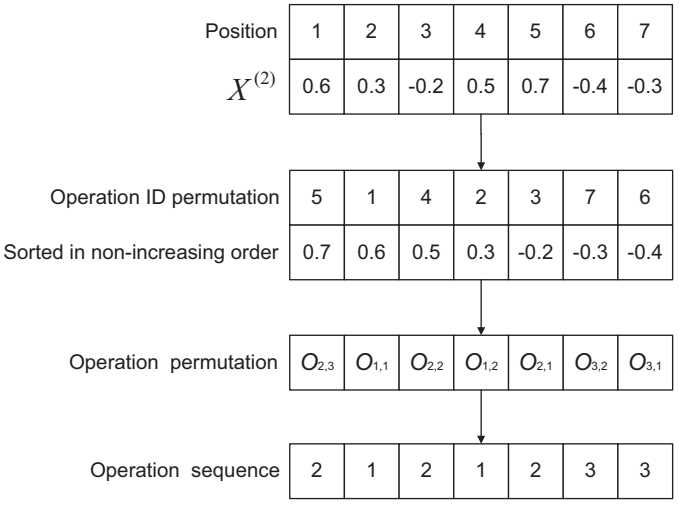

Fig. 5. The conversion from $X^{(2)}$ to the operation sequence vector.

For the conversion in the first part, let $s(j)$ denotes the size of alternative machine set for operation j, where $1 \le j \le l$, what we need to do is map the real number $x(j) \in [-\delta, \delta]$ to the integer $u(j) \in [1, s(j)]$. The concrete procedure is: firstly convert $x(j)$ to a real number belong to $[1, s(j)]$ by linear transformation, then $u(j)$ is given the nearest integer value for the converted real number, which is shown in Eq. (1).

$$u(j) = round\left(\frac{1}{2\delta}(s(j)-1)(x(j)+\delta)+1\right), \quad 1 \le j \le l \tag{1}$$

where $round(x)$ is the function that rounds the number x to the nearest integer. In the particular case $s(j) = 1$, $u(j)$ always equals to 1 no matter what value of $x(j)$ is.

In the second part, the largest position value (LPV) rule [13] is firstly used to construct an ID permutation of operations by ordering the operations in their non-increasing position value. Then each operation ID is replaced by the job number it belongs to. Suppose that we have a vector $X^{(2)} = \{0.6, 0.3, -0.2, 0.5, 0.7, -0.4, -0.3\}$ for the problem in Table 1, then an example of conversion is illustrated in Fig. 5.

4.3.3. Active decoding of two-vector code

The schedules for a scheduling problem are generally categorized into three classes: active schedule, semi-active schedule, and non-delay schedule [1]. It has been proved that there must exist the optimal schedule in the active schedule when considering the makespan criterion [44]. Therefore, we can decode the two-vector code to an active schedule in order to reduce the search space.

To illustrate the active schedule, the left-shift needs to be first introduced. This strategy is used to shift the operations to an earlier start time and make the schedule as compact as possible. A left-shift is called local left-shift if some operations can be started earlier in time without changing the operation sequence on each machine. Global left-shift is similar to local left-shift, the difference lies in that global left-shift may change the operation sequence on each machine. A schedule is called semi-active if no local left-shift exists. A schedule is called active if no global left-shift exists.

To make sure the decoded schedule of a two-vector code is an active schedule, when we arrange an operation $O_{i,j}$ to be executed on machine M_k with processing time $p_{i,j,k}$, the earliest available idle time interval on M_k is searched for allocating it. If no such time interval, the operation is scheduled at the end of machine M_k. Let $[S_x, E_x]$ denotes an idle time interval for machine M_k, $c_{i,j}$ denotes the

completion time of operation $O_{i,j}$, then the time interval is available for $O_{i,j}$ only when the following inequalities are satisfied:

$$\begin{cases} \max\{S_x, c_{i,j-1}\} + p_{i,j,k} \le E_x, & \text{if} \quad j \ge 2 \\ S_x + p_{i,j,k} \le E_x, & \text{if} \quad j = 1 \end{cases} \tag{2}$$

If the available time interval is found, the start time $s_{i,j}$ of operation $O_{i,j}$ is taken as:

$$s_{i,j} = \begin{cases} \max\{S_x, c_{i,j-1}\}, & \text{if} \quad j \ge 2 \\ S_x, & \text{if} \quad j = 1 \end{cases} \tag{3}$$

Otherwise, let ET_k be the current end time for machine M_k, the start time $s_{i,j}$ is set as follows:

$$s_{i,j} = \begin{cases} \max\{ET_k, c_{i,j-1}\}, & \text{if} \quad j \ge 2 \\ ET_k, & \text{if} \quad j = 1 \end{cases} \tag{4}$$

After all the operations are scheduled, the makespan can be calculated as $C_{\max} = \max_{1 \le k \le r}(ET_k)$. The decoding procedure for the two vector MA and OS is shown in Algorithm 1.

Algorithm 1. The procedure of decoding the two-vector code

1. Get a triple group $[O_{i,j}, M_k, p_{i,j,k}]$ for each operation $O_{i,j}$ by reading the vector MA
2. Transform OS to a list of ordered operations $\{op_1, op_2, \ldots, op_l\}$
3. **for** j = 1 to l **do**
4. Get the corresponding triple group for operation op_j
5. Search the earliest available time interval for op_j on its chosen machine
6. **if** the available time interval is found **then**
7. the operation op_j is allocated here and its start time is set according to Eq. (3)
8. **else**
9. the operation op_j is allocated at the end of its selected machine and its start time is set according to Eq. (4)
10. **endif**
11. **end for**
12. **return** the decoded active schedule and the corresponding makespan

4.4. Initialize the harmony memory

The initialization strategy is important in the evolution algorithms which can affect the convergence speed and the quality of the final solution. To guarantee the initial HM with certain quality and diversity, we generate one harmony vector by a heuristic approach and the remainder are all produced randomly.

A harmony vector $X = \{x(1), x(2), \ldots, x(n)\}$ is randomly produced simply according to flowing formula

$$x(j) = x_{\min}(j) + (x_{\max}(j) - x_{\min}(j)) \times rand(0, 1), \quad j = 1, 2, \ldots, n \tag{5}$$

where $x_{\min}(j) = -\delta$ and $x_{\max}(j) = \delta$ for each dimension j, $rand(0, 1)$ is a random function returning a real number between 0 and 1 with uniform distribution.

For the harmony vector to be heuristically generated, a two-vector code is formed by heuristics in advance, thereafter the two-vector code is converted to a harmony vector X.

4.4.1. Generating a two-vector code based on heuristics

Firstly, we form a schedule for the FJSP by two stages. In the first stage, the approach by localization (AL) [37] heuristic is adopted to assign each operation to a suitable machine. AL takes into account both the processing time of operations and the workloads of machines. In the AL procedure, all the operations are traversed in order, for each operation, the machine with the minimum processing time is selected, then the processing times of rest operations on the selected machine are updated by adding this minimum time. In the second stage, the known dispatching rule, most work remaining (MWR), is used to determine how to sequence the operations on the machines. By using this rule, the job with most remaining work will have the highest priority to be scheduled.

After obtaining the schedule, we encode it to a two vector code. The machine assignment vector can be apparently got directly according to Eq. (6). Then all the operations represented by fixed IDs are sequenced in the scheduling order according to MWR rule, which is represented by $\pi = \{\pi(1), \pi(2), \ldots, \pi(l)\}$. Here, it is not necessary to transform π to OS further because the schedule obtained by heuristics is always feasible.

4.4.2. Converting the two-vector code to the harmony vector

The conversion also consists of two parts just as the illustration in Section 4.3.2.

In the first part related to machine assignment, the transformation is in fact an inverse linear transformation of Eq. (1). But the case of $s(j) = 1$ should be considered alone, $x(j)$ chooses a random value between $[-\delta, \delta]$ when $s(j) = 1$. The transformation can be performed as follows

$$x(j) = \begin{cases} \dfrac{2\delta}{s(j) - 1}(u(j) - 1) - \delta, & s(j) \ne 1 \\ x(j) \in [-\delta, \delta], & s(j) = 1 \end{cases} \tag{6}$$

where $1 \le j \le l$.

Corresponding to the LPV rule, the second half part of the harmony vector, $X^{(2)} = \{x(l+1), x(l+2), \ldots, x(2l)\}$, can be obtained according to the vector $\pi = \{\pi(1), \pi(2), \ldots, \pi(l)\}$ as the following

$$x(\pi(j) + l) = \delta - \frac{2\delta}{l - 1}(j - 1), \quad 1 \le j \le l \tag{7}$$

Suppose that $\pi = \{2, 3, 7, 1, 4, 5, 6\}$, $\delta = 0.6$, then according to Eq. (7), $x(9) = 0.6$, $x(10) = 0.4$, and so on. Finally, the second half part of the harmony vector $X^{(2)} = \{0, 0.6, 0.4, -0.2, -0.4, -0.6, 0.2\}$.

4.5. Improvise a new harmony

A new harmony vector, $X_{new} = \{x_{new}(1), x_{new}(2), \ldots, x_{new}(n)\}$, is generated from the HM based on three rules: memory consideration, pitch adjustment, and random selection. In the memory consideration stage, each decision variable $x_{new}(j)$ of the new harmony vector can be chosen from any value in the specified HM range $(x_1(j)$–$x_{HMS}(j))$. The *HMCR*, which varies between 0 and 1, is the probability of selecting one value from the historical values stored in the HM (memory consideration), while $(1 - HMCR)$ is the probability of randomly choosing the possible range of values (random selection), just as shown in Eq. (8):

$$x_{new}(j) = \begin{cases} x_{new}(j) \in \{x_1(j), x_2(j), \ldots, x_{HMS}(j)\}, & \text{with probability } HMCR \\ x_{new}(j) \in [x_{\min}(j), x_{\max}(j)] & \text{with probability } (1 - HMCR) \end{cases} \tag{8}$$

If $x_{new}(j)$ is chosen from the HM, then the pitch adjustment rule is further applied. This rule uses the *PAR* parameter, which is the rate of pitch adjustment and can be depicted as follows:

$$x_{new}(j) = \begin{cases} x_{new}(j) \pm rand(0, 1) \times bw, & \text{with probability } PAR \\ x_{new}(j) & \text{with probability } (1 - PAR) \end{cases} \tag{9}$$

where bw is an arbitrary distance bandwidth.

To make the HS more adaptive to the FJSP, the proposed HHS employed the modified pitch adjustment rule, which is expressed by Eq. (10)

$$x_{new}(j) = \begin{cases} x_{best}(j), & \text{with probability } PAR \\ x_{new}(j) & \text{with probability } (1 - PAR) \end{cases} \tag{10}$$

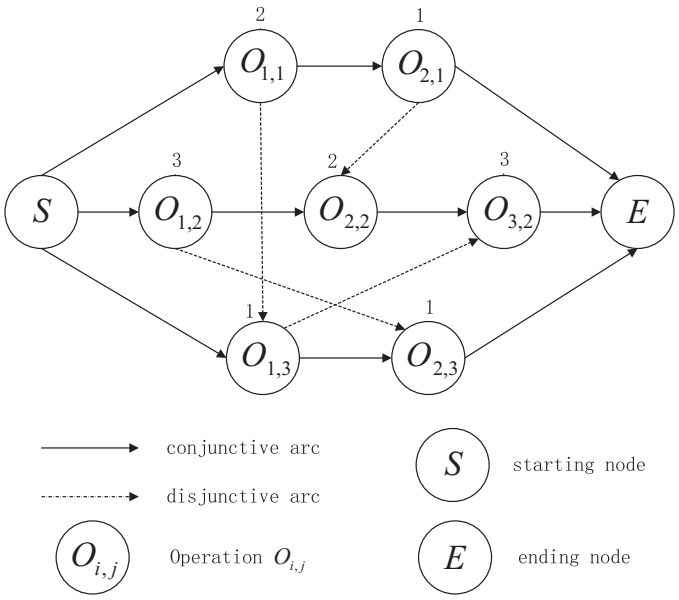

Fig. 6. Illustration of disjunctive graph.

This modification can well inherit a good solution structure of X_{best} and make the algorithm have fewer parameters.

In summary, the main steps of improvising a new harmony in our proposed approach is described in Algorithm 2.

Algorithm 2. The procedure of improvising a new harmony in the HHS

```
for j = 1 to n do
  if rand(0, 1) < HMCR then
    x_new(j) = x_i(j)  where i ∈ {1, 2, ..., HMS}
    if rand(0, 1) < PAR then
      x_new(j) = x_best(j)
    end if
  else
    x_new(j) = x_min(j) + (x_max(j) − x_min(j)) × rand(0, 1)
  end if
end for
```

4.6. Problem-dependent local search

To enhance the searching ability, a local search procedure is incorporated to the HS algorithm by improving each candidate harmony vector generated in the improvisation phase. In the proposed HHS, the local search is applied to the schedule represented by the disjunctive graph rather than directly applied to the harmony vector, which can be helpful to introduce problem-specific knowledge. So, when a harmony vector is to be improved by the local search, it should be firstly decoded to a schedule represented by the disjunctive graph.

4.6.1. Disjunctive graph model

A schedule of FJSP can be represented with disjunctive graph $G = (V, C \cup D)$. In the graph, V represents a set of all the nodes, each node denotes an operation in the FJSP (including dummy starting and terminating operations); C is the set of all the conjunctive arcs, these arcs connect two adjacent operations within one job and the directions of them represent the processing order between two connected operations; D means a set of all the disjunctive arcs, these arcs connect tow adjacent operations performed on the same machine and their direction also show the processing order. The processing time for each operation is generally labeled above the corresponding node and regarded as the weight of the node. Take the problem shown in Table 1 for instance, a possible schedule represented by the disjunctive graph is illustrated in Fig. 6, in which $O_{1,1}, O_{3,1}, O_{2,3}$ are processed in succession on the machine M_1, $O_{1,2}$, $O_{2,2}$ are executed successively on the machine M_2, and $O_{2,1}, O_{3,2}$ are performed in turn on the machine M_3.

The disjunctive graph has the following two basic and important properties:

Property 1. *If a schedule is feasible, then there exists no cyclic paths in the corresponding disjunctive graph.*

Property 2. *If a disjunctive graph is acyclic, then the longest path from the starting node S to the ending node E (critical path) denotes the makespan for the corresponding schedule.*

4.6.2. Neighborhoods based on common critical operations

For the FJSP, most of the effective neighborhoods for the local search are based upon the critical path in the disjunctive graph [21,22,41]. This type of neighborhoods is motivated by the fact that to move an operation on a critical path of the current disjunctive graph maybe decrease the makespan. The move of an operation $O_{i,j}$ is performed in two steps consisting of deletion and insertion [22]:

(1) Delete the node v representing $O_{i,j}$ from its current machine sequence by removing all its disjunctive arcs in the disjunctive graph. Set the weight of node v equal to 0.
(2) Assign $O_{i,j}$ to an machine M_k and choose the position of v in the processing order of M_k, insert the node v by adding its disjunctive arcs and setting the weight of node v equal to $p_{i,j,k}$.

To move an operation on a critical path is profitable because of the following simple theory:

Lemma 1. *Let P be a critical path in the current disjunctive graph, then to move an operation $O_{i,j} \notin P$ cannot reduce the makespan.*

This theory can be understood loosely by a simple way: the longest path P of the current schedule remains present in the disjunctive graph after moving an operation that does not belong to P, so the makespan cannot be reduced.

The greatest advantages of this type of neighborhoods lie in that unnecessary moves are avoided according to Lemma 1 and the size of neighborhoods is effectively reduced compared with moving an arbitrary operation. But it seems not enough, because for the current schedule represented by disjunctive graph, the number of operations on a critical path is usually many still. Therefore, to form a new schedule with shorter makespan from the current one, lots of operations may be tried to be moved, this is obviously very time consuming. To speed up the local search procedure in our proposed algorithm, we improved this type of neighborhood by introducing the concept of common critical operations, which is inspired by the fact that there usually exist more than one critical path in the disjunctive graph. The common critical operations can be explained by the following definitions:

Definition 1. One operation is a critical operation only when its earliest starting time equals to its latest starting time.

Definition 2. One critical path in the disjunctive graph is a path from the starting node S to the ending node E composed of only critical operations.

Definition 3. One operation is called a common critical operation only when it is a critical operation and located on all the critical paths in the disjunctive graph. Specially, all the critical operation are common critical operations when there exists only one critical path.

The earliest starting time of an operation is the earliest time at which the execution of this operation can begin, while the latest starting time of an operation means the latest time at which the execution of this operation can begin without delaying the

makespan. Regard to the common critical operations, the following property is helpful to get all of them in the disjunctive graph:

Property 3. *There are two critical operations op_i and op_j, the time intervals $[s_i, c_i]$ and $[s_j, c_j]$ represent their processing durations, respectively. If their exits overlapping part between $[s_i, c_i]$ and $[s_j, c_j]$, then neither of op_i and op_j is a common critical operation.*

Proof. Because op_j is a critical operation, it must be located on one critical path P. Suppose that op_i is a common critical operation, so it is located on all the critical paths, certainly on the path P. Therefore, op_i and op_j are on the same critical path P. So their processing durations $[s_i, c_i]$ and $[s_j, c_j]$ cannot be overlapped. It is contradictory, so the supposition is not valid, op_i is not a common critical operation. Similarly, op_j is not a common critical operation either. □

The move strategy in the improved neighborhood structure is to move common critical operations rather than operations on one critical path. Its rationality can be expressed bellow:

Corollary 1. *To move an operation $O_{i,j}$ that is not a common critical operation in the disjunctive graph cannot reduce the makespan.*

Proof. Because $O_{i,j}$ is not a common critical operation, there must exist one critical path P not containing the operation $O_{i,j}$, that is, $O_{i,j} \notin P$. According to Lemma 1, this corollary is true. □

More unnecessary moves are further avoided and the size of neighborhoods gets smaller by only moving common critical operations. Consequently, the computational load is reduced and the efficiency is improved in the local search procedure. For example, there exist two critical paths in Fig. 6, which are $S \to O_{1,1} \to O_{1,2} \to O_{2,2} \to O_{2,3} \to E$ and $S \to O_{2,1} \to O_{2,2} \to O_{2,3} \to E$, but only the operation $O_{2,2}$ is common critical operation. If moving move operations on a critical path, then at least there are there candidate operations to be moved, whereas only the operation $O_{2,2}$ is considered when moving common critical operations.

As illustrated above, we have specified what operation is to be deleted: common critical operation. Next, the insertion strategy presented in [41] is adopted to insert the deleted operation so that an acceptable neighborhood of the current schedule is obtained. Let G' is the disjunctive graph yielded by deleting a common critical operation $O_{i,j}$ in G, $ES'_{x,y}$ and $EC'_{x,y}$ respectively denote the earliest starting time and the earliest completion time for each remained operation $O_{x,y}$ in G'. In order not to increase the makespan after insertion, take the makespan of G as the "required" makespan of G' and calculate the latest starting time $LS'_{x,y}$ for each operation $O_{x,y}$ according to this makespan. Then, for any two adjacent operations $O_{\alpha,\beta}$ and $O_{\mu,\nu}$ on the machine M_k, the max idle time interval $[S_x, E_x]$ is formed, where

$$\begin{cases} S_x = EC'_{\alpha,\beta} = ES'_{\alpha,\beta} + p_{\alpha,\beta,k} \\ E_x = LS'_{\mu,\nu} \end{cases} \tag{11}$$

Due to precedence constrains between operations, whether the max idle time interval $[S_x, E_x]$ is available for $O_{i,j}$ to insert can be justified by the following:

$$\begin{cases} \max\{S_x, EC'_{i,j-1}\} + p_{i,j,k} < \min\{E_x, LS'_{i,j+1}\}, & j \neq 1, j \neq n_i \\ S_x + p_{i,j,k} < \min\{E_x, LS'_{i,j+1}\}, & j = 1 \\ \max\{S_x, EC'_{i,j-1}\} + p_{i,j,k} < E_x, & j = n_i \end{cases} \tag{12}$$

When there are no such idle time intervals to insert each of the common critical operations, delete an arbitrary operation further after deleting a common critical operation, so as to form more and larger idle time intervals, and then to insert them in the same way as mentioned above. Obviously, it is more time consuming, so this process is only executed when it is failing to move one common critical operation. To sum up, the procedure of generating an acceptable neighborhood in the local search is shown in Algorithm 3.

Algorithm 3. The procedure of generating an acceptable neighborhood

```
Get all the critical operations in the current disjunctive graph G
according to Property 1
To pick out all the common critical operations
{cop_1, cop_2, ..., cop_w} from the critical operations by means of the
Property 3
for i = 1 to w do
  Delete cop_i from G to yield G'
  Calculate all the max idle time intervals in G'
  if an available time interval is found for cop_i then
    Insert the operation cop_i to yield G''
    return the disjunctive graph G''
  end if
end for
for i = 1 to w do
  Delete cop_i from G to yield G'
  for each operation op_j in G' do
    Delete op_j from G' to yield G''
    Calculate all the max idle time intervals in G''
    if two available time intervals are found for cop_i and op_j then
      Insert cop_i and op_j to yield G'''
      return the disjunctive graph G'''
    end if
  end for
end for
return an empty disjunctive graph
```

4.6.3. Local search procedure

Let $X = \{x(1), x(2), \ldots, x(l), x(l+1), x(l+2), \ldots, x(2l)\}$ be the harmony vector to be improved by the local search. First, the vector X is transformed to a schedule represented by the disjunctive graph through the two-vector code and decoding. Then an iterative process to find the acceptable neighborhood of the current schedule is continuously executed until the maximal iterations are met or no acceptable neighborhood is found. Finally, the schedule obtained by this process is transformed to the harmony vector $Y = \{y(1), y(2), \ldots, y(l), y(l+1), y(l+2), \ldots, y(2l)\}$. Algorithm 4 describes the computation procedure of local search in detail.

Algorithm 4. The procedure of local search in the HHS

```
Convert the harmony vector X to a two-vector code
Decode the two-vector code to a feasible schedule represented by
the disjunctive graph G
while the maximal iterations are not met do
  To yield G' from G by executing Algorithm 3
  if G' is not empty then
    G ← G'
  else
    Exit the while loop
  end if
end while
Get the vector Y^(1) = {y(1), y(2), ..., y(l)} directly from the machine
assignment in G according to Eq. (6)
Sort all the operations in the non-decreasing order of the earliest
start time and yield the operation ID permutation π = {π(1), π(2),
..., π(l)}
Rearrange elements in X^(2) = {x(l+1), x(l+2), ..., x(2l)} to yield
Y^(2) = {y(l+1), y(l+2), ..., y(2l)} combining π and LPV rule
return the harmony vector Y
```

4.7. Update harmony memory

In the proposed HHS, the HM will be updated after a new harmony vector $X'_{new} = \{x'_{new}(1), x'_{new}(2), \ldots, x'_{new}(n)\}$ is generated by the local search, If the new harmony vector X'_{new} is better than the existing worst harmony X_{worst} in the HM in terms of the objective function value, then X'_{new} will replace X_{worst} and become the new member of the HM, otherwise nothing should be done. If X'_{new} is included in the HM, the label updating of the best and worst harmony vectors in the HM must be further considered.

5. Experimental details

5.1. Experimental setup

To test the performance of the proposed HHS for solving the FJSP, the algorithm is implemented in Java language and run on a PC with 2.83 GHz and 15.9 GB of RAM memory. We report our experimental results and compare them with the results obtained by other authors. Several sets of problem instances are considered in our experiment as follows:

(1) Kacem data: The data set is a set of three problems (problem 8×8, problem 10×10, problem 15×10) from Kacem et al. [37]. Problem 8×8 is a P-FJSP instance that consists of eight jobs with 27 operations that can be performed on eight machines. Problem 10×10 is a T-FJSP instance that consists of 10 jobs with 30 operations that can be processed on 10 machines. Problem 15×10 is a T-FJSP instance that consists of 15 jobs with 56 operations that can be executed on 10 machines.
(2) Fdata: The data set is a set of 20 problems from Fattahi et al. [31]. These problems are categorized into two classes: small size flexible job shop scheduling problems (SFJS01:10), and medium and large size flexible job shop scheduling problems (MFJS01:10). The number of jobs ranges between 2 and 12, the number of machines ranges between 2 and 8, the number of operations for each job ranges between 2 and 4, and the number of operations for all the jobs ranges between 4 and 48.
(3) BRdata: The data set is a set of 10 problems from Brandimarte [35]. The number of jobs ranges between 10 and 20, the number of machines ranges between 4 and 15, the flexibility per operation ranges between 1.43 and 4.10, and the number of operations for all the jobs ranges between 55 and 240.
(4) DPdata: The data set is a set of 18 problems from Dauzère-Pérès and Paulli [21]. The number of jobs ranges between 10 and 20, the number of machines ranges between 5 and 10, the flexibility per operation varies between 1.13 and 5.02, and the number of operations for all the jobs ranges between 196 and 387.
(5) BCdata: The data set is a set of 21 problems from Barnes and Chambers [45]. The number of jobs ranges between 10 and 15, the number of machines ranges between 4 and 15, the flexibility per operation varies between 1.07 and 1.30, and the number of operations for all the jobs ranges between 100 and 225.
(6) HUdata: The data set is a set of 129 problems from Hurink et al. [20]. These problems are divided into three subsets, Edata, Rdata and Vdata, depending on the average number of alternative machines for each operation (flexibility). The number of jobs ranges between 6 and 30, the number of machines ranges between 5 and 15, the flexibility per operation ranges between 1.15 and 7.5, and the number of operations for all the jobs ranges between 36 and 300.

Due to the natural of nondeterminacy for the proposed algorithms, we carry out 30 independent runs for each instance from Kacem data, Fdata, and BRdata, and only run five independent times for each instance from the other data sets to just have an overall performance comparison. The following four metrics are recorded to describe the computational results:

(1) $BC_{\max}$: the best makespan obtained among several runs.
(2) AV ($C_{\max}$): the average makespan obtained by several runs.
(3) *SD*: the standard deviation of makespan obtained by several runs.
(4) AV (CPU): the average computational time to achieve the solutions among several runs in terms of seconds.

Table 3
Parameters setting of *NI* and $loop_{\max}$ for each data set.

Data set	*NI*	$loop_{\max}$
Kacem data	400	20
Fdata	3000	30
BRdata	5000	200
DPdata	3000	300
BCdata	3000	300
HUdata	3000	300

In the FJSP literature, $BC_{\max}$ is mainly considered when comparing two algorithms [23,26,38,39]. But to show high efficiency of our proposed HHS, we also compare AV (CPU) with several algorithms on some problem instances, in which this metric is also reported. However, the different computing hardware, programming platforms and coding skills used in each algorithm make this comparisons notoriously problematic [46]. So, when concerning AV (CPU), we enclose the original name of the CPU, the programming language, and the original computational time for the corresponding algorithm, which in order to have a rough and relatively fair understanding of the efficiency of referred algorithms. Furthermore, in Section 5.3, we re-implement two high performing GA algorithms respectively proposed in [26,27], and carry out detail statistical comparisons of our HHS with the two GA algorithms, in order to further validate the algorithmic advantage of the HHS in solution quality. So the results of the two GA algorithms reported in Section 5.3 are obtained by our re-implemented algorithms, while the results of comparative algorithms shown in Section 5.2 are directly from the literature. It is worthy mentioning that the kind of comparison in Section 5.3 seems to be a little bit out of the existing practice of experimental study on the FJSP in the literature.

The parameters in our proposed HHS include harmony memory size (*HMS*), harmony memory considering rate (*HMCR*), pitch adjusting rate (*PAR*), the bound factor (δ), the number of improvisations (*NI*), and the max iterations of local search ($loop_{\max}$). In our experiment, we set $HMS = 5$, $HMCR = 0.95$, $PAR = 0.3$, $\delta = 1$. As for the parameter *NI* and $loop_{\max}$, we adjust them according to different data sets in order to obtain the satisfying solutions in acceptable computational time. The settings of the two parameters for each data set are given in Table 3.

5.2. Computational results and comparisons

The proposed HHS is first tested on three problems of Kacem data. Our computational results are shown in Table 4. Table 5 compares our presented algorithm with the following five algorithms: AL +CGA of Kacem et al. [37], PSO +SA of Xia and Wu [30], PVNS of Yazdani et al. [39], AIA of Bagheri et al. [38], TSPCB of Li et al. [33]. The second column up to fifth one in this table represent the best makespan resulted from AL +CGA, PSO +SA, PVNS. It is obviously that the proposed HHS obtains best results among theses algorithms for all three problems. It is notable that *SD* equals to 0 for each problem in Table 4, so our algorithm not only shows strong ability but also stability on Kacem data. The AIA and TSPCB algorithms have the same performance with HHS regarding the best makespan. But they seem to take much more

Table 4
Results of HHS on Kacem data.

Problem	Proposed HHS			
	$BC_{\max}$	AV ($C_{\max}$)	*SD*	AV (CPU)
8×8	14	14	0	0.00
10×10	7	7	0	0.01
15×10	11	11	0	0.42

Table 5
Comparison between the proposed HHS and five existing algorithms on Kacem data.

Problem	AL + CGA	PSO + SA	PVNS	AIA [a]		TSPCB [b]		Proposed HHS [c]	
				BC_{max}	AV (CPU)	BC_{max}	AV (CPU)	BC_{max}	AV (CPU)
8 × 8	15	15	**14**	**14**	0.76	**14**	4.68	**14**	0.00
10 × 10	**7**	**7**	**7**	**7**	8.97	**7**	1.72	**7**	0.01
15 × 10	24	12	12	**11**	109.22	**11**	9.82	**11**	0.42

Note: The best BC_{max} values for each problem are marked in bold.
[a] The CPU time on a 2.0 GHz processor in C++.
[b] The CPU time on a Pentium IV 1.6 GHz processor in C++.
[c] The CPU time on an Intel 2.83 GHz Xeon processor in Java.

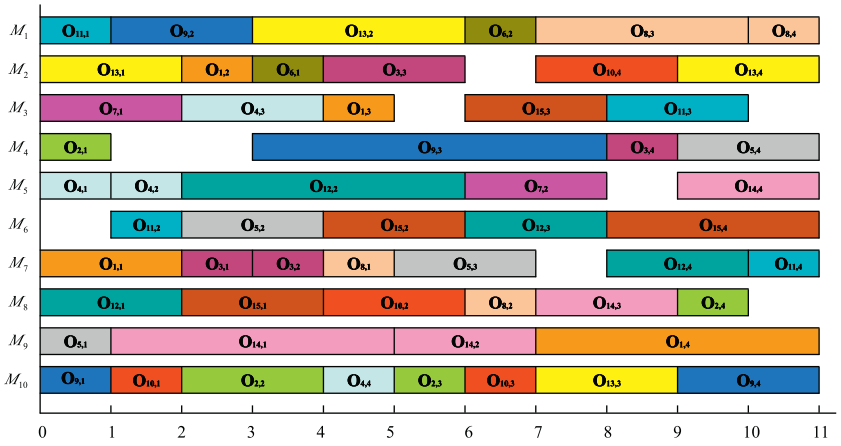

Fig. 7. Gantt Chart of solution of problem 15 × 10.

average CPU time than the proposed HHS. The Gantt Chart of one solution obtained by HHS for the Problem 15 × 10 is illustrated in Fig. 7.

The second data set under investigation is Fdata. Table 6 displays the computational results of our proposed HHS. In this table, the problem names are listed in the first column, the second and third columns represent the number of jobs and machines respectively, the fourth column refers to the lower bound of the problem. In the FJSP literature, one of the best performances on Fdata is reported by the AIA algorithm [38], which outperforms all the six algorithms proposed in [31]. In Table 7, our proposed HHS is compared with AIA. Relative deviation criterion represented by (*dev*) is employed for the comparing, which is defined as

$$dev = \left[\frac{BC_{max}(\text{comp}) - BC_{max}(\text{HHS})}{BC_{max}(\text{comp})}\right] \times 100\% \quad (13)$$

where BC_{max}(HHS) is the best makespan obtained by our algorithm and BC_{max}(comp) is the best makespan of the algorithm that we compares ours to. As can be seen, our proposed HHS dominates the AIA algorithm for all the instances of Fdata. and six new better solutions for the instance MFJS02, MFJS03, MFJS05, MFJS06, MFJS09 and MFJS10 are found. For the instance MFJS05, MFJS09, MFJS10, the solutions are improved significantly. One optimal schedule obtained by HHS for the instance MFJS05 is shown in Fig. 8. When considering the efficiency, HHS and AIA are both enough efficient for the small instance (SFJS01: 10) with nearly zero time consuming, but for the medium and large size instances (MFJS01: 10), HHS displays the obvious superiority, it takes much less average CPU time than AIA. Therefore, the proposed HHS outperforms the AIA algorithm on Fdata no matter concerning the effectiveness or the efficiency.

Another data set that was studied is BRdata. In Table 8, we reported the performance of HHS on this data set. The first and second columns include the name and size of the problem,

Table 6
Results of HHS on Fdata.

Problem	*n*	*m*	LB	Proposed HHS			
				BC_{max}	AV (C_{max})	*SD*	AV (CPU)
SFJS01	2	2	66	66	66	0	0.00
SFJS02	2	2	107	107	107	0	0.00
SFJS03	3	2	221	221	221	0	0.00
SFJS04	3	2	355	355	355	0	0.00
SFJS05	3	2	119	119	119	0	0.00
SFJS06	3	3	320	320	320	0	0.00
SFJS07	3	5	397	397	397	0	0.00
SFJS08	3	4	253	253	253	0	0.00
SFJS09	3	3	210	210	210	0	0.00
SFJS10	4	5	516	516	516	0	0.00
MFJS01	5	6	396	468	468	0	0.01
MFJS02	5	7	396	446	447.53	0.86	0.01
MFJS03	6	7	396	466	466.83	1.05	0.12
MFJS04	7	7	496	554	559.87	5.67	0.06
MFJS05	7	7	414	514	514	0	0.02
MFJS06	8	7	469	634	634.4	2.19	0.01
MFJS07	8	7	619	879	879.07	0.37	0.11
MFJS08	9	8	619	884	885.1	3.36	0.08
MFJS09	11	8	764	1055	1065.2	14.24	0.94
MFJS10	12	8	944	1196	1209.07	8.59	0.69

M_1: $(O_{4,1}$: 0–87$)(O_{2,1}$: 87–301$)$
M_2: $(O_{3,1}$: 0–62$)(O_{5,1}$: 62–185$)(O_{1,2}$: 185–315$)(O_{2,2}$: 315–385$)(O_{5,3}$: 381–481$)$
M_3: $(O_{1,1}$: 0–100$)(O_{7,1}$: 100–245$)(O_{7,2}$: 245–369$)$
M_4: $(O_{6,1}$: 0–154$)(O_{6,2}$: 154–304$)(O_{1,3}$: 315–465$)$
M_5: $(O_{4,2}$: 87–260$)(O_{3,3}$: 260–360$)(O_{7,3}$: 369–514$)$
M_6: $(O_{4,3}$: 260–396$)(O_{2,3}$: 396–491$)$
M_7: $(O_{3,2}$: 62–207$)(O_{5,2}$: 207–293$)(O_{6,3}$: 304–484$)$

Fig. 8. Scheduling obtained for problem MFJS05.

Table 7
Comparison between the proposed HHS and AIA on Fdata.

Problem	n	m	LB	Proposed HHS [a]		AIA [b]		
				BC_{max}	AV (CPU)	BC_{max}	AV (CPU)	*dev* (%)
SFJS01	2	2	66	**66**	0.00	**66**	0.03	0
SFJS02	2	2	107	**107**	0.00	**107**	0.03	0
SFJS03	3	2	221	**221**	0.00	**221**	0.04	0
SFJS04	3	2	355	**355**	0.00	**355**	0.04	0
SFJS05	3	2	119	**119**	0.00	**119**	0.04	0
SFJS06	3	3	320	**320**	0.00	**320**	0.04	0
SFJS07	3	5	397	**397**	0.00	**397**	0.04	0
SFJS08	3	4	253	**253**	0.00	**253**	0.05	0
SFJS09	3	3	210	**210**	0.00	**210**	0.05	0
SFJS10	4	5	516	**516**	0.00	**516**	0.05	0
MFJS01	5	6	396	**468**	0.01	**468**	9.23	0
MFJS02	5	7	396	**446**	0.01	448	9.35	+0.45
MFJS03	6	7	396	**466**	0.12	468	10.06	+0.43
MFJS04	7	7	496	**554**	0.06	**554**	10.54	0
MFJS05	7	7	414	**514**	0.02	527	10.61	+2.47
MFJS06	8	7	469	**634**	0.01	635	22.18	+0.16
MFJS07	8	7	619	**879**	0.11	**879**	24.82	0
MFJS08	9	8	619	**884**	0.08	**884**	26.94	0
MFJS09	11	8	764	**1055**	0.94	1088	30.76	+3.03
MFJS10	12	8	944	**1196**	0.69	1267	30.94	+5.60

Note: The best BC_{max} values for each problem are marked in bold.
[a] The CPU time on an Intel 2.83 GHz Xeon processor in Java.
[b] The CPU time on a 2.0 GHz processor in C++.

respectively. In the third column, the average number of alternative machines for each operation is shown for each problem. In the fourth column, (LB, UB) stands for the lower and upper bounds of the problem. In the literature, GA of Chen et al. [23], GA of Pezzella et al. [26], AIA of Bagheri et al. [38], PVNS of Yazdani et al. [39], TSPCB of Li et al. [33], MA of Raeesi and Kobti [34], and eGA of Zhang et al. [27] treat exactly the same problems. The comparative results between the proposed HHS and these algorithms are listed in Table 9. As can be seen from Table 9, our HHS shows competitive performance on BRdata. In particular, as for BC_{max}, the HHS outperforms GA_Chen in 7 out of 10 instances, outperforms AIA in 5 out of 10 instances, outperforms both GA_Pezzella and PVNS in 4 out of 10 instances, outperforms both MA and eGA in 2 out of 10 instances; the HHS is only outperformed by eGA on the instance MK10. Overall, according to the average improvement, the HHS achieves the best among all the mentioned algorithms. As for the efficiency, the AIA is obviously much more time consuming than our HHS. The TSPCB and eGA both have smaller AV (CPU) values on several instances compared with the HHS. Nevertheless, on the whole, the efficiency of the HHS appears to be at least comparable with that of TSPCB and eGA.

In Table 10, we summarize the computational results of three instance classes with regard to the mean relative error (*MRE*). The first column reports the data set, the second column shows the number of instances for each class. The next five columns report the mean relative error of the best solution obtained by our HHS, by GA of Chen et al. [23], GA of Pezzella et al. [26], AIA of Bagheri et al. [38], and PVNS of Yazdani et al. [39], respectively. The relative error (*RE*) is defined as follows:

$$RE = \left[\frac{BC_{max} - LB}{LB}\right] \times 100\% \quad (14)$$

where BC_{max} is the best makespan obtained by the reported algorithm and *LB* is the best-known lower bound. The table shows that our proposed method displays the best performance among the related ones on all of the three data sets, in respect of *MRE* criterion. And only for the Hurink Rdata, the PVNS yield the same best result with the proposed HHS.

5.3. Further comparisons of HHS with other algorithms

In this subsection, we would like to conduct the statistical comparison of our HHS with GA of Pezzella et al. [26] and eGA of Zhang et al. [27]. The algorithms GA_Pezzella and eGA are carefully re-implemented following all the explanations given by the authors in the original papers. The implemented GA_Pezzella has the similar performance with that reported in [26]. However, the implemented eGA is not so effective as the authors claimed in [27]. If the parameters population size and number of generations are set according to [27], the implemented eGA could not obtain satisfying computational results. Because the two GA algorithms are indeed based on the similar idea, both of which are GA with a mix of initialization strategies, we adopt the setting of the two parameters in [26] for the eGA: the population size is set as 5000

Table 8
Results of HHS on BRdata.

Problem	$n \times m$	Flex	(LB, UB)	Proposed HHS			
				BC_{max}	AV (C_{max})	*SD*	AV (CPU)
MK01	10 × 6	2.09	(36, 42)	40	40	0	0.07
MK02	10 × 6	4.10	(24, 32)	26	26.63	0.49	0.74
MK03	15 × 8	3.01	(204, 211)	204	204	0	0.01
MK04	15 × 8	1.91	(48, 81)	60	60.03	0.18	1.04
MK05	15 × 4	1.71	(168, 186)	172	172.8	0.41	7.47
MK06	10 × 15	3.27	(33, 86)	58	59.13	0.63	60.73
MK07	20 × 5	2.83	(133, 157)	139	139.57	0.50	10.59
MK08	20 × 10	1.43	523	523	523	0	0.02
MK09	20 × 10	2.53	(299, 369)	307	307	0	0.39
MK10	20 × 15	2.98	(165, 296)	205	211.13	2.37	373.01

Table 9
Comparison between the proposed HHS and existing algorithms on BRdata.

Problem	Proposed HHS[a]		GA_Chen		GA_Pezzella		AIA[b]		PVNS		TSPCB[c]		MA		eGA[d]		
	BC_{max}	AV (CPU)	BC_{max}	*dev* (%)	BC_{max}	*dev* (%)	BC_{max}	AV (CPU)	*dev* (%)	BC_{max}	*dev* (%)	BC_{max}	AV (CPU)	*dev* (%)	BC_{max}	*dev* (%)	BC_{max}	AV (CPU)	*dev* (%)
MK01	**40**	0.07	**40**	0	**40**	0	**40**	97.21	0	**40**	0	**40**	2.80	0	**40**	0	**40**	1.60	0
MK02	**26**	0.74	29	+10.34	**26**	0	**26**	103.46	0	**26**	0	**26**	19.31	0	**26**	0	**26**	2.60	0
MK03	**204**	0.01	**204**	0	**204**	0	**204**	247.37	0	**204**	0	**204**	0.98	0	**204**	0	**204**	1.30	0
MK04	**60**	1.04	63	+4.76	**60**	0	**60**	152.07	0	**60**	0	62	40.82	+3.23	**60**	0	**60**	6.20	0
MK05	**172**	7.47	181	+4.97	173	+0.58	173	171.95	+0.58	173	+0.58	**172**	20.23	0	**172**	0	173	7.30	+0.58
MK06	**58**	60.73	60	+3.33	63	+7.94	63	245.62	+7.94	60	+3.33	65	27.18	+10.77	59	+1.69	**58**	15.70	0
MK07	**139**	10.59	148	+6.08	**139**	0	140	161.92	+0.71	141	+1.42	140	35.29	+0.71	**139**	0	144	17.30	+3.47
MK08	**523**	0.02	**523**	0	**523**	0	**523**	392.25	0	**523**	0	**523**	4.65	0	**523**	0	**523**	2.20	0
MK09	**307**	0.39	308	+0.32	311	+1.29	312	389.71	+1.60	**307**	0	310	70.38	+0.97	**307**	0	**307**	30.20	0
MK10	205	373.01	212	+3.30	212	+3.30	214	384.54	+4.21	208	+1.44	214	89.83	+4.21	216	+5.09	**198**	36.60	-3.54
Average improvement		+3.31		+1.31		+1.50		+0.68		+1.99		+0.68		+0.05					

Note: The best BC_{max} values for each problem are marked in bold.
[a] The CPU time on an Intel 2.83 GHz Xeon processor in Java.
[b] The CPU time on a 2.0 GHz processor in C++.
[c] The CPU time on a Pentium IV 1.6 GHz processor in C++.
[d] The CPU time on a Pentium IV 1.8 GHz processor in C++.

Fig. 9. Convergence curves in solving the instance MK06 by the eGA, GA_Pezzella, and HHS.

and the number of generations is set as 1000. The values of all the other parameters of the two algorithms are just kept the same with those recommended in the original papers.

In Table 11, we report the statistical results of HHS, GA_Pezzella and eGA over 30 independent runs on BRdata. From Table 11, the HHS achieves the best AV (C_{max}) in 8 out of 10 instances, while both GA_Pezzella and eGA only achieve two. Regarding to the BC_{max}, the HHS is not dominated by GA_Pezzella and eGA on any instance. As for the efficiency, the HHS also shows superiority in most of instances. However, GA_Pezzella is better than the HHS when tackling the instance MK10 no matter in terms of makespan or computational time. The MK03 and MK09 seem to be relatively easy problems, all the three algorithms can solve them optimally in very short time.

In order to find significant differences among the results obtained by the proposed HHS and the two implemented GA algorithms, the significance test is further carried out. Since the obtained makespan values may present neither normal distribution nor homogeneity of variance, the non-parametric tests are considered to be used according to the recommendations made in [47]. Specifically, the Wilcoxon signed-rank test, a pairwise non-parametric statistical test, is adopted to check whether there are significant differences in the optimization effects of the three algorithms on each problem instance. The results are presented in Table 12, and a level of significance $\alpha = 0.05$ is used in all the tests. As can be seen from Table 12, the HHS is significantly better than GA_Pezzella except for the instances MK03, MK08, and MK10. There is no significant difference between the HHS and GA_Pezzella on the instances MK03 and MK08. As for the instance MK10, GA_Pezzella is significantly better than the HHS. Compared with eGA, there exists no significant difference between the HHS and eGA on the instances MK02, MK03, MK05, and MK08; on all the other instances, the HHS is significantly better than eGA.

Fig. 9 depicts the typical convergence curves of the average makespan obtained over 30 independent runs by eGA, GA_Pezzella, and HHS when solving the instance MK06. From Fig. 9, it can be seen that our HHS could converge fast to lower makespan values than both GA_Pezzella and eGA.

6. Discussion

Based on the simulation tests and comparison study in Section 5, it is safely concluded that the proposed HHS is effective, efficient, and robust for solving the FJSP with makespan criterion.

Table 10
Mean relative error (*MRE*) over best-known lower bound.

Data set	Num	Proposed HHS (%)	GA_Chen (%)	GA_Pezzella (%)	AIA (%)	PVNS (%)
DPdata	18	**3.76**	7.91	7.63	N/A	5.11
BCdata	21	**22.89**	38.64	29.56	N/A	26.66
Hurink Edata	43	**2.67**	5.59	6.00	6.83	3.86
Hurink Rdata	43	**1.88**	4.41	4.42	3.98	**1.88**
Hurink Vdata	43	**0.39**	2.59	2.04	1.29	0.42

Note: N/A means the corresponding data is not available. The best MRE values for each data set are marked in bold.

Table 11
Statistical results of HHS, GA_Pezzella and eGA over 30 independent runs on BRdata.

Problem	Proposed HHS				Implemented GA_Pezzella				Implemented eGA			
	BC_{max}	AV (C_{max})	*SD*	AV (CPU)	BC_{max}	AV (C_{max})	*SD*	AV (CPU)	BC_{max}	AV (C_{max})	*SD*	AV (CPU)
MK01	40	**40**	0	0.07	40	41.13	0.43	4.62	40	41.33	0.66	0.50
MK02	26	26.63	0.49	0.74	26	27.13	0.43	5.88	26	**26.53**	0.57	8.08
MK03	204	**204**	0	0.01	204	**204**	0	0.21	204	**204**	0	0.00
MK04	60	**60.03**	0.18	1.04	62	63.07	0.69	59.18	64	64.67	0.48	6.19
MK05	172	**172.8**	0.41	7.47	173	173.27	0.52	13.87	173	173	0	19.77
MK06	58	**59.13**	0.63	60.73	61	61.7	0.75	68.68	60	62	0.91	133.32
MK07	139	**139.57**	0.50	10.59	140	140.9	0.66	28.07	140	142	1.34	25.00
MK08	523	**523**	0	0.02	523	**523**	0	0.35	523	**523**	0	0.62
MK09	307	**307**	0	0.39	307	308.77	1.89	169.45	309	309.03	0.18	367.99
MK10	205	211.13	2.37	373.01	205	**209.3**	1.78	232.73	218	220.67	1.12	564.47

Note: The best AV (C_{max}) values for each problem are marked in bold.

Table 12
Significance tests on the makespan obtained on BRdata instances. The *p*-values are from one sided Wilcoxon signed-rank test. The level of significance α is set to 0.05.

Problem	HHS vs. GA_Pezzella			HHS vs. eGA		
	R^+	R^-	*p*-Value	R^+	R^-	*p*-Value
MK01	0	435	<0.0001	0	378	<0.0001
MK02	0	78	0.0012	63	42	0.2611
MK03	0	0	–	0	0	–
MK04	0	465	<0.0001	0	465	<0.0001
MK05	0	66	0.0018	0	21	–
MK06	0	465	<0.0001	0	435	<0.0001
MK07	0	276	<0.0001	0	351	<0.0001
MK08	0	0	–	0	0	–
MK09	0	120	0.0003	0	465	<0.0001
MK10	259.5	65.5	0.0047	0	465	<0.0001

The effectiveness could be attributed to the stress of the balance of exploration and exploitation in designing the HHS algorithm, while the high efficiency could be explained by the simple improvising mechanism and fast convergence of HS, the speed up strategy in the local search, and the hybrid initial scheme.

In comparison to other state-of-the-art meta-heuristics for the FJSP, one of the major advantages of the HHS lies in that it has a simpler algorithm structurer and seems easier to be implemented, nevertheless, it has shown comparable strong search ability both on effectiveness and efficiency. Besides, there are fewer parameters in the HHS, making it more controllable and more suitable for practical use. As for the disadvantage, there is still room for the HHS to be improved for handling some large-scale problems, for example, the HHS could not obtain the best result for the instance MK10 in BRdata among the referred algorithms. This shortage may be remedied by seeking for better strategies to escape the local optimum during the search.

Moreover, it should be noted that our proposed HHS has great flexibility and scalability. First, the components of HS and local search in the HHS are of low coupling, the modification of one is independent on another. Second, the conversion techniques developed in the HHS can also be employed by other continuous evolutionary algorithms to deal with the FJSP. Last, with problem-dependent conversion mechanism and local search, the framework of HHS could be easily adapted to solve other combinational optimization problems.

7. Conclusion and future works

To the best of our knowledge, this was the first report to apply the harmony search algorithm to solve the flexible job shop scheduling problem with makespan criterion. In our proposed method, the converting techniques are developed to make the continuous HS adaptive to solve the discrete FJSP. Through the two-vector code and active decoding, a harmony vector is mapped into a feasible active schedule, which could largely reduces the search space. The initialization scheme integrating heuristic and random strategies is introduced to initial the HM, which makes the HM with certain quality and diversity. To balance the exploration and exploitation of search space, we hybridize the strong global searching ability of HS with local improvement ability of local search. In addition, an improved neighborhood structure based on common critical operations is presented, which not only brings in problem-specific scheduling information but also speeds up the local search procedure.

The proposed algorithm is tested on 201 benchmark problems. The computational results and comparisons show that our presented HHS algorithm is especially competitive to the related ones in terms of solution quality and time requirements. In the future research, the following directions are suggested:

- Since hybrid algorithms incorporating the neural network (NN) and evolutionary meta-heuristic have shown potential in many practical applications [48–50], to introduce the NN into the framework of HS may lead to more effective solution procedures for the FJSP.
- Developing novel and effective neighborhood structures for the FJSP in the local search is meaningful and challenging;
- Real world problems usual involve flexible processing plan and fuzzy constraints [51–54]. However, the investigation on the fuzzy FJSP is still very limited [55–57], which seems to be unjustified.
- Applying HS to other kinds of combination optimization problems may also be a promising direction.

Acknowledgments

This research is supported by National Natural Science Foundation of China (Grant No. 61175110), National S&T Major Projects of China (Grant No. 2011ZX02101-004) and National Basic Research Program of China (973 Program) (Grant No. 2012CB316305).

References

[1] M. Pinedo, Scheduling: theory, algorithms, and systems, OR Spectrum.
[2] P. Van Laarhoven, E. Aarts, J. Lenstra, Job shop scheduling by simulated annealing, Operations Research (1992) 113–125.
[3] E. Nowicki, C. Smutnicki, A fast taboo search algorithm for the job shop problem, Management Science (1996) 797–813.
[4] J. Gonçalves, J. de Magalh aes Mendes, M. Resende, A hybrid genetic algorithm for the job shop scheduling problem, European Journal of Operational Research 167 (1) (2005) 77–95.
[5] D. Lochtefeld, F. Ciarallo, Helper-objective optimization strategies for the job-shop scheduling problem, Applied Soft Computing 11 (6) (2011) 4161–4174.
[6] M.R. Garey, D.S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Mathematics of Operations Research 1 (1976) 117–129.
[7] Z. Geem, J. Kim, et al., A new heuristic optimization algorithm: harmony search, Simulation 76 (2) (2001) 60–68.
[8] K. Lee, Z. Geem, A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice, Computer Methods in Applied Mechanics and Engineering 194 (36) (2005) 3902–3933.
[9] K. Lee, Z. Geem, S. Lee, K. Bae, The harmony search heuristic algorithm for discrete structural optimization, Engineering Optimization 37 (7) (2005) 663–684.
[10] M. Mahdavi, M. Fesanghary, E. Damangir, An improved harmony search algorithm for solving optimization problems, Applied Mathematics and Computation 188 (2) (2007) 1567–1579.
[11] Z. Geem, K. Lee, Y. Park, Application of harmony search to vehicle routing, American Journal of Applied Sciences 2 (12) (2005) 1552–1557.
[12] A. Vasebi, M. Fesanghary, S. Bathaee, Combined heat and power economic dispatch by harmony search algorithm, International Journal of Electrical Power & Energy Systems 29 (10) (2007) 713–719.
[13] L. Wang, Q. Pan, M. Fatih Tasgetiren, Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms, Expert Systems with Applications 37 (12) (2010) 7929–7936.
[14] J. Ser, M. Matinmikko, S. Gil-López, M. Mustonen, Centralized and distributed spectrum channel assignment in cognitive wireless networks: a harmony search approach, Applied Soft Computing 12 (2) (2012) 921–930.
[15] H. Ishibuchi, T. Yoshida, T. Murata, Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling, IEEE Transactions on Evolutionary Computation 7 (2) (2003) 204–223.
[16] K. Tan, S. Chiam, A. Mamun, C. Goh, Balancing exploration and exploitation with adaptive variation for evolutionary multi-objective optimization, European Journal of Operational Research 197 (2) (2009) 701–713.
[17] J. Lin, Y. Chen, Analysis on the collaboration between global search and local search in memetic computation, IEEE Transactions on Evolutionary Computation 15 (5) (2011) 608–623.
[18] P. Brucker, R. Schlie, Job-shop scheduling with multi-purpose machines, Computing 45 (4) (1990) 369–375.
[19] N. Najid, S. Dauzere-Peres, A. Zaidat, A modified simulated annealing method for flexible job shop scheduling problem, in: IEEE International Conference on Systems, Man and Cybernetics, 2002, Vol. 5, IEEE, 2002, p. 6.
[20] J. Hurink, B. Jurisch, M. Thole, Tabu search for the job-shop scheduling problem with multi-purpose machines, OR Spectrum 15 (4) (1994) 205–215.
[21] S. Dauzère-Pérès, J. Paulli, An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, Annals of Operations Research 70 (1997) 281–306.
[22] M. Mastrolilli, L. Gambardella, Effective neighbourhood functions for the flexible job shop problem, Journal of Scheduling 3 (1) (2000) 3–20.
[23] H. Chen, J. Ihlow, C. Lehmann, A genetic algorithm for flexible job-shop scheduling, in: IEEE International Conference on Robotics and Automation, Vol. 2, IEEE, 1999, pp. 1120–1125.
[24] H. Jia, A. Nee, J. Fuh, Y. Zhang, A modified genetic algorithm for distributed scheduling problems, Journal of Intelligent Manufacturing 14 (3) (2003) 351–362.
[25] H. Zhang, M. Gen, Multistage-based genetic algorithm for flexible job-shop scheduling problem, Journal of Complexity International 11 (2005) 223–232.
[26] F. Pezzella, G. Morganti, G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, Computers & Operations Research 35 (10) (2008) 3202–3212.
[27] G. Zhang, L. Gao, Y. Shi, An effective genetic algorithm for the flexible job-shop scheduling problem, Expert Systems with Applications 38 (4) (2011) 3563–3573.
[28] B. Girish, N. Jawahar, A particle swarm optimization algorithm for flexible job shop scheduling problem., in: IEEE International Conference on Automation Science and Engineering, 2009. CASE 2009, IEEE, 2009, pp. 298–303.
[29] S. Rahmati, M. Zandieh, A new biogeography-based optimization (BBO) algorithm for the flexible job shop scheduling problem, The International Journal of Advanced Manufacturing Technology 58 (9) (2012) 1115–1129.
[30] W. Xia, Z. Wu, An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, Computers & Industrial Engineering 48 (2) (2005) 409–425.
[31] P. Fattahi, M. Saidi Mehrabad, F. Jolai, Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, Journal of Intelligent Manufacturing 18 (3) (2007) 331–342.
[32] J. Gao, L. Sun, M. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, Computers & Operations Research 35 (9) (2008) 2892–2907.
[33] J. Li, Q. Pan, P. Suganthan, T. Chua, A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem, The International Journal of Advanced Manufacturing Technology 52 (5) (2011) 683–697.
[34] M.N. Raeesi, Z. Kobti, A memetic algorithm for job shop scheduling using a critical-path-based local search heuristic, Memetic Computing 4 (3) (2012) 231–245.
[35] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Annals of Operations Research 41 (3) (1993) 157–183.
[36] L. Tung, L. Lin, R. Nagi, Multiple-objective scheduling for the hierarchical control of flexible manufacturing systems, International Journal of Flexible Manufacturing Systems 11 (4) (1999) 379–409.
[37] I. Kacem, S. Hammadi, P. Borne, Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems., IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 32 (1) (2002) 1–13.
[38] A. Bagheri, M. Zandieh, I. Mahdavi, M. Yazdani, An artificial immune algorithm for the flexible job-shop scheduling problem, Future Generation Computer Systems 26 (4) (2010) 533–541.
[39] M. Yazdani, M. Amiri, M. Zandieh, Flexible job-shop scheduling with parallel variable neighborhood search algorithm, Expert Systems with Applications 37 (1) (2010) 678–687.
[40] L. Xing, Y. Chen, P. Wang, Q. Zhao, J. Xiong, A knowledge-based ant colony optimization for flexible job shop scheduling problems, Applied Soft Computing 10 (3) (2010) 888–896.
[41] 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 (2011) 1–13.
[42] L. Wang, S. Wang, Y. Xu, G. Zhou, M. Liu, A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem, Computers & Industrial Engineering 62 (4) (2012) 917–926.
[43] M. Gen, Y. Tsujimura, E. Kubota, Solving job-shop scheduling problem using genetic algorithms., in: IEEE International Conference on Computer and Industrial Engineering, Ashikaga, Japan, 1994, pp. 576–579.
[44] S. French, Sequencing and scheduling: an introduction to the mathematics of the job-shop, Ellis Horwood Chichester, 1982.
[45] J.W. Barnes, J.B. Chambers, Flexible Job Shop Scheduling by tabu search, Graduate program in operations research and industrial engineering, The University of Texas at Austin, Technical Report Series, ORP 96-09, 1996.
[46] J. Beck, T. Feng, J. Watson, Combining constraint programming and local search for job-shop scheduling, INFORMS Journal on Computing 23 (1) (2011) 1–14.
[47] J. Demšar, Statistical comparisons of classifiers over multiple data sets, The Journal of Machine Learning Research 7 (2006) 1–30.
[48] A. Bhattacharya, A. Abraham, P. Vasant, C. Grosan, Evolutionary artificial neural network for selecting flexible manufacturing systems under disparate level-of-satisfaction of decision maker, International Journal of Innovative Computing, Information and Control 3 (1) (2007) 131–140.
[49] T. Ganesan, P. Vasant, I. Elamvazuthi, Optimization of nonlinear geological structure mapping using hybrid neuro-genetic techniques, Mathematical and Computer Modelling 54 (11) (2011) 2913–2922.
[50] T. Ganesan, P. Vasant, I. Elamvazuthi, Hybrid neuro-swarm optimization approach for design of distributed generation power systems, Neural Computing & Applications (2012) 1–13.
[51] A. Bhattacharya, P. Vasant, Soft-sensing of level of satisfaction in toc product-mix decision heuristic using robust fuzzy-lp, European Journal of Operational Research 177 (1) (2007) 55–70.
[52] P. Vasant, A. Bhattacharya, B. Sarkar, S. Mukherjee, Detection of level of satisfaction and fuzziness patterns for mcdm model with modified flexible s-curve mf, Applied Soft Computing 7 (3) (2007) 1044–1054.
[53] M. Díaz-Madroñero, D. Peidro, P. Vasant, Vendor selection problem by using an interactive fuzzy multi-objective approach with modified s-curve membership functions, Computers & Mathematics with Applications 60 (4) (2010) 1038–1048.
[54] P. Vasant, I. Elamvazuthi, F. Jeffrey, Fuzzy technique for optimization of objective function with uncertain resource variables and technological coefficients, International Journal of Modeling, Simulation, and Scientific Computing 1 (3) (2010) 349–367.
[55] D. Lei, Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling, Applied Soft Computing 12 (8) (2012) 2237–2245.
[56] Y. Zheng, Y. Li, D. Lei, Multi-objective swarm-based neighborhood search for fuzzy flexible job shop scheduling, The International Journal of Advanced Manufacturing Technology 60 (9–12) (2012) 1063–1069.
[57] D. Lei, X. Guo, Swarm-based neighbourhood search algorithm for fuzzy flexible job shop scheduling, International Journal of Production Research 50 (6) (2012) 1639–1649.