A hybrid evolutionary algorithm to solve the job shop scheduling problem

Article *in* Annals of Operations Research · February 2013

Impact Factor: 1.22 · DOI: 10.1007/s10479-013-1332-5

T. C. E. Cheng

The Hong Kong Polytechnic …

751 PUBLICATIONS **16,705** CITATIONS

Zhipeng Lü

Huazhong University of Scie…

52 PUBLICATIONS **620** CITATIONS

Ann Oper Res
DOI 10.1007/s10479-013-1332-5

A hybrid evolutionary algorithm to solve the job shop scheduling problem

T.C.E. Cheng · Bo Peng · Zhipeng Lü

Abstract This paper presents a Hybrid Evolutionary Algorithm (HEA) to solve the Job Shop Scheduling Problem (JSP). Incorporating a tabu search procedure into the framework of an evolutionary algorithm, the HEA embraces several distinguishing features such as a longest common sequence based recombination operator and a similarity-and-quality based replacement criterion for population updating. The HEA is able to easily generate the best-known solutions for 90 % of the tested difficult instances widely used in the literature, demonstrating its efficacy in terms of both solution quality and computational efficiency. In particular, the HEA identifies a better upper bound for two of these difficult instances.

Keywords Job shop scheduling · Evolutionary algorithm · Recombination operator · Population updating

1 Introduction

Given a set $J = \{1, \dots, n\}$ of n jobs and a set $M = \{1, \dots, m\}$ of m machines, the job shop scheduling problem (JSP) seeks to find a feasible schedule of the operations on the machines that minimizes the makespan or the maximum job completion time, C_{max}, i.e., the finishing time of the last operation completed in the schedule. Each job $j \in J$ consists of

The research was supported in part by the Hong Kong Scholars Programme and the National Natural Science Foundation of China under grant number 61100144 and 61272014.

T.C.E. Cheng
Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
e-mail: Edwin.Cheng@polyu.edu.hk

B. Peng · Z. Lü (✉)
School of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, P.R. China
e-mail: zhipeng.lui@gmail.com

B. Peng
e-mail: 283224262@qq.com

Published online: 23 February 2013

n_j ordered operations $O_{j,1}, \ldots, O_{j,n_j}$, each of which must be processed on one of the m machines. Let $O = \{0, 1, \ldots, o, o+1\}$ denote the set of all the operations to be scheduled, where operations 0 and $o+1$ are dummies, have no duration, and represent the initial and final operations, respectively. Each operation $k \in O$ is associated with a fixed processing duration d_k. Each machine can process at most one operation at a time and once an operation begins processing on a given machine, it must complete processing on that machine without interruption. Furthermore, let p_k be the predecessor operation of operation $k \in O$. The operations are interrelated by two kinds of constraints. First, operation $k \in O$ can only be scheduled if the machine on which it is processed is idle. Second, precedence constraints require each operation $k \in O$ to be scheduled after its predecessor operation p_k has been completed.

A schedule can be represented by a vector of finishing times of all the operations $(F_1, \ldots, F_o, F_{o+1})$. Let E_h be the set of operations being processed on machine $h \in M$. We can formulate JSP as follows:

$$Minimize\ C_{max} = F_{o+1} \tag{1}$$

subject to:

$$F_{p_k} + d_k \le F_k; \quad k = 1, \ldots, o+1, \tag{2}$$

$$F_j - F_i \ge d_i \quad \text{or} \quad F_i - F_j \ge d_j; \quad (i, j) \in E_h,\ h \in M, \tag{3}$$

$$F_i \ge 0; \quad k = 1, \ldots, o+1. \tag{4}$$

The objective function (1) minimizes the makespan, i.e., the finishing time of operation $o+1$. Constraint (2) imposes the precedence relations between operations of the same job, while constraint (3) imposes the precedence relations between operations on the same machine. Finally, constraint (4) requires the finishing times of all the operations to be non-negative.

JSP is one of the most important scheduling problems that arise in situations where a set of activities have to be performed by a set of scarce resources. Job shop scheduling arises naturally in a variety of real-world applications, such as software development planning (Barba and Valle 2010), production planning (Uzsoy et al. 1994), computer system control (Bensana et al. 1986), machine (port) scheduling (Yin et al. 2013), and so on. Moreover, the application potential of JSP is much greater than might be imagined due to the possibilities of imposing a variety of side constraints and objectives on the problem in an explicit manner (Essafi et al. 2008; Guyon et al. 2012; Lancia et al. 2011; Pezzella et al. 2008), which makes JSP a general model for many real-life scheduling problems that exist in operations and supply chain management.

As a canonical NP-hard problem, JSP has been widely considered as one of the most challenging scheduling problems (Garey et al. 1976). This is illustrated by the fact that a relatively small instance with ten jobs and ten machines proposed by Muth and Thompson (1963) had remained unsolved for more than 25 years, and until now no benchmark JSP is solved to optimality for $n \times m$ instances, where $n = 20$ and $m = 20$.

During the past decades, JSP has attracted much attention from a significant number of researchers and a large number of heuristic and metaheuristic algorithms have been proposed to find optimal or near-optimal solutions for JSP, which include:

- *Heuristic Procedures based on Priority Rules*: French (1982), and Ingimundardottir and Runarsson (2011).
- *Shifting Bottleneck Heuristic*: Adams et al. (1988), and Balas and Vazacopoulos (1998).

- *Simulated Annealing (SA)*: van Laarhoven et al. (1992), Lourenco (1995), and Kolonko (1999).
- *Tabu Search (TS)*: Taillard (1994), Nowicki and Smutnicki (1996, 2005), and Zhang et al. (2007, 2008).
- *Genetic Algorithms (GA)*: Gen et al. (1994), Della Croce et al. (1995), and Kobayashi et al. (1995).
- *Memetic Algorithms*: Yang et al. (2008), Hasan et al. (2009), and Ren and Wang (2012).
- *GRASP*: Aiex et al. (2003) and Fernandes et al. (2007).

Among the state-of-the-art algorithms for JSP, several important aspects of the hybrid evolutionary algorithm for JSP are missing in the previous literature: First, the diversity of the population is not thoroughly considered, which is usually implemented by the population updating strategy. A good population updating strategy should be able to maintain a pool of high-quality and diversified solutions. Second, the recombination operator should consider the problem structure of JSP such that elite components of the parent solution are inherited in the offspring solutions, while keeping the relative differences between the latter and their parent solutions. Third, the incorporation of a strong local search procedure into the evolutionary algorithm can significantly enhance its search capability. Moreover, a good tradeoff between intensification and diversification should be carefully considered.

All these issues motivate us to develop a more robust hybrid evolutionary algorithm for solving JSP. In this paper we propose a hybrid evolutionary algorithm (HEA) for JSP that embraces the following distinguishing features: a new recombination operator based on the problem structure of JSP, i.e., a longest common sequence based recombination operator; a similarity-and-quality based population updating strategy; and an effective TS-based local search procedure and a tradeoff between the evolutionary algorithm and local search. We evaluate the effectiveness and efficiency of the HEA computationally by applying it to solve two sets of 60 difficult instances widely used in the literature.

The remainder of this paper is organized as follows: In Sect. 2 we give detailed descriptions of the HEA for JSP. In Sect. 3 we present the computational results and comparisons between the HEA and two best-performing algorithms in the literature for solving two sets of 60 difficult benchmark JSP instances. We conclude the paper and suggest topics for future research in Sect. 4.

2 Hybrid evolutionary algorithm

2.1 Main scheme

Hybrid evolutionary algorithms, also called memetic algorithms, have shown great promise as an effective means for solving a large number of constraint satisfaction and optimization problems. By combining the more global recombining search and the more intensive local search, a hybrid evolutionary algorithm is expected to strike a better balance between the exploration and exploitation of the search space.

In principle, our HEA repeatedly operates between a recombination operator that is used to generate new offspring solutions and a TS procedure that brings individual offspring to a better solution. As soon as the offspring solutions are improved by TS, the population is updated based on two criteria: the quality of the solution and the diversity of the population.

We present the general architecture of the HEA in Algorithm 1. It is composed of four main components: population initialization, a TS procedure, a recombination operator, and a population updating rule. Starting with an initial random population, the HEA uses the TS

Algorithm 1 Pseudo-code of the HEA for JSP

```
1:  Input: $J$, $M$ and $d_k$
2:  Output: $C_{max}$ and the best solution $x^*$ found so far
3:  $P = \{x^1, \ldots, x^p\} \leftarrow$ Population_Initialization()          /* Section 2.3 */
4:  for $i = \{1, \ldots, p\}$ do
5:      $x^i \leftarrow$ Tabu_Search($x^i$)          /* Section 2.4 */
6:  end for
7:  $x^* = arg\ min\{f(x^i) \mid i = 1, \ldots, p\}$
8:  repeat
9:      randomly choose two individuals $x^j$ and $x^k$ from $P$
10:     $(x^{p+1}, x^{p+2}) \leftarrow$ Recombination_Operator($x^j, x^k$)          /* Section 2.5 */
11:     $(x^{p+1}, x^{p+2}) \leftarrow$ Tabu_Search($x^{p+1}, x^{p+2}$)          /* Section 2.4 */
12:     if $f(x^{p+1}) < f(x^*)$ then
13:         $x^* = x^{p+1}$
14:     end if
15:     if $f(x^{p+2}) < f(x^*)$ then
16:         $x^* = x^{p+2}$
17:     end if
18:     $\{x^1, \ldots, x^p\} \leftarrow$ Population_Updating($x^1, \ldots, x^{p+1}, x^{p+2}$)     /* Section 2.6 */
19: until a stop criterion is met
```

procedure to optimize each individual to reach a local optimum (lines 4–6). Then the recombination operator is employed to generate new offspring solutions (line 10), whereupon a new round of TS is again launched to optimize the objective function. Subsequently, the population updating rule decides whether such improved solutions should be inserted into the population and which existing individuals should be replaced (line 18). In the following subsections we discuss the main components of the HEA in detail.

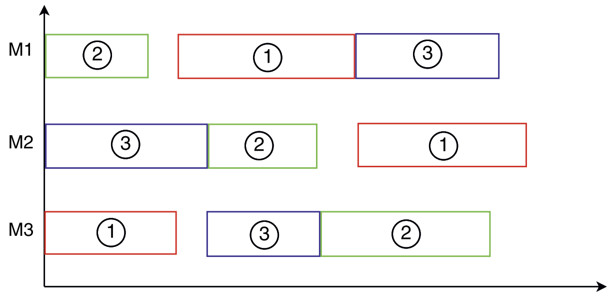
2.2 Solution representation, coding and encoding

Defining an appropriate representation of the individual solutions is one of the most critical issues for designing an optimization algorithm. In this paper we use the solution representation introduced in Ren and Wang (2012), i.e., if the sequence of operations processed on each machine is determined, we can obtain a unique schedule (solution).

For the encoding scheme, we adopt the approach proposed by Falkenauer and Bouffoix (1991), which was recently applied in Ren and Wang (2012), whereby a chromosome is composed of m substrings, corresponding to m different machines. Each substring represents the sequence of operations processed on each machine, denoted by a permutation of the integers from 1 to n. Thus, the total length of a chromosome is $n \cdot m$.

Consider an example of a 3-job, 3-machine problem given in Fig. 1. The corresponding chromosome is: [(2 1 3) (3 2 1) (1 3 2)]. Here, substrings (2 1 3), (3 2 1), and (1 3 2) represent the sequences of operations processed on machines 1, 2, and 3, respectively. Conversely, from the chromosome [(2 1 3) (3 2 1) (1 3 2)], we can also deduce the corresponding schedule as shown in Fig. 1. Obviously, if using this coding and decoding method corresponds to an infeasible schedule, we adopt the method proposed by Ren and Wang (2012) to convert an infeasible schedule into a feasible one.

Fig. 1 Representation of the solution



2.3 Initial population

The initial population of an evolutionary algorithm can be produced by various methods such as priority dispatching rules, diverse insertion heuristics, the shifting bottleneck procedure, and random methods. In general, the initial solution method has little influence on the solution quality, but it may slightly affect the running time. Therefore, in our HEA, feasible solutions of the initial population are generated randomly.

2.4 Tabu search procedure

As demonstrated by Glover (1997), and utilized to solve JSP by Nowicki and Smutnicki (2005), and Zhang et al. (2007), TS is very effective for intensification purposes. In addition, TS has the ability of escaping from local optima. Therefore, TS has become one of the most successful approaches for solving JSP. In contrast to TSSA (which combines TS and SA) proposed by Zhang et al. (2008) that uses the N6 neighborhood, our algorithm uses the N7 neighborhood proposed by Zhang et al. (2007). Our experiments show that this neighborhood can yield powerful search capability.

2.4.1 Move evaluation

When a set of feasible moves have been constructed, each move should be evaluated to calculate or estimate its makespan. In our TS procedure, we employ a fast estimation technique to perform neighborhood move evaluations, which was proposed for N6 in Balas and Vazacopoulos (1998), and is now adapted to N7 neighborhood in our algorithm. According to our experiments, this method is quite efficient in executing our TS procedure.

2.4.2 Tabu list

Tabu list helps the search to forbid re-visiting the recently traversed solutions. The TSAB algorithm in Nowicki and Smutnicki (2005) stores moves in a tabu list. This may be very suitable for the N5 neighborhood that it used. In Zhang et al. (2007), the TSSA algorithm forbids attributes of solutions instead of attributes of moves. Since our algorithm uses the N7 neighborhood, which is similar to the N6 neighborhood of TSSA, the same structure for the tabu list is used. Specifically, if a move consists of interchanging operations u and v, all the operations between u and v, and their positions are stored in the tabu list.

2.4.3 Tabu tenure

TS typically incorporates a *tabu list* as a recency-based memory structure to ensure that solutions visited within a certain span of iterations, called tabu tenure, will not be re-visited (Glover and Laguna 1997). In our implementation, we set

$$RL = \max\{(f - UB)/d_1,\ d_2\}; \tag{5}$$

$$TabuTenure = tt + rand(RL); \tag{6}$$

where d_1, d_2, and tt are given constants, f denotes the makespan of the current solution, and UB represents the best makespan found so far. At the beginning of the search, the gap between f and UB is much greater than d_2 such that the search can converge quickly to local optimum search regions. However, with the execution of the algorithm, the gap between f and UB becomes smaller and smaller such that d_2 dominates the second part of the tabu tenure. In this way, the search will be confined to a small search space.

2.4.4 Move selection

In each iteration, our TS procedure restricts consideration to moves not currently tabu and selects a move that produces the smallest makespan value. In the case where two or more moves have the same best move value, ties are broken randomly. Accompanying this rule, a simple aspiration criterion is applied that permits a move to be selected in spite of being tabu if it leads to a solution better than the current best solution.

However, a situation may arise where all possible moves are tabu and none of them is able to satisfy the aspiration criterion. In such case, our move selection procedure randomly selects a move among all the possible moves, which is similar to TSSA but quite different from the NKTS algorithm by Nasiri and Kianfar (2011), where the first move in the move array is chosen.

2.4.5 Termination criterion of TS

Our TS procedure stops when the optimal solution is found or the total number of iterations within the current round of TS exceeds a given limit α. We call α the tabu search cutoff.

2.5 Recombination operator

The recombination operator is one of the most important ingredients of a hybrid evolutionary algorithm since it can determine the global search capability of the algorithm. It is reasonable that a good recombination operator should not only generate offspring solutions different from their parents but also carry elite components of their parents to offspring solutions. For this purpose, problem specific structures have to be considered in designing the recombination operator. In our algorithm we propose a new recombination operator based on the idea of the so-called *longest common sequence* as follows.

2.5.1 Longest common sequence

Definition 1 (Longest common sequence) For substrings x_j and y_j of two individuals $x = \{x_1, \dots, x_m\}$ and $y = \{y_1, \dots, y_m\}$ ($j \in \{1, \dots, m\}$), there is a longest common sequence between the two substrings x_j and y_j, which may consist of a few jobs. We call the number of such jobs as the similarity degree between substrings x_j and y_j, denoted as $LCS(j)$.

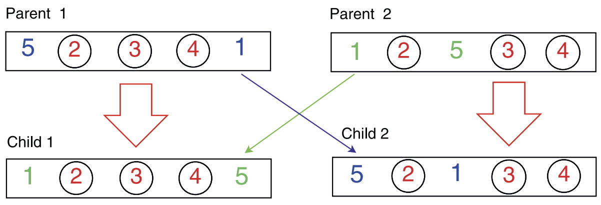


Fig. 2 Recombination process of the first machine for two individuals

We use a 5-job, 3-machine problem as an example to explain this definition. Assume that $x = [(1\ 2\ 3\ 4\ 5)\ (5\ 2\ 3\ 4\ 1)\ (5\ 4\ 3\ 2\ 1)]$ and $y = [(1\ 2\ 3\ 5\ 4)\ (1\ 2\ 3\ 4\ 5)\ (5\ 4\ 3\ 2\ 1)]$ are two given individuals. By comparing the corresponding substrings of these two individuals, we observe that in the first substrings of the two individuals, the longest common sequence is $\{1\ 2\ 3\ 4\}$ or $\{1\ 2\ 3\ 5\}$. Therefore, $LCS(1) = 4$. Similarly, the longest common sequence of the second substrings of the two individuals is $\{2\ 3\ 4\}$, thus $LCS(2) = 3$. Since the third substrings of the two individuals are the same, $LCS(3) = 5$. Evidently, the larger the number of LCS is, the more similar are the corresponding substrings in two individuals.

2.5.2 LCS-based recombination operator

Suppose parent 1 (denoted by t_1) and parent 2 (denoted by t_2) are two parent individuals, and child 1 (denoted by c_1) and child 2 (denoted by c_2) are two children individuals produced by the recombination operator designed in this paper, the recombination operator is presented in Algorithm 2.

Algorithm 2 Pseudo-code of the LCS-based recombination operator

```
Step 1: Divide the jobs of each machine into two complementary sets $A$ and $B$ for two individuals. Set $A$ consists of the jobs in the longest common sequence, while set $B$ consists of the others.
Step 2: Copy set $A$ to $c_1$ by keeping its job positions in $t_1$, while copying set $B$ to $c_1$ by preserving its job order in $t_2$.
Step 3: Copy set $A$ to $c_2$ by keeping its job positions in $t_2$, while copying set $B$ to $c_2$ by preserving its job order in $t_1$.
```

We use a 5-job, 2-machine problem as an example to explain the steps of the recombination operator. Suppose two parent individuals, parent 1 = [(5 2 3 4 1)(1 2 3 5 4)] and parent 2 = [(1 2 5 3 4)(5 4 3 2 1)], are given. First, we divide the jobs of the first machine into two complementary sets {2 3 4} and {1 5}. Because {2 3 4} is the job set in the longest common sequence for parent 1 (5 2 3 4 1) and parent 2 (1 2 5 3 4), we can easily find that $B = \{1\ 5\}$. Second, copy set $A = \{2\ 3\ 4\}$ to c_1 by preserving its job positions in t_1 and copy set $B = \{1\ 5\}$ to c_1 by preserving its job order in t_2. So the job sequence of c_1 on the first machine becomes (1 2 3 4 5). Third, copy set $A = \{2\ 3\ 4\}$ to c_2 by preserving its job positions in t_2 and copy set $B = \{1\ 5\}$ to c_2 by preserving its job order in t_1. Thus, the job sequence of c_2 on the first machine is (5 2 1 3 4). Similarly, on the second machine, we can get the job sequence of c_1 as (3 2 1 5 4) and that of c_2 as (5 4 1 2 3). This process is illustrated in Figs. 2 and 3.

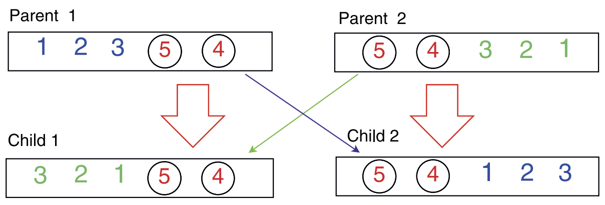


Fig. 3 Recombination process of the second machine for two individuals

2.6 Population updating

In the HEA, when two offspring are obtained by the recombination operator, we improve them by the tabu search procedure and then decide whether the optimized offspring should be inserted into the population, replacing the *worst* solutions in the population according to a similarity-and-quality based goodness score function. The main idea is to favor the inclusion of offspring x^* in the population if x^* is *good* enough (in terms of its objective function value) and is not too *similar* to any current solution in the population.

For JSP, the similarity between two solutions was first studied by Ren and Wang (2012). In this paper we propose a new similarity definition for estimating the similarity degree between two solutions. Unlike Ren and Wang (2012), the basic idea is that we evaluate the longest common sequence of two solutions as the similarity degree, which approximately measures the minimum number of neighborhood moves required to change one solution to another. In order to clarify the exposition, we make use of the following definitions:

Definition 2 (Similarity degree between two solutions) Given two individuals $x = \{x_1, \dots, x_m\}$ and $y = \{y_1, \dots, y_m\}$, the similarity degree between them is defined as the sum of the job number of the longest common sequence of the two solutions, denoted by s_{xy}:

$$s_{xy} = \sum_{i=1}^{m} LCS(i). \tag{7}$$

In fact, the larger the value s_{xy} is, the more similar are the individuals x and y. We can easily observe that if $s_{xy} = n \cdot m$, the individuals x and y are the same solution.

Definition 3 (Similarity degree between one solution and a population) Given a population $P = \{x^1, \dots, x^p\}$ and the similarity degree s_{ij} between any two solutions x^i and x^j $(i, j = 1, \dots, p, i \neq j)$, the similarity degree between a solution x^i $(i = 1, \dots, p)$ and the population P is defined as the maximum similarity degree between x^i and any other solution in P, denoted by $S_{i,P}$:

$$S_{i,P} = \max\{s_{ij} \mid x^j \in P, j \neq i\}. \tag{8}$$

Definition 4 (Goodness score of a solution for a population) Given a population $P = \{x^1, \dots, x^p\}$ and the similarity degree $S_{i,P}$ for any solution x^i $(i = 1, \dots, p)$, the goodness score of solution x^i for population P is defined as:

$$g(i, P) = \beta \widetilde{A}\big(f\big(x^i\big)\big) + (1-\beta)\widetilde{A}(S_{i,P}), \tag{9}$$

where $f(x^i)$ represents the makespan of solution x^i and $\widetilde{A}(\cdot)$ represents the normalized function:

$$\widetilde{A}(y) = \frac{y_{max} - y}{y_{max} - y_{min} + 1}, \tag{10}$$

where y_{max} and y_{min} denote the maximum and minimum values of y in the population P, respectively. The number "1" is used to avoid the possibility of a 0 denominator and β is a constant parameter, which we set at 0.6 in this paper.

It is clear that the greater the goodness score $g(i, P)$ is, the better is the solution x^i. One observes that this goodness score function simultaneously considers the factors of solution quality and population diversity. On the one hand, we should maintain a population of elite solutions. On the other hand, we have to emphasize the importance of the diversity of the solutions to avoid premature convergence of the population.

We present the pseudo-code of our population updating strategy in Algorithm 3. Given two offspring x^{p+1} and x^{p+2} optimized by TS and a population $P = \{x^1, \dots, x^p\}$, we use the following rule to decide whether x^{p+1} and x^{p+2} should be inserted into the population. First of all, x^{p+1} and x^{p+2} are temporarily inserted into the population P (line 3). Then, the goodness score of each solution $x^i \in P'$ is calculated according to Eq. (9) (lines 4–6) and the two worst solutions in the new population P' (with the smallest values of goodness score) are identified, denoted by x^u and x^v (line 7). Then, we remove these two individuals x^u and x^v from the temporary population P' to constitute a new population P (line 8). Note that x^u and x^v may not necessarily be different from x^{p+1} and x^{p+2}.

Algorithm 3 Pseudo-code of the population updating strategy

```
1: **Input**: Population $P = \{x^1, \dots, x^p\}$ and two offspring solutions $\{x^{p+1}, x^{p+2}\}$
2: **Output**: Updated population $P = \{x^1, \dots, x^p\}$
3: Tentatively add $x^{p+1}$ and $x^{p+2}$ to population $P$: $P' = P \cup \{x^{p+1}, x^{p+2}\}$
4: **for** $i = 1, \dots, p, p+1, p+2$ **do**
5:     Calculate the goodness score of $x^i$ $(g(i, P'))$ according to Eq. (9)
6: **end for**
7: Identify two individuals with the worst goodness scores in the new population $P$:
     $x^u = arg\,min\{g(i, P') \mid i = \{1, \dots, p+1, p+2\}\}$
     $x^v = arg\,min\{g(i, P') \mid i = \{1, \dots, p+1, p+2\} \backslash \{u\}\}$
8: Generate new population by removing both two individuals:
   $P = \{x^1, \dots, x^p, x^{p+1}, x^{p+2}\} \backslash \{x^u, x^v\}$
```

3 Computational results

In this section we report extensive experimental results of applying the HEA to tackle 60 notoriously difficult instances available in the literature. We also compare the HEA with two of the best-performing algorithms in the literature, namely TSSA, which stands for the algorithm by Zhang et al. (2008), and HGA, which denotes the newly proposed hybrid genetic algorithm by Ren and Wang (2012).

3.1 Test instances

We considered two sets of JSP instances in our experiments, constituting a total of 60 instances. All these instances are available in the OR-Library.[1]

[1] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/jobshopinfo.html.

Table 1 The settings of some important parameters

Parameter	Section	Description	Value
tt	2.4.3	tabu tenure constant 1	2
d_1	2.4.3	tabu tenure constant 2	5
d_2	2.4.3	tabu tenure constant 3	12
α	2.4.5	tabu search cutoff	12000
β	2.6	goodness score coefficient	0.6
p	2.6	population size	30

– The first set of benchmark JSP consists of 43 basic instances, including three instances denoted as FT06, FT10, and FT20 (size $n \times m = 6 \times 6, 10 \times 10, 20 \times 5$) due to Fisher and Thompson (1963), and 40 instances denoted as LA01-40 (size $n \times m = 10 \times 5, 15 \times 5, 20 \times 5, 10 \times 10, 15 \times 10, 20 \times 10, 30 \times 10, 15 \times 15$) due to Lawrence (1984).
– The second set of benchmark JSP consists of three instances ABZ7-9 (size $n \times m = 20 \times 15$) due to Adams et al. (1988), four instances denoted as YN1-4 (size $n \times m = 20 \times 20$) due to Yamada and Nakano (1992), and ten instances labeled SWV01-10 (size $n \times m = 20 \times 10, 20 \times 15$) due to Storer et al. (1992).

3.2 Experimental protocol

We programmed the HEA in C++ and compiled it using g++ on a PC running the Linux operating system with a AMD 2.8 GHz Opteron CPU. We obtained the computational results without special tuning of the parameters, i.e., all the parameters used in our algorithm were fixed (constant) for all the instances considered. Table 1 gives the descriptions and settings of the parameters used in the HEA, in which the last column denotes the settings for the set of all the instances. Given the stochastic nature of the HEA, we solved each problem instance ten times independently. For each run, we set the total time limit for our algorithm as one CPU hour on our computer.

To measure the solution quality of our algorithm, we calculate the mean relative error (MRE) using the relative deviation formula $\mathrm{RE} = 100 \times (\mathrm{UB}_{solve} - \mathrm{LB}_{best})/\mathrm{LB}_{best}$ for each instance, where LB_{best} is a lower bound and UB_{solve} is the best makespan found by any of the tested algorithms.

3.3 Computational results on the first set of instances

We conducted the first experiment to evaluate the performance of the HEA in solving the first set of 43 basic JSP instances: FT06, 10, 20, and LA01-40. The numbers of their operations range from 36 to 300. Despite their relatively small size, these instances are very hard to solve. For example, FT10 had remained unsolved for 20 years after its proposal. However, these instances have now been solved, some of them by the branch and bound algorithm and some by the approximate algorithm.

Table 2 provides a summary of the performance comparisons between the HEA and TSSA for instances LA01-40. For each group of instances, the best MRE (b-MRE), the average MRE (av-MRE), and the average running time are listed for each algorithm. One observes that the HEA performs quite well and obtains competitive results for these instances. Specifically, the av-MRE value of the HEA is lower than that of TSSA, and the

Table 2 A summary of comparisons between the HEA and TSSA for the 40 LA instances

Problemgroup	Size	HEA			TSSA		
		b-MRE	av-MRE	T_{av} (s)	b-MRE	av-MRE	T_{av} (s)
LA01-05	10×5	0.00	0.00	0.16	0.00	0.00	0.0
LA06-10	10×5	0.00	0.00	0.27	–	–	–
LA11-15	10×5	0.00	0.00	0.39	–	–	–
LA16-20	10×10	0.00	0.00	0.33	0.00	0.00	0.2
LA21-25	15×10	0.00	0.00	12.83	0.00	0.03	13.6
LA26-30	20×10	0.02	0.02	20.14	0.02	0.02	15.2
LA31-35	30×10	0.00	0.00	0.89	–	–	–
LA36-40	15×15	0.00	0.01	217.48	0.03	0.19	36.1

b-MRE of the HEA is also better than that of TSSA, although the average running time of HEA is slightly longer than TSSA.

We show in Table 3 the performance of the HEA compared with that of two best-performing algorithms (TSSA and HGA) in the literature for all the 43 FT and LA benchmark JSP instances in more detail. One finds that for the JSP instance classes FT and LA, the HEA obtains the optimal solutions for all the instances except for one instance LA29, for which it obtains a value of 1153 instead of 1152. TSSA misses the optimal values for instances LA29 and LA40, while HGA misses the optimal values for more instances. From this table, we can conclude that the HEA is quite competitive in solving these basic instances.

3.4 Computational results on the second set of instances

In order to further evaluate the HEA, we tested it with more challenging instances. In this section we report the performance of the HEA in tackling the 17 most difficult instances among the ABZ, YN, and SWV benchmark instances, and present a detailed comparison with other algorithms. The majority of these instances have been considered challenging by many researchers. Among these instances, the optimal solutions are known only for four instances. In particular, instances YN1-4(size $n * m = 20 \times 20$) and SWV06-10(size $n * m = 20 \times 15$) have been widely regarded as two of the most difficult benchmark JSP instances, for which no optimal solutions have been found so far.

Table 4 summarizes the results of this experiment. In this table, the column named UB(LB) lists the best-known upper bound (lower bound) and the next columns named Best, M_{av}, and T_{av} show the best makespan, average makespan, and average computing time in seconds obtained by the HEA over ten independent runs, respectively. The last three columns show the results of TSSA.

From Table 4, one observes that the HEA outperforms TSSA in terms of solution quality. Specifically, the HEA matches ten best-known solutions and improves the upper bound for two instances, namely SWV06 and SWV08, whereas TSSA, which is one of the best-performing algorithms in the literature, finds the best upper bounds for 6 out of 17 instances. It is worth mentioning that the best makespan obtained by the HEA is better than that of TSSA in most cases and the M_{av} value obtained by the HEA is clearly lower than that of TSSA for all the instances. Moreover, it is noteworthy that the HEA finds a better upper bound for instances SWV06-1672 and SWV08-1752, while the previous best upper bounds were SWV06-1675 and SWV08-1755. In terms of the average running time, one finds that

Table 3 Detailed comparisons between the HEA and two best-performing algorithms TSSA and HGA for the 43 basic instances

Problem	Size	UB(LB)	HEA			TSSA			HGA
			Best	M_{av}	T_{av} (s)	Best	M_{av}	T_{av} (s)	Best
FT06	6×6	55	55	55	0.03	–	–	–	55
FT10	10×10	930	930	930	4.37	930	930	3.8	930
FT20	20×5	1165	1165	555	0.39	–	–	–	1165
LA01	10×5	666	666	666	0.18	–	–	–	666
LA02	10×5	655	655	655	0.15	–	–	–	655
LA03	10×5	597	597	597	0.15	–	–	–	597
LA04	10×5	590	590	590	0.14	–	–	–	590
LA05	10×5	593	593	593	0.19	–	–	–	593
LA06	15×5	926	926	926	0.29	–	–	–	926
LA07	15×5	890	890	890	0.19	–	–	–	890
LA08	15×5	863	863	863	0.28	–	–	–	863
LA09	15×5	951	951	951	0.29	–	–	–	951
LA10	15×5	958	958	958	0.31	–	–	–	958
LA11	20×5	1222	1222	1222	0.38	–	–	–	1222
LA12	20×5	1039	1039	1039	0.4	–	–	–	1039
LA13	20×5	1150	1150	1150	0.4	–	–	–	1150
LA14	20×5	1292	1292	1292	0.41	–	–	–	1292
LA15	20×5	1207	1207	1207	0.37	–	–	–	1207
LA16	10×10	945	945	945	0.31	–	–	–	945
LA17	10×10	784	784	784	0.2	–	–	–	784
LA18	10×10	848	848	848	0.22	–	–	–	848
LA19	10×10	842	842	842	0.4	842	842	0.5	844
LA20	10×10	902	902	902	0.55	–	–	–	907
LA21	15×10	1046	1046	1046	14.5	1046	1046	15.2	1046
LA22	15×10	927	927	927	8.41	–	–	–	935
LA23	15×10	1032	1032	1032	0.39	–	–	–	1032
LA24	15×10	935	935	935	34.76	935	936.2	19.8	953
LA25	15×10	977	977	977	6.12	977	977.1	13.8	981
LA26	20×10	1218	1218	1218	0.53	–	–	–	1218
LA27	20×10	1235	1235	1235	7.12	1235	1235	11.7	1236
LA28	20×10	1216	1216	1216	0.45	–	–	–	1216
LA29	20×10	1152	1153	1153	91.97	1153	1159.2	63.9	1160
LA30	20×10	1355	1355	1355	0.66	–	–	–	1355
LA31	30×10	1784	1784	1784	0.89	–	–	–	1784
LA32	30×10	1850	1850	1850	0.93	–	–	–	1850
LA33	30×10	1719	1719	1719	0.91	–	–	–	1719
LA34	30×10	1721	1721	1721	0.87	–	–	–	1721
LA35	30×10	1888	1888	1888	0.83	–	–	–	1888
LA36	15×15	1268	1268	1268	15.3	1268	1268	9.9	1287
LA37	15×15	1397	1397	1397	125.03	1397	1402.5	42.1	1407
LA38	15×15	1196	1196	1196	128.35	1196	1199.6	47.8	1196
LA39	15×15	1233	1233	1233	66.82	1233	1233.8	28.6	1233
LA40	15×15	1222	1222	1222.6	751.9	1224	1224.5	52.1	1229

Table 4 Computational results and comparisons for the 17 most difficult instances

Problem	Size	UB(LB)	HEA			TSSA		
			Best	M$_{av}$	T_{av} (s)	Best	M_{av}	T$_{av}$ (s)
ABZ7	20 × 15	656	657	657.6	662.3	658	661.8	85.9
ABZ8	20 × 15	665(645)	667	667.9	663.4	667	670.3	90.7
ABZ9	20 × 15	678(661)	678	678	303.02	678	684.8	90.2
ABZ7-9			2.05	2.12	542.9	2.10	2.80	88.9
YN1	20 × 20	884(826)	884	886.1	299.5	884	891.3	106.3
YN2	20 × 20	904(861)	904	906.5	1138	907	911.2	110.4
YN3	20 × 20	892(827)	892	894.3	830.2	892	895.5	110.8
YN4	20 × 20	968(918)	968	969.25	555.8	969	972.6	108.7
YN1-4			6.33	6.57	705.88	6.4	6.99	109.1
SWV01	20 × 10	1407	1407	1411.2	908.94	1412	1423.7	142.1
SWV02	20 × 10	1475	1475	1475	292.66	1475	1480.3	119.7
SWV03	20 × 10	1398(1369)	1398	1398.4	443.84	1398	1417.5	139.1
SWV04	20 × 10	1470(1450)	1470	1471.6	1600.38	1470	1483.7	143.9
SWV05	20 × 10	1424	1425	1427.1	1466.23	1425	1443.8	146.7
SWV01-05			0.71	0.83	1133.32	0.78	1.76	109.1
SWV06	20 × 15	1675(1591)	***1672***	1678.9	887.85	1679	1700.1	192.5
SWV07	20 × 15	1594(1446)	1595	1598.9	1490.11	1603	1631.3	190.2
SWV08	20 × 15	1755(1640)	***1752***	1764.6	871.55	1756	1786.9	190
SWV09	20 × 15	1656(1604)	1657	1660.9	1182.24	1661	1689.2	193.8
SWV10	20 × 15	1743(1631)	1743	1754.7	1620.65	1754	1783.7	184.6
SWV06-10			6.48	6.97	1210.48	6.91	8.66	190.2

Newly found upper bounds by the HEA are indicated in bold and italic

the HEA algorithm is comparable to TSSA for most instances. Although the HEA takes a longer running time than TSSA for some cases, the HEA is acceptable because it can obtain better results in these cases.

Finally, it should be noted that unlike many algorithms that work on the basis of better initial solutions generated by some specialized methods, the HEA obtains good results from totally random initial solutions. This implies that the HEA is robust and effective, and its performance could be further improved if some specialized methods are incorporated.

4 Conclusions

In this paper we present the HEA, a hybrid metaheuristic algorithm, for solving the notoriously difficult JSP. The HEA embraces a longest common sequence based recombination operator for generating offspring solutions and an effective TS procedure. Besides, based on a new definition of similarity between two solutions, the HEA also uses a population updating strategy that considers both solution quality and population diversity.

Based on the computational results of applying the HEA to solve two sets of 43 well-known basic JSP instances and 17 more challenging benchmark JSP instances, we show that the HEA obtains highly competitive outcomes in comparison with the best-known results in the literature. For the SWV instances, the HEA improves the upper bound for two instances, whose optimal solutions are still unknown.

The success of the HEA in tackling JSP reveals that it is essential to introduce meaningful diversification mechanisms, highlight the tradeoff between intensification and diversification, and incorporate problem-specific knowledge in designing heuristic search algorithms. Following this spirit, we hope to design even more robust and effective metaheuristic algorithms for solving JSP and other optimization problems. Finally, given that many ideas introduced in this paper are independent of JSP, it is worthwhile to test their effectiveness in treating other difficult permutation and scheduling problems.

Acknowledgements We would like to thank the anonymous referees for their helpful comments that help to improve our paper. We appreciate Dr. Chaoyong Zhang's kind help to explain the TS algorithm for JSP to us.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, *34*(3), 391–401.

Aiex, R. M., Binato, S., & Resende, M. G. C. (2003). Parallel GRASP with path-relinking for jobshop scheduling. *Parallel Computing in Numerical Optimization*, *29*(4), 393–430.

Balas, E., & Vazacopoulos, A. (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, *44*(2), 262–275.

Barba, I., & Valle, C. D. (2010). A job-shop scheduling model of software development planning for constraint-based local search. *International Journal of Software Engineering and its Applications*, *4*(4), 1–16.

Bensana, E., Correge, M., Bel, G., & Dubois, D. (1986). An expert-system approach to industrial job-shop scheduling. In *Proceedings of IEEE international conference on robotics and automation* (Vol. 3, pp. 1645–1650).

Della Croce, F., Tadei, R., & Volta, G. (1995). A genetic algorithm for the job shop problem. *Computers & Operations Research*, *22*(1), 15–24.

Essafi, I., Mati, Y., & Dauzère-Pérès, S. (2008). A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research*, *35*(8), 2599–2616.

Falkenauer, E., & Bouffoix, S. (1991). A genetic algorithm for job shop. In *Proceedings of the 1991 IEEE international conference on robotics and automation* (Vol. 1, pp. 824–829).

Fernandes, S., & Lourenco, H. R. (2007). A GRASP and branch-and-bound metaheuristic for the job-shop scheduling. *Computer Science Evolutionary Computation in Combinatorial Optimization*, *4006*, 60–71.

Fisher, H., & Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial scheduling* (pp. 225–251).

French, S. (1982). *Sequencing and scheduling*. New York: Wiley.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and job-shop scheduling. *Mathematics of Operations Research*, *1*, 117–129.

Gen, M., Tsujimura, Y., & Kubota, E. (1994). Solving job-shop scheduling problems by genetic algorithm. In *IEEE international conference on humans, information and technology systems, man, and cybernetics* (Vol. 2, pp. 1577–1582).

Glover, F., & Laguna, M. (1997). *Tabu search*. Boston: Kluwer Academic.

Guyon, O., Lemaire, P., Pinson, E., & Rivreau, D. (2012). Solving an integrated job-shop problem with human resource constraints. *Annals of Operations Research*. doi:10.1007/s10479-012-1132-3.

Hasan, S. M. K., Sarker, R., Essam, D., & Cornforth, D. (2009). Memetic algorithms for solving job-shop scheduling problems. *Memetic Computing*, *1*(1), 69–83.

Ingimundardottir, H., & Runarsson, T. P. (2011). Supervised learning linear priority dispatch rules for job-shop scheduling. *Learning and Intelligent Optimization*, *6683*, 263–277.

Kobayashi, S., Ono, I., & Yamamura, M. (1995). An efficient genetic algorithm for job shop scheduling problems. In *Proceedings of the 6th international conference on genetic algorithms* (pp. 506–511).

Kolonko, M. (1999). Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, *113*(1), 123–136.

Lancia, G., Rinaldi, F., & Serafini, P. (2011). A time-indexed LP-based approach for min-sum job-shop problems. *Annals of Operations Research*, *186*, 175–198.

Lawrence, S. (1984). *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques* (Technical Report). GSIA, Carnegie Mellon University.

Lourenco, H. R. (1995). Job-shop scheduling: computational study of local search and large-step optimization methods. *European Journal of Operational Research*, *83*(2), 347–364.

Muth, J. F., & Thompson, G. L. (1963). *Industrial scheduling*. Englewood Cliffs: Prentice Hall.

Nasiri, M. M., & Kianfar, F. (2011). A GA/TS algorithm for the stage shop scheduling problem. *Computers and Industrial Engineering*, *61*(1), 161–170.

Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science*, *42*(6), 797–813.

Nowicki, E., & Smutnicki, C. (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, *8*(2), 145–159.

Pezzella, F., Morgantia, G., & Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, *35*(10), 3202–3212.

Ren, Q., & Wang, Y. P. (2012). A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, *39*(10), 2291–2299.

Storer, R. H., Wu, S. D., & Vaccari, R. (1992). New search spaces for sequencing problems with applications to job-shop scheduling. *Management Science*, *38*(10), 1495–1509.

Taillard, E. D. (1994). Parallel tabu search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, *6*(2), 108–117.

Uzsoy, R., Lee, C. Y., & Louis, A. M. V. (1994). A review of production planning and scheduling models in the semiconductor industry, Part-II: Shop floor control. *IIE Transactions*, *26*(5), 44–55.

van Laarhoven, P. J. M., Aarts, E. H. L., & Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, *40*(1), 113–125.

Yamada, T., & Nakano, R. (1992). A genetic algorithm applicable to large-scale job-shop problems. In *Proceedings of the second international workshop on parallel problem solving from nature* (pp. 281–290).

Yang, J., Sun, L., Lee, H. P., Qian, Y., & Liang, Y. (2008). Clonal selection based memetic algorithm for job shop scheduling problems. *Journal of Bionic Engineering*, *5*(2), 111–119.

Yin, Y., Cheng, S. R., Cheng, T. C. E., Wu, W. H., & Wu, C. C. (2013). Two-agent single-machine scheduling with release times and deadlines. *International Journal of Shipping and Transport Logistics*, *5*(1), 75–94.

Zhang, C. Y., Li, P. G., Rao, Y. Q., & Guan, Z. L. (2007). A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, *34*(11), 3229–3242.

Zhang, C. Y., Li, P. G., Rao, Y. Q., & Guan, Z. L. (2008). A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, *35*(1), 282–294.