HHS/LNS: An Integrated Search Method for Flexible Job Shop Scheduling

Yuan Yuan*, Hua Xu*
*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, China
Email: yyyxhdy@gmail.com, xuhua@mail.tsinghua.edu.cn

Abstract—The flexible job shop scheduling problem (FJSP) is a generalization of the classical job shop scheduling problem (JSP), where each operation is allowed to be processed by any machine from a given set, rather than one specified machine. In this paper, two algorithm modules, namely, hybrid harmony search (HHS) and large neighborhood search (LNS) are developed for the FJSP with makespan criterion. The HHS is an evolutionary-based algorithm with the memetic paradigm, while the LNS is typical of constraint-based approaches. To form a stronger search mechanism, an integrated search method is proposed for the FJSP based on the two algorithms, which starts with the HHS, and then the solution is further improved by the LNS. Computational simulations and comparisons demonstrate that, the proposed HHS alone can effectively solve some medium to large FJSP instances, when integrated with the LNS, it shows competitive performance with state-of-the-art algorithms on very hard and large-scale problems, some new upper bounds among the unsolved benchmark instances have even been found.

I. INTRODUCTION

The flexible job shop scheduling problem (FJSP) is a generalization of the classical job shop scheduling problem (JSP), where each operation is allowed to be processed by any machine from a given set, rather than one specified machine. The FJSP is more difficult than the classical JSP, because of its additional decision to assign each operation to the appropriate machine. It has been proved that the FJSP is strongly NP-hard even if each job has at most three operations and there are two machines [1].

Due to the computational complexity of the FJSP, exact algorithms are not effective, especially for instances on a large scale. So, meta-heuristics have become the mainstream of research for this problem in the past 10 years. In the initial study on this subject, tabu search (TS) was adopted most successfully to the FJSP [2]–[5]. The TS proposed by Mastrolilli and Gambardella [5] still represents the state-of-the-art until now. Recently, two categories of techniques have been more emphasized to address the FJSP, which are evolutionary-based approaches and constraint-based approaches.

Evolutionary-based approaches attempt to solve the FJSP by using evolutionary algorithms. The basic mechanism for this type of approaches is to encode the scheduling solutions to some form of codes, then the corresponding decoding algorithm is carried out to evaluate the codes during the optimization process. Thus far, many evolutionary meta-heuristics have been studied for the FJSP, such as genetic algorithm (GA) [6], [7], artificial immune algorithm (AIA) [8], artificial bee colony (ABC) algorithm [9], estimation of distribution algorithm (EDA) [10], and so on. Among these existing works, the memetic paradigm [11], which introduces the problem-dependent local search into the evolutionary algorithm, seems to be more promising to produce high quality solutions for the FJSP [7], [9], [10].

Constraint-based approaches are mainly built on a foundation of constraint programming (CP) techniques. To apply CP to the FJSP is natural because the FJSP is a constraint optimization problem in essence. However, the pure CP is only effective for some small size instances of the FJSP due to the exponentially growing search space. Several important advancements in CP, such as discrepancy search (DS) [12], large neighborhood search (LNS) [13] and iterative flattening search (IFS) [14], changed this situation. Most recently, these techniques have been well tested on the FJSP and achieved the excellent performance on some standard benchmarks [15]–[17].

Since both of the two categories of techniques have been applied successfully in the FJSP, therefore, integration of them to form a stronger search mechanism appears to be promising. In this paper, two algorithm modules for the FJSP with makespan criterion are developed: hybrid harmony search (HHS) and large neighborhood search (LNS), standing for the evolutionary-based and constraint-based approaches respectively. The HHS algorithm is designed with memetic paradigm, which explores the search space using harmony search (HS) [18], whereas a local search procedure based on the critical path is embedded in the HS to perform the exploitation. The LNS algorithm is devised to improve the current solution continuously through focusing on re-optimizing its subpart employing CP based search.

The integration of HHS and LNS in this paper is motivated by the following aspects: the HHS algorithm can generate the high quality solutions quickly, however, when the evolution procedure reaches some extent, the solution is hard to be further improved by increasing the iteration times or the population size; the LNS have a strong ability to intensify, but the ability degrades heavily along with the increasing of problem space, and another defect of LNS is that it depends on the initial solution, bad initial solution may lead to a large amount of computation time and poor quality results.
The remainder of this paper is organized as follows. Section II formulates the studied problem. Section III and Section IV introduce the HHS module and the LNS module respectively. How to integrate the two modules into a framework is described in Section V. Afterwards, experimental studies are presented in Section VI. Finally, the paper is summarized in Section VII.

II. PROBLEM FORMULATION

The FJSP is formally formulated 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 \( j \)th operation of job \( J_i \), can be executed on any machine selected among a given subset \( M_{i,j} \subseteq M \). The processing time of each operation is machine dependent. We denote \( p_{i,j,k} \) to be 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 minimizes the makespan. The makespan means the time needed to complete all the jobs and can be defined as \( C_{\text{max}} = \max_{1 \leq i \leq q}(C_i) \), where \( C_i \) is the completion time 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 executed 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 explicitly, a sample instance of FJSP is shown in Table I, 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.

<table>
<thead>
<tr>
<th>Job</th>
<th>Operation</th>
<th>( M_1 )</th>
<th>( M_2 )</th>
<th>( M_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( J_1 )</td>
<td>( O_{1,1} )</td>
<td>2</td>
<td>–</td>
<td>3</td>
</tr>
<tr>
<td>( J_1 )</td>
<td>( O_{1,2} )</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>( J_2 )</td>
<td>( O_{2,1} )</td>
<td>–</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>( J_2 )</td>
<td>( O_{2,2} )</td>
<td>6</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>( J_2 )</td>
<td>( O_{2,3} )</td>
<td>3</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>( J_3 )</td>
<td>( O_{3,1} )</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>( J_3 )</td>
<td>( O_{3,2} )</td>
<td>3</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

III. HYBRID HARMONY SEARCH

A. Outline of HS

The harmony search (HS) [18] is one of the latest population-based evolutionary meta-heuristics. It is originally designed for the continues optimization problem, which is defined as minimize (or maximize) \( f(X) \) such that \( x(j) \in [x_{\text{min}}(j), x_{\text{max}}(j)] \), where \( f(X) \) is the objective function, \( X = \{x(1), x(2), \ldots, x(n)\} \) is a candidate solution consisting of \( n \) decision variables, and \( x_{\text{min}}(j) \) and \( x_{\text{max}}(j) \) are the lower and upper bounds for each decision variable, respectively. To solve the problem, the HS maintain a harmony memory (HM) which consists of harmony vectors and can be represented as \( HM = \{X_1, X_2, \ldots, X_{\text{HM}}\} \), where HMS denotes the harmony memory size (HMS). \( X_i = \{x_i(1), x_i(2), \ldots, x_i(n)\} \) is the \( i \)th harmony vector in the HM. The best and worst harmony vectors in the HM are separately labeled as \( X_{\text{best}} \) and \( X_{\text{worst}} \). The workflow of HS is simply described as follows. Firstly, the initial harmony memory is generated from a uniform distribution in the ranges \( [x_{\text{min}}(j), x_{\text{max}}(j)] \), where \( 1 \leq j \leq n \). Then, a new candidate harmony is generated from the HM based on three rules: memory consideration, pitch adjustment and random selection. In this paper, the modified pitch adjustment rule [19] is adopted, which can well inherit a good solution structure of \( X_{\text{best}} \) and make the algorithm have fewer parameters. The pseudocode of generating a new candidate harmony, so called “improvisation” in the HS, is depicted in Fig 1, where HMCR is the harmony memory considering rate (HMCR), PAR is the pitch adjusting rate (PAR), \( \text{rand}(0,1) \) is a random function returning a real number between 0 and 1 with uniform distribution. Following the improvisation, the HM is updated by replacing the worst harmony in the HM with the generated harmony only if its fitness (measured in terms of the objective function) is better than that of the worst harmony. The procedures of improvising and updating are repeated until the termination criterion is satisfied. For more details about HS, refer to [18], [20].

```plaintext
1: for each \( j \in [1,n] \) do
2:   if \( \text{rand}(0,1) < \text{HMCR} \) then \( \triangleright \) memory consideration
3:     \( x_{\text{new}}(j) \leftarrow x_i(j) \), where \( i \in \{1,2,\ldots,\text{HMS}\} \);
4:   else if \( \text{rand}(0,1) < \text{PAR} \) then \( \triangleright \) pitch adjustment
5:     \( x_{\text{new}}(j) \leftarrow x_{\text{best}}(j) \);
6:   end if
7: else \( \triangleright \) random selection
8:   \( x_{\text{new}}(j) \leftarrow x_{\text{new}}(j) \in [x_{\text{min}}(j), x_{\text{max}}(j)] \);
9: end if
10: end for
```

Fig. 1. Pseudocode of the improvisation.

B. Procedure of HHS

The procedure of the proposed HHS algorithm is based on the HS and its algorithmic flow is depicted in Fig 2. Unlike the basic HS, a local search procedure is performed to improve the harmony vector generated in the improvisation phase. In the HHS, the local search is applied to the schedule solution 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 \( X_{\text{new}} \) is to be improved by the local search, it should be firstly mapped to a disjunctive graph, then the
improved schedule is remapped to the harmony vector \( X'_{\text{new}} \) just after the local search. The disjunctive graph is a sort of representation of the schedule solution for the JSP or the FJSP, details of which can be referred in [2], [21].

1: Set the algorithm parameters and the stopping criterion.
2: Initialize the HM randomly.
3: Evaluate each harmony in the HM and label the \( X_{\text{best}} \) and \( X_{\text{worst}} \).
4: while the stopping criterion is not met do
5: Improve a new harmony \( X_{\text{new}} \) from the HM.
6: Perform local search to \( X_{\text{new}} \) and yield \( X'_{\text{new}} \).
7: Update the HM.
8: end while
9: return the best harmony found.

C. Adaptation of HHS to the FJSP

1) Representation and Initialization: In the proposed HH-S, a harmony vector, \( X = \{x(1), x(2), \ldots, x(n)\} \) is still represented as an \( n \)-dimensional real vector. The dimension \( n \) satisfies the constraint \( n = 2l \), where \( l \) is the number of all the operations in the FJSP to solve. 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 \( X^{(2)} = \{x(l+1), x(l+2), \ldots, x(2l)\} \) 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 III-C2. What's more, to deal with the problem conveniently, the intervals \([x_{\text{min}}(j), x_{\text{max}}(j)]\), \( j = 1, 2, \ldots, n \), are all set as \([-\delta, \delta]\), \( \delta > 0 \).

The population is initialized randomly and uniformly. A harmony vector, \( X = \{x(1), x(2), \ldots, x(n)\} \), is randomly produced according to the following formula:

\[
x(j) = -\delta + 2\delta \times \text{rand}(0, 1), \quad j = 1, 2, \ldots, n
\]

2) Two-vector Code: Because the harmony is represented as a real number vector, to directly map it to the schedule solution seems to be troublesome. The two-vector code is adopted as the bridge between them, which can be illustrated in Fig 3.

This kind of code consists of two vectors: machine assignment vector and operation sequence vector, corresponding to two subproblems in the FJSP. For explaining the two vectors, 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 Table II for the instance shown in Table I. 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 Table II.

The machine assignment vector, denoted by \( \text{MA} = \{u(1), u(2), \ldots, u(l)\} \), is an array of \( l \) integer values. In the vector, \( u(j) \), \( 1 \leq j \leq l \), represents the operation \( j \) choose the \( u(j) \) th machine in its alternative machine set. For the problem in Table I, a possible machine assignment vector is shown in Figure 4 and its meaning is also revealed. For example, \( u(1) = 2 \) indicates that the operation \( O_{1,1} \) choose the 2nd machine in its alternative machine set, that is machine \( M_3 \).

As for the operation sequence vector, expressed as \( \text{OS} = \{v(1), v(2), \ldots, v(l)\} \), it’s the ID permutation of all the operations. The order of occurrence for each operation in the OS indicates its scheduling priority. Take the instance shown in Table I for example, a possible operation sequence vector is represented as \( \text{OS} = \{3, 1, 4, 2, 6, 5, 7\} \). The OS can be directly 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. It must be noted that not all the ID permutations are feasible for the operation sequence vector because of the designated priority of operations lying in an job. That is to say, the operations within an job should keep the relative priority order in the OS.

The decoding of the two-vector code is divided into two stages. The first step is to assign each operation to the selected machine according to the MA. Then the second is to treat all the operations one by one according to their order in the OS, each operation under treatment is allocated in the best available processing time for the corresponding machine. A schedule generated by this way can be guaranteed to be an active schedule [22]. To encode a schedule solution to the two-vector code is more direct, the MA is obtained just by the machine assignment in the schedule, while the OS is got through sorting all the operations in the non-decreasing order of the earliest start time.

3) Conversion Techniques: Conversions include two different types, just as shown in Fig 3.
The first type of conversion has two situations. One is in the evaluation process, the harmony vector needs to be converted to the two-vector code first, then the code is decoded to an active schedule, the fitness of the harmony is given the value of makespan corresponding to the schedule. The other is in the local search process, the harmony vector to be improved is converted to the two-vector code, then the two-vector code is decoded to the schedule represented by the disjunctive graph. For a given harmony vector, \( X = \{x(1), x(2), \ldots, x(l), x(l+1), x(l+2), \ldots, x(2l)\} \), where \(-\delta \leq x(j) \leq \delta, j = 1, 2, \ldots, 2l\), this kind of conversion 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 \( MA = \{u(1), u(2), \ldots, u(l)\} \). Let \( s(j) \) denotes the size of alternative machine set for operation \( j \), where \( 1 \leq j \leq l \), what needs to do is map the real number \( x(j) \) 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 Equation (2).

\[
u(j) = \text{round}(\frac{1}{2\delta}(s(j) - 1)(x(j) + \delta) + 1), \quad 1 \leq j \leq l \tag{2}\]

\( \text{round}(x) \) is the function that rounds the number \( x \) to the nearest integer. In the second part, \( X^{(2)} = \{x(l+1), x(l+2), \ldots, x(2l)\} \) is converted to the operation sequence vector \( OS = \{v(1), v(2), \ldots, v(l)\} \). To realize this transformation, the largest position value (LPV) rule [23] is used to construct an ID permutation of operations by ordering the operations in their non-increasing position value. However, as mentioned in Section III-C2, the obtained permutation may be not feasible for the OS. So, the repair procedure is further carried out to adjust the relative order of operations within an job in the permutation. Suppose that we have a vector \( X^{(2)} = \{0.6, -0.4, 0.5, -0.2, 0.7, 0.3, -0.3\} \) for the instance in Table I, then an example of conversion is illustrated in Figure 5.

\[
x(j) = \begin{cases} \frac{2\delta}{s(j)-1}(u(j) - 1) - \delta, & s(j) \neq 1 \\ x(j) \in [-\delta, \delta], & s(j) = 1 \end{cases}
\]

where \( 1 \leq j \leq l \). For the second part, the vector \( X^{(2)} = \{x(l+1), x(l+2), \ldots, x(2l)\} \), is obtained by rearranging elements in the old \( X^{(2)} \) before improved. The rearrangement makes the new \( X^{(2)} \) correspond to the OS of the improved schedule according to the LPV rule.

4) Local Search Strategy: The local search is employed to enhance the local exploitation ability of HS. In the proposed HHS, a local search procedure similar to the one used in [9] is adopted, in which the neighborhood of a solution is obtained by moving one operation on a critical path (critical operation) in the disjunctive graph. The move of an operation is performed in two steps consisting of deletion and insertion, which means to delete an operation in the disjunctive graph, then insert it to the available place to yield no worse schedule. The difference of our local search procedure lies in that, when the local optimum of moving one critical operation is found, the current solution is further improved by moving two operations simultaneously, at least one of which is critical. Because moving two operations is more time consuming, it is only executed when it is fail to move one critical operation. The procedure of generating an acceptable neighborhood in the local search is described in Fig 6. For more details about how to move the operation, refer to [9]. The local search procedure is terminated when the maximal iterations are met or no worse solution is obtained.
IV. LARGE NEIGHBORHOOD SEARCH

A. Outline of LNS

The large neighborhood search (LNS) [13] is a powerful technique, which combines constraint programming (CP) and local search to solve optimization problems. Unlike typical local search that makes small changes to the current solution, such as moving one or two operations in the scheduling, the LNS selects a subset of variables to relax from the problem. Once variables are chosen, unassign them while keeping the remaining variables fixed (destruction), and then search for a better solution by re-optimizing only the unassigned variables (construction). Steps of destruction and construction are iterated in the LNS until the termination condition is met. The basic architecture of LNS is outlined in Fig. 7.

1: Produce initial solution.
2: while termination condition is not met do
3: Choose a subset of variables to relax.
4: Fix the remaining variables.
5: if search finds improvement then
6: Update current solution.
7: end if
8: end while

Fig. 7. LNS Architecture.

The main idea of LNS is simple in fact. Through the operator of destruction, the original problem is reduced, then the CP is employed to solve the reduced problem which can overcome the defect of CP existing in exploring the large search space. The key benefit from a CP perspective is that the strong CP propagation techniques can be exploited to prune the search space more effectively when compared with the pure tree search [24].

B. Constraint-based Model for the FJSP

For illustrating the constraint-based model for the FJSP, two more notations are added based on the Section II. Let \( \sigma_{i,j}, \mu_{i,j} \) denote the start time and the selected machine of operation \( O_{i,j} \) in the schedule, respectively. Then, a solution to the FJSP is composed of pairs of values \((\sigma_{i,j}, \mu_{i,j})\) for all the operations. The solution is feasible if it satisfies the following three kinds of constraints:

- Precedence Constraint: the operation within an job must satisfy the designated priority order. It is formulated as follows: \[ \forall i \in [1,u], \forall j \in [1,n_i - 1], \sigma_{i,j} + p_{i,j,\mu_{i,j}} \leq \sigma_{i,j+1}. \]
- Resource Constraint: one operation can only be processed on its available machines, that is \( \forall O_{i,j}, \mu_{i,j} \in M_{i,j}. \)
- Capacity Constraint: the machine can only process only one operation at a time, which is formulated like this: \( \forall O_{x,y,} \), \( O_{x,y,} \) if \( \mu_{x,y} = \mu_{x,y} \), then \( \sigma_{x,y} + p_{x,y,\mu_{x,y}} \leq \sigma_{x,y} \) or \( \sigma_{x,y} + p_{x,y,\mu_{x,y}} \leq \sigma_{x,y}. \)

The FJSP is to minimize the makespan \( C_{\text{max}} = \max_{1 \leq i \leq u, 1 \leq j \leq n_i} \{\sigma_{i,j} + p_{i,j,\mu_{i,j}}\} \) under the above three constraints.

C. Destruction Procedure

In the destruction procedure, some of variables are chosen to relax while the other are kept fixed. For the FJSP, the partial-order schedule (POS) relaxation [25] is adopted, which chooses a set \( R \) of operations first, then each of the remaining operations is fixed on the current machine and operations executed on the same machine are kept their relative precedence order in the current schedule. Let \( (\sigma, \mu) \) is the current schedule solution and \( (\sigma', \mu') \) is the one to be constructed. The POS relaxation can be formulated as the following: \( \forall O_{x,y} \), \( O_{x,y} \notin R \) and \( \mu_{x,y} = \mu_{x,y} \), if \( \sigma_{x,y} + p_{x,y,\mu_{x,y}} \leq \sigma_{x,y} \), then \( \sigma_{x,y} + p_{x,y,\mu_{x,y}} \leq \sigma_{x,y} \) and \( \mu_{x,y} = \mu_{x,y} \), \( \mu_{x,y} = \mu_{x,y} \). It is beneficial for the POS relaxation to fix only the relative precedence order between the remaining operations, rather than the actual start time, which leaves more room for the re-optimization.

How to choose the set \( R \) is called the neighborhood heuristic in LNS. In this paper, the time-window neighborhood heuristic is adopted, which generates the time window \([t_{\text{min}}, t_{\text{max}}]\) randomly and \( R \) is the set of all operations processed between the interval \([t_{\text{min}}, t_{\text{max}}]\). To further intensify the search, an additional neighborhood is constructed from the time-window neighborhood by selecting a subset \( Q \subseteq R \) and the machine assignment of operations in \( Q \) is fixed, that is \( \forall O_{i,j} \in Q, \mu(O_{i,j}) = \mu(O_{i,j}). \)

D. Construction Procedure

The construction procedure is to form the new schedule solution based on the current one by using CP search. Our search method is naive, which relies on the fact that, once each operation is assigned to an available machine and operations within the same machine are completely ranked under constraints, the minimal makespan is equal to the sum of durations of the operations along the longest path of the precedence graph (disjunctive graph). Briefly, the search is performed as follows: assign operations to the chosen machines, considering the operations with the fewest machine selections first; and then rank operations on each machine separately, giving priority to those machines with the least slack; following the rank, the earliest and latest start time for each operation and the makespan are calculated according to the precedence graph; finally set the earliest start time as the start time for all operations. To avoid searching for too long, the failure limit is set for each construction procedure of LNS. Moreover, the deep first search is used as our search controller, and because the search is implemented on the constraint-based system, it can well inherit the benefit of constraint propagation in CP.

V. FRAMEWORK OF HHS/LNS

Given the complexities of the two existing algorithm modules, our method to integration in this paper is relatively naive. The HHS module is executed at first. By the mechanism of HS that replace the worst harmony in the HM at each iteration, the HM can also be regarded as an elite solution pool at the end of running the HHS algorithm. Because the assignment of machines to operations is critical for the FJSP, we can extract
some good machine assignment information from the elite solutions in the HM before entering the LNS module. The procedure of extraction works as follows: for each operation, the machine selected by the best harmony in the HM is added to its available machine list, while the machine among the remaining that selected most frequently by the other elite solutions is added if its selected frequency is no less than $\tau$, where $\tau$ is the adjustable parameter. So, each operation has reduced machines to choose via the extraction. Following the extraction, the best solution found by the HHS is used as the initial solution for the LNS, the LNS is then run for a specified CPU time to further improve it and return the best solution found.

As the description above, our integration features in two folds: firstly, the best solution found by the HHS provide a good start point for the LNS; secondly, the extracted machine assignment information restricts the search of LNS to a more promising problem space, which could strengthen the ability of intensification.

Obviously, there is much more we could do. However, this simple integration directly and concisely achieves our motivations for this research, just as described in Section I.

VI. EXPERIMENTAL STUDY

A. Experimental Setup

To test the performance of the proposed algorithms, the HHS module is implemented in Java while the LNS module is implemented on top of the COMET [26] system. Both algorithms are run on an Intel 2.83GHz Xeon processor with 15.9Gb of RAM.

In our experiment, two well known FJSP benchmark sets are involved. One is the BRdata set of FJSP instances from Brandimarte [2], which is used to validate the effectiveness of the proposed HHS algorithm. The integrated search method HHS/LNS is evaluated on a set of much harder and larger instances from DPdata, which is provided by Dauzére-Pérès and Paulli [4].

Due to the nature of nondeterminacy for the proposed algorithms, we carry out five runs on each problem instance in order to obtain meaningful results, just as some existing FJSP literatures suggested [5]–[7]. Four metrics including the best makespan (Best), the average makespan (Ave), the standard deviation of makespan (SD), and the average computational time ($T_{av}$) obtained among five runs are applied to describe the computational results. To show the effectiveness, we compare the best makespan obtained by the proposed algorithm with several representative algorithms in the FJSP literature. Relative deviation criterion represented by (dev) is employed for the comparing, which is defined as

$$dev = \left[\frac{(MK_{comp} - MK_{proposed})}{MK_{comp}}\right] \times 100\% \quad (4)$$

where $MK_{proposed}$ and $MK_{comp}$ are the best makespan values obtained by our method and the comparative algorithm, respectively.

The parameters in the HHS algorithm module include harmony memory size (HMS), harmony memory considering rate (HMCR), pitch adjusting rate (PAR), the bound factor ($\delta$), the number of generations (NG), and the maximum iterations of local search ($iter_{max}$). The LNS algorithm module has three parameters: the failure limit for each construction procedure ($maxFail$), the probability of the operation in $R$ belonging to $Q$ ($P_t$), and the maximum CPU time limit ($T_{max}$). For the integrated method HHS/LNS, there is another parameter $\tau$ that is mentioned in Section V besides the parameters in the HHS and LNS. In our experiment, the parameter settings are summarized in Table III, which are adjusted according to the test on different values for parameters and the experimental results.

B. Computational Results and Comparisons

Firstly, BRdata is investigated to validate the effectiveness of our proposed HHS. The computational results and comparisons are summarized in Table IV. The first and second columns include the name and size of the instance, respectively. In the third column, the average number of alternative machines for each operation is shown for each instance, which is called “flexibility” in the FJSP. In the fourth column, (LB, UB) stands for the lower and upper bounds of the instance. The best known solution (BKS) for each instance in the literature is listed in the fifth column. The best makespan obtained by the HHS is compared with four recently proposed evolutionary-based algorithms, including GA of Pezzella et al. [6], AIA of Bagheri et al. [8], ABC of Wang et al. [9] and BEDA of Wang et al. [10]. The bold values indicate that the corresponding algorithm found the best among the five approaches. As can be seen from Table IV, the proposed HHS compares favorably with the aforesaid algorithms on BRdata. In fact, the HHS outperforms GA in 4 out of 10 instances, outperforms AIA in 5 out of 10 instances, and outperforms both ABC and BEDA in 2 out of 10 instances. For the instances MK06 and MK10, the HHS obtains the better solution than any of them. Measures concerning average relative deviation also reveal that all the four existing evolutionary-based algorithms are dominated by the proposed HHS. Compared with the best known solutions, the HHS matches 8 out of 10 ones. Regarding for the computational results, the HHS not only shows strong ability but also stability and efficiency. Half of the instances are solved with the SD equal to 0, while the rest with small SD
values. Moreover, all 10 instances are computed in no more than 3 minutes except for the instance MK10. Based on the simulation tests and comparison study, it is safe to conclude that the HHS is enough for solving some medium to large FJSP instances effectively, efficiently, and robustly.

We also run the proposed HHS/LNS on BR data, but this is not necessary to solve the BR data instances by the integrated method. The reasons may be that the solutions yielded by the HHS alone reach or are quite close to the actual optimum values of instances.

To demonstrate the effectiveness of the HHS/LNS, it is evaluated on 12 instances in DP data. The DP data instance has more operations and larger operation processing times than BR data instance in general, empirical evidence with various solution approaches [4, 5, 7, 15] supports that DP data is one of the most complex benchmark data set in the literature. In Table V, the computational results and comparisons are reported. The HHS/LNS is compared with three state-of-the-art algorithms, which are TS of Mastrolilli and Gambardella [5], hGA of Gao et al. [7] and CDDS of Himida et al. [15]. Best and Ave denote the best and average makespan obtained among the five independent runs respectively for the algorithms HHS/LNS, TS and hGA. Different from three above algorithms, the CDDS is deterministic by nature, and it is not a single algorithm in fact, it consists of a group of four deterministic algorithms (CDDS-N1, CDDS-N2, CDDS-N3, and CDDS-N4). Best and Ave for the CDDS represent the best and average makespan got by the four algorithms respectively. From Table V, it can be seen that our results are quite competitive with state-of-the-art algorithms. The proposed HHS/LNS have a general superior performance in terms of Best and Ave compared with TS, hGA and CDDS. As for the Best, the HHS/LNS outperforms both TS and CDDS in 8 out of 12 instances, and outperforms hGA in 11 out of 12 instances, the HHS/LNS does not yield the best only for 4 instances, but the results are slightly worse, which are in average 0.3% from the best ones. Overall, the HHS/LNS outperforms TS, hGA and CDDS by 0.07%, 0.34%, and 0.03% on total 12 instances in terms of average dev, respectively. It is worth noting that the HHS/LNS obtains a remarkable 8 new best known solutions to instances 01a, 02a, 03a, 05a, 06a, 08a, 11a, 12a in DP data, among which the instances 01a and 03a are solved optimally. Concerning the computational time of the HHS/LNS, it is notoriously problematic to make comparisons with other algorithms because of the differences lying in the computing hardware, software engineering decisions and coding skill. However, given very recent related literatures [17, 27], the allocated CPU time on large-scale FJSP instances for the HHS/LNS is quite acceptable and reasonable.

![Table IV](image)

**TABLE IV**

<table>
<thead>
<tr>
<th>Instance</th>
<th>n x m</th>
<th>Flex (LB, UB)</th>
<th>BKS</th>
<th>HHS (%)</th>
<th>GA (%)</th>
<th>AIA (%)</th>
<th>ABC (%)</th>
<th>BEDA (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MK01</td>
<td>10 x 6</td>
<td>2.09 (36, 42)</td>
<td>40</td>
<td>40.0</td>
<td>3.87</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>MK02</td>
<td>15 x 6</td>
<td>4.10 (24, 32)</td>
<td>26</td>
<td>26.2</td>
<td>5.79</td>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>MK03</td>
<td>15 x 8</td>
<td>3.01 (204, 211)</td>
<td>204</td>
<td>204.0</td>
<td>36.60</td>
<td>204</td>
<td>204</td>
<td>204</td>
</tr>
<tr>
<td>MK04</td>
<td>15 x 8</td>
<td>1.91 (48, 81)</td>
<td>60</td>
<td>60.0</td>
<td>13.30</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>MK05</td>
<td>15 x 4</td>
<td>1.71 (168, 186)</td>
<td>172</td>
<td>172.8</td>
<td>35.78</td>
<td>173</td>
<td>+0.58</td>
<td>172</td>
</tr>
<tr>
<td>MK06</td>
<td>10 x 15</td>
<td>3.27 (33, 86)</td>
<td>58</td>
<td>59.4</td>
<td>111.65</td>
<td>63</td>
<td>+6.35</td>
<td>60</td>
</tr>
<tr>
<td>MK07</td>
<td>20 x 5</td>
<td>2.83 (133, 157)</td>
<td>139</td>
<td>139.0</td>
<td>26.16</td>
<td>139</td>
<td>+0.71</td>
<td>139</td>
</tr>
<tr>
<td>MK08</td>
<td>20 x 10</td>
<td>1.43 (533)</td>
<td>533</td>
<td>533.0</td>
<td>0.00</td>
<td>533</td>
<td>533</td>
<td>533</td>
</tr>
<tr>
<td>MK09</td>
<td>20 x 10</td>
<td>2.53 (299, 369)</td>
<td>307</td>
<td>307.0</td>
<td>0.00</td>
<td>307</td>
<td>307</td>
<td>307</td>
</tr>
<tr>
<td>MK10</td>
<td>20 x 15</td>
<td>2.98 (165, 296)</td>
<td>197</td>
<td>203.0</td>
<td>437.69</td>
<td>212</td>
<td>4.74</td>
<td>208</td>
</tr>
</tbody>
</table>

**Table VI summarizes the mean relative error (MRE) of the best results obtained by the pure HHS, pure LNS and HHS/LNS on 12 DP data instances.**

![Table VI](image)

**TABLE VI**

<table>
<thead>
<tr>
<th>Instances</th>
<th>HHS (%)</th>
<th>LNS (%)</th>
<th>HHS/LNS (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>01a−02a</td>
<td>2.35</td>
<td>17.73</td>
<td>1.53</td>
</tr>
</tbody>
</table>

Table VI summarizes the mean relative error (MRE) of the best results obtained by the pure HHS, pure LNS and HHS/LNS on 12 DP data instances. The relative error (RE) is defined as \( RE = \frac{(MK - LB)}{LB} \times 100\% \), where \( MK \) is the best makespan obtained by the reported algorithm and \( LB \) is the best-known lower bound. To make a relative
fair comparison with the integrated method, $T_{\text{max}}$ of the pure LNS here is extended to 2000 seconds. From Table VI, it is obviously that the HHS/LNS performs the more effective search than the pure algorithms.

VII. CONCLUSION AND FUTURE WORK

In this paper, two algorithm modules, HHS and LNS, have been developed for the FJSP with makespan criterion. An integrated search method HHS/LNS is established on the base of them. The HHS/LNS is in fact a two-stage algorithm, which starts by executing the HHS, and then the LNS is adopted to further improve the solution obtained by the HHS. Empirical results on two well known benchmark sets illustrate that, the proposed HHS alone can effectively deal with some medium to large FJSP instances, while the HHS/LNS shows the competitive performance with state-of-the-art algorithms on very hard and large-scale problems. In the future work, the proposed HHS/LNS will be experimented on the large-scale practical production data. We also plan to take a closer investigation into the empirical characteristics of the presented HHS and LNS, which is beneficial for us to design more effective integrated method for the FJSP.

APPENDIX

Detailed scheduling schemes of the new best known solutions that were presented in Section VI-B are available at http://learn.tsinghua.edu.cn:8080/2010210742/NBKS.rar.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China (Grant No: 60875073, 61175110), National S&T Major Projects of China (Grant No: 2009ZX02001, 2011ZX02101-004) and National Basic Research Program of China (973 Program) (Grant No2012CB316300).

REFERENCES


