Int. J. Production Economics 174 (2016) 93–110

Contents lists available at ScienceDirect

Int. J. Production Economics

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

An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem

Xinyu Li, Liang Gao*

State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

ARTICLE INFO

Article history:
Received 20 April 2012
Accepted 17 December 2015
Available online 4 February 2016

Keywords:
Flexible job shop scheduling
Hybrid algorithm
Makespan
Genetic algorithm
Computational time

ABSTRACT

Flexible job shop scheduling problem (FJSP) which is an extension of the classical job shop scheduling problem is a very important problem in the modern manufacturing system. It allows an operation to be processed by any machine from a given set. It has been proved to be a NP-hard problem. In this paper, an effective hybrid algorithm (HA) which hybridizes the genetic algorithm (GA) and tabu search (TS) has been proposed for the FJSP with the objective to minimize the makespan. The GA which has powerful global searching ability is utilized to perform exploration, and TS which has good local searching ability is applied to perform exploitation. Therefore, the proposed HA has very good searching ability and can balance the intensification and diversification very well. In order to solve the FJSP effectively, effective encoding method, genetic operators and neighborhood structure are used in this method. Six famous benchmark instances (including 201 open problems) of FJSP have been used to evaluate the performance of the proposed HA. Comparisons among proposed HA and other state-of-the-art reported algorithms are also provided to show the effectiveness and efficiency of proposed method. The computational time of proposed HA also has been compared with other algorithms. The experimental results demonstrate that the proposed HA has achieved significant improvement for solving FJSP regardless of the solution accuracy and the computational time. And, the proposed method obtains the new best solutions for several benchmark problems.

1. Introduction

Production scheduling is one of the most important issues in the planning and scheduling of the modern manufacturing systems (Chen et al., 1999). Optimization technology of the production scheduling can introduce significant improvements to the efficiency of manufacturing facilities through eliminating or reducing scheduling conflicts, reducing flow-time and work-in-process, improving production resources utilization and adapting to irregular shop floor disturbances. There are several workshop styles (including job shop scheduling problem, JSP) in the manufacturing system (Wang and Cheng, 2015; Liou and Hsieh, 2015). The classical JSP which is one of the most difficult problems in this field had been proved to be a NP-hard problem (Graey et al., 1976). It is assumed that there is no flexibility of the resources (including machines and tools) for each operation of every job. It may meet the requirements of traditional manufacturing system. However, in modern manufacturing enterprises, many flexible manufacturing systems and Numerical Control (NC) machines are introduced to improve the production efficiency (Seebacher and Winkler, 2014). These systems and machines can process several types of the operations. This means that the assumption about one machine only processing one type of the operation in JSP cannot match this situation. In this case, the flexible job shop scheduling problem (FJSP) attracts more and more attentions from the researchers and engineers (Chaudhry and Khan, 2016).

FJSP which is an extension of the classical JSP allows one operation to be processed by any machine from a given set. The FJSP can be decomposed into two sub-problems: the machine selection problem (MS) and the operations sequencing problem (OS). The JSP only contains the OS problem. Therefore, in the same scale, FJSP is more complicated than JSP. So, it is also a NP-hard problem. After Brucker and Schile (1990) first presented this problem in 1990, many methods have been presented to solve this problem (Chaudhry and Khan, 2016). The current approaches for solving FJSP mainly include exact algorithm (Demir and Isleyen, 2013), dispatching rules (Baykasoglu and Ozbakir, 2010), evolutionary algorithm (EA) (Pezzella et al., 2008; Demir and Isleyen, 2013; Palacios et al., 2015a; Yuan and Xu, 2015), swarm intelligence (SI) based approaches (Huang et al., 2013; Gao et al., 2015), local search (LS) algorithms (Mastrolilli and Gambardella, 2000; Yazdani et al., 2010; Rossi, 2014; Jia and Hu, 2014) and so on. The

* Corresponding author. Tel.: +86 27 87559419; fax: +86 27 87543074.
E-mail address: gaoliang@hust.edu.cn (L. Gao).

http://dx.doi.org/10.1016/j.ijpe.2016.01.016
0925-5273/

research results of FJSP are also used in some real-world applications (Mati et al., 2010; Guevara et al., 2014; Calleja and Pastor, 2014; Driss et al., 2015).

Every algorithm has its own advantages. However, it also lacks some abilities. And no algorithm can solve all types of optimization problems based on the No Free Lunch Theorem (Wolpert and Macready, 1997). Therefore, in this paper, a hybrid algorithm (HA), which combines the global search and local search (LS) by using genetic algorithm (GA) to perform exploration and a guided LS algorithm (tabu search, TS) to perform exploitation, has been proposed for the FJSP. The GA has powerful global searching ability and TS has valuable local searching ability. This method combines the advantages of GA and TS together. Therefore, it has very effective searching ability and can balance the intensification and diversification very well. In order to solve the FJSP effectively, effective encoding method, genetic operators and neighborhood structure are used in this method. Six famous benchmark instances (including 201 open problems) of FJSP are used to evaluate the performance of proposed HA. Through experimental studies, the merits of the proposed approach can be demonstrated clearly.

The remainder of this paper is structured as follows. Literature review is presented in Section 2. Problem formulation is discussed in Section 3. HA-based approach for FJSP is proposed in Section 4. Experimental studies are reported in Section 5. Section 6 describes the conclusions and future works.

2. Literature review

The reported methods for FJSP can be divided into two categories. The first one is the exact method and the other one is the approximation method. The exact method contains mathematical programming (MP) methods. The approximation method contains some dispatch rules (DRs) and artificial intelligence (AI) based approaches.

Brucker and Schile (1990) first presented FJSP and proposed a polynomial graphical algorithm (PGA) to solve a two-job FJSP. Torabi et al. (2005) proposed a mixed integer nonlinear program for the common cycle multi-product lot-scheduling problem in deterministic flexible job shops, which needed to determine the machine allocation, sequencing, lot-sizing and scheduling decisions simultaneously. Gomes et al. (2005) presented a new integer linear programming model for FJSP and used the commercial mixed-integer linear programming (MILP) software to solve this problem. Demir and Isleyen (2013) evaluated several mathematical models of FJSP. Roshanaei et al. (2013) developed two novel MILP models for FJSP. The exact method can obtain exact optimal solution. However, exact algorithms are not effective for solving large scale FJSP instances (the total number of operations is more than 200) (Pezzella et al., 2008). Therefore, most presented methods on FJSP focused on the approximation method including dispatch rules (DRs) and artificial intelligence (AI) based approaches.

Paulli (1995) applied some exist DRs to solve the MS subproblem and several different TS methods to deal with the OS subproblem. Tay and Ho (2008) used DRs discovered through genetic programming to solve the multi-objective FJSP. Baykasoglu and Ozbakir (2010) analyzed the effects of DRs on the scheduling performance of job shops with different flexibility levels. Ziaee (2014) proposed a heuristic based on a construction procedure for solving FJSP. The merits of DRs are simple and can solve large scale problems. However, the results quality of the DRs is not very good. Therefore, many methods on FJSP are the AI-based approaches, such as artificial immune algorithm (Bagheri et al., 2010), filtered beam search (Wang and Yu, 2010; Birgin et al., 2015), discrepancy search (Hmida et al., 2010), harmony search (Yuan and Xu, 2013a; Yuan et al., 2013b; Gao et al., 2014), evolutionary algorithm (EA), swarm intelligence (SI) based approaches, local search (LS) algorithms, hybrid algorithms (HA) and so on.

Evolutionary algorithm (EA) is an effective type of meta-heuristic method, including genetic algorithm (GA), genetic programming, evolution strategies and evolution programming. Jensen (2003) considered the issue of robust and flexible solutions for FJSP. Ho et al. (2007) proposed architecture for learning and evolving of FJSP called LEarnable Genetic Architecture (LEGA). Pezzella et al. (2008) developed a GA which integrated different strategies for generating the initial population, selecting the individuals for reproduction and reproducing new individuals to solve FJSP. Giovanni and Pezzella (2010) proposed an improved GA to solve the distributed FJSP. Zhang et al. (2011) proposed a modified GA for FJSP and obtained good results. Chen et al. (2012) developed an algorithm based on GA and grouping GA for FJSP. Chiang and Lin (2013) proposed a multi-objective EA which utilized effective genetic operators and maintained population diversity carefully for the multi-objective FJSP. Xiong et al. (2013) presented a multi-objective EA for the robust scheduling for FJSP with random machine breakdowns. Demir and Isleyen (2014) developed an effective GA for the FJSP with overlapping in operation. Driss et al. (2015) developed a GA which employed a new chromosome representation and some different strategies for crossover and mutation for FJSP. But, its experimental result of problem MK01 was wrong. The authors also applied the proposed GA on a drug manufacturing company. The above published researches show that the EAs are effective for the scheduling problems because of their powerful global search ability. However, because of lack of neighborhood search procedure, they do not have good local search ability. They can be improved by combining with other local search algorithms.

Swarm intelligence (SI) method is another effective type of meta-heuristic method, including ant colony optimization (ACO) algorithm, particle swarm optimization (PSO) algorithm, and artificial bee colony (ABC) algorithm and so on. ACO is good at solving general and combinatorial optimization problems (Girish and Jawahar, 2009). Rossi and Dini (2007) presented an ACO based software system for solving FJSP in a job shop environment with routing flexibility, sequence-dependent setup and transportation time. Girish and Jawahar (2009) proposed an ACO for FJSP under makespan criterion and used ILOG Solver to evaluate the performance of the proposed algorithms. Huang et al. (2013) developed a two-pheromone ACO algorithm for the FJSP considering due window and sequential dependent setup time of jobs. Rossi (2014) proposed an ACO with reinforced pheromone relationships based on a disjunctive graph model for FJSP considering sequence-dependent setup and transportation times. Gao et al. (2006) developed an effective general PSO algorithm for FJSP. Li et al. (2011a) developed a Pareto-based ABC algorithm for the multi-objective FJSP. Wang et al. (2012) proposed an ABC algorithm for FJSP considering makespan. Gao et al. (2015) proposed a two-stage ABC algorithm for FJSP with new job insertion. SI methods have the similar search characteristics with EAs.

Local search (LS) method includes tabu search (TS), simulated annealing (SA), variable neighborhood search (VNS) and so on. Its effectiveness mainly depends on the design of neighborhood structures. TS is one of the most effective methods for solving the scheduling problem because of its properties. Hurink et al. (1994) applied TS techniques to solve FJSP. Peres and Paulli (1997) developed a TS to solve the FJSP. Mastrolilli and Gambardella (2000) proposed a TS procedure with two effective neighborhood functions to solve FJSP and obtain good results. Bozejko et al. (2010) proposed a parallel double-level meta-heuristic approach for the FJSP based on two methods implemented on the higher level: TS and population based approach. Vilcot and Billaut (2011)

developed a TS for the multi-criteria FJSP considering makespan and maximum lateness. Jia and Hu (2014) proposed a novel path-relinking algorithm based on the TS algorithm with back-jump tracking for the multi-objective FJSP. VNS is one of the renowned meta-heuristics which has been successfully applied to solve several optimization problems. Amiri et al. (2010) proposed a VNS algorithm to solve the FJSP to minimize makespan. Yazdani et al. (2010) developed a parallel VNS algorithm to solve the FJSP for minimizing the makespan. Karimi et al. (2012) proposed a knowledge based VNS algorithm for the FJSP. Based on the above analysis, most LS methods are the single point based methods. This characteristic makes them lack of global search ability. So, they should be combined with other global search algorithms.

Some researchers also tried to hybridize several algorithms together to construct some effective hybrid algorithms (HA) for FJSP. Zribi et al. (2007) proposed a hierarchical method for FJSP. In this method, for the MS sub-problem, they proposed two methods; for the OS sub-problem, they used a hybrid GA to deal with it. Gao et al. (2008) developed a hybrid genetic and variable neighborhood descent algorithm for FJSP. Ho and Tay (2008) combined evolutionary algorithm and guided LS to solve the multi-objective FJSP. Li et al. (2011b) proposed a hybrid TS algorithm with an efficient neighborhood structure for FJSP. Moslehi and Mahnam (2011) presented a Pareto approach based on PSO and local search for multi-objective FJSP. Zhang et al. (2012) hybridized GA and TS for FJSP with transportation constraints and bounded processing times. Yuan and Xu (2013c) combined differential evolution with a local search algorithm based on the critical path for FJSP. Palacios et al. (2015b) developed a hybrid genetic tabu search algorithm for the fuzzy FJSP. Yuan and Xu (2015) proposed the memetic algorithms combining non-dominated sorting genetic algorithm (NSGAII) with a new local search algorithm for multi-objective FJSP.

To sum up, different algorithms have their own advantages and disadvantages. The properties of different algorithms (including MP, DRs, EA, SI, LS and HA) developed for FJSP have been analyzed in Table 1 (Chaudhry and Khan, 2016). The researches should develop effective method based on the properties of different algorithms.

Based on the analysis of the above methods, this research proposes a new HA-based approach for FJSP. Details of proposed HA will be given in the following sections.

3. Problem formulation

The $n \times m$ FJSP can be defined as follows (Gao et al., 2006):

"There is a set of n jobs $J=\{J_1, J_2, J_3,\ldots, J_n\}$ and a set of m machines $M=\{M_1, M_2, M_3,\ldots, M_m\}$. Each job J_i consists of a sequence of operations $\{O_{i1}, O_{i2}, O_{i3}, \ldots, O_{in_i}\}$ where n_i is the number of operations that J_i comprises. Each operation O_{ij} ($i=1, 2, \ldots, n$; $j=1, 2, \ldots, n_i$) has to be processed by one machine out of a set of given machines $M_{ij} \subseteq M$."

The problem is thus to both determine an assignment and a sequence of operations on the machines to satisfy some criteria. It contains of two sub-problems: the machine selection (MS) problem and the operation sequencing (OS) problem. Therefore, the FJSP is more complicated and challenging than the classical JSP because it requires a proper selection of a machine from a set of available machines to process each operation of every job (Ho et al., 2007).

In this paper, the scheduling objective is to minimize the maximal completion time of all the operations, i.e. makespan. The mathematical model of FJSP can refer to Ozguven et al. (2010) and Roshanaei et al. (2013). The hypotheses considered in this paper are summarized as follows:

Table 1
The properties of different algorithms developed for FJSP.

Type	Algorithm	Procedure	Advantage	Disadvantage
MP	PGA	Construct network and calculate the shortest path	Obtain exact solution	Scale of problems is very small
	MILP	Construct the mathematical models and solved by software		
DRs		Generate several priority/dispatching rules	Simple and can solve large scale problems	Not very good solution quality
EA	GA	Inspired by natural evolution and seek solutions by applying crossover and mutation	Good global search ability	Lack of local search ability
	DE	Based on vector differences and suited for numerical optimization problems		
SI	ACO	Inspired by the pheromone trail laying behavior of real ant colonies whereby it mimics ants' social behaviors in finding shortest paths	Good global search ability	Lack of local search ability
	PSO	Inspired by the flocking and schooling patterns of birds		
	ABC	Inspired by the intelligent behavior of honey bees, that is, foraging behavior of honey bee colonies		
LS	TS	Based on the premise that problem solving, to qualify as intelligent, must incorporate adaptive memory and responsive exploration	Good local search ability	Lack of global search ability
	SA	Exploit an analogy to the process in which a metal cools and freezes into a minimum energy crystalline structure and search for a minimum in a more general system		
	VNS	Explore different neighborhoods of the current incumbent solution, and move from there to a new one if an improvement noted		
HA	GA+VNS	Hybridize GA for global searching and VNS for local searching	Good searching ability and balance its intensification and diversification well	Complicated algorithm design and may cost much computation time
	GA+TS	GA solves the assignment problem with transportation, and TS finds and improves the sequence on each resource		
	DE+LS	A LS algorithm based on the critical path is embedded in the DE framework		
	MA	A new LS algorithm is embedded in the NSGAII framework		

(1) Jobs are independent. Job preemption is not permitted and each machine can handle only one job at a time.
(2) The different operations of one job cannot be processed simultaneously.
(3) All jobs and machines are available at time zero simultaneously.
(4) After a job is processed on a machine, it is immediately transported to the next machine on its process, and the transmission time is assumed to be negligible.
(5) Setup time for the operations on the machines is independent of the operation sequence and is incorporated into the processing time.

4. Proposed hybrid algorithm for FJSP

4.1. Workflow of the proposed HA

In this paper, the proposed HA hybridizes GA and a guided LS algorithm (TS) for FJSP. However, there are various methods to design the HAs. In this paper, the TS algorithm is inserted into the procedure of GA for local searching. Workflow of proposed HA is shown in Fig. 1. All the details of the proposed HA will be given separately in the following sub-sections. The overall procedure of the proposed approach is described as follows:

Step 1: Set the parameters of the proposed HA;
Step 2: Initialization: Initialize the population and Set $Gen=1$, *Gen* is the current generation;
Step 3: Evaluation: Evaluate every individual in the population by the objective;
Step 4: Is the termination criteria satisfied?
If yes, go to *Step* 7;
Else, go to *Step* 5;
Step 5: Generate the new population:
Step 5.1: Use the genetic operators (including selection, crossover and mutation operators) to generate the new population;
Step 5.2: Apply the guided LS algorithm (TS) to improve the quality of every individual;
Step 6: Set $Gen=Gen+1$ and go to *Step* 3;
Step 7: Output the best solution.

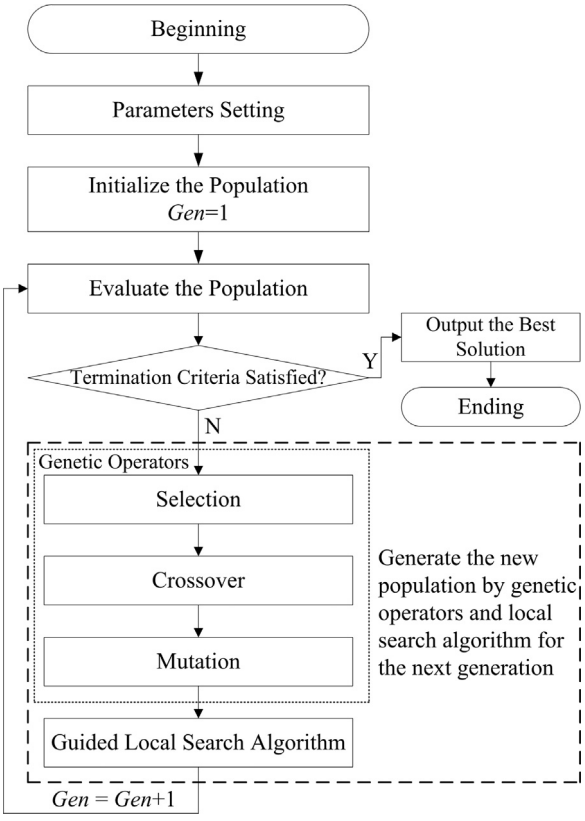

Fig. 1. The workflow of the proposed HA.

4.2. Encoding and decoding

Chromosomes are corresponding to the solutions of the FJSP. Encoding of the individual is very important in the GA. In this paper, the encoding method in Gao et al. (2006) is adopted. Because FJSP contains two sub-problems, the chromosome includes two strings. The first one is the OS string and the second one is the MS string. They have different encoding methods.

The encoding method of OS string is the operation-based representation method, which is comprised of the jobs' numbers. This representation uses an un-partitioned permutation with On_i-repetitions of job numbers (On_i is the total number of operations of job i). In this representation, each job number appears On_i times in the OS string. By scanning the OS string from left to right, the fth appearance of a job number refers to the fth operation of this job. The important feature of this representation is that any permutation of the OS string can be decoded to a feasible solution. Assuming there are n jobs, the length of the OS string is equal to $\sum_{i=1}^{n} On_i$. The initial OS population is generated based on the encoding principle randomly.

The MS string denotes the selected machines for the corresponding operations of each job. The length of this string is also equal to $\sum_{i=1}^{n} On_i$. It contains n parts, and the length of ith part is On_i. The ith part of this string denotes the selected machine set of the corresponding operations of job i. Every gene in this string denotes the selected machine for the fixed operations. The operation number does not change in the whole searching process. Assuming the hth operation of job i can be processed by a machine set$S_{ih}=\{m_{ih1}, m_{ih2}, ..., m_{ihc_{ih}}\}$, the ith part of MS string can be denoted as $\{g_{i1}g_{i2}\cdots g_{ih}\cdots g_{iOn_i}\}$, and g_{ih} is an integer between 1 and c_{ih} and it means that the hth operation of job i is assigned to the g_{ih}th machine $m_{ihg_{ih}}$ in S_{ih}. The initial MS string is

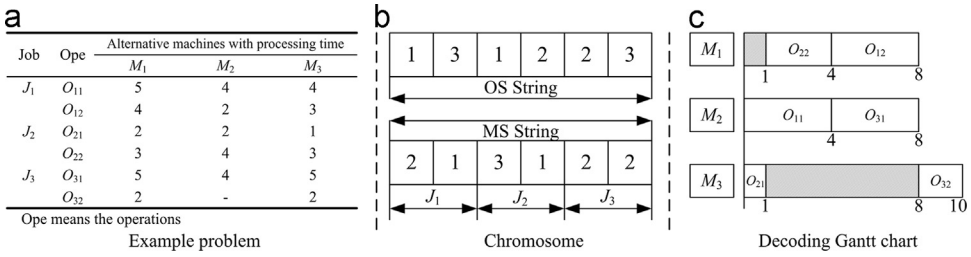

Fig. 2. Encoding and decoding example.

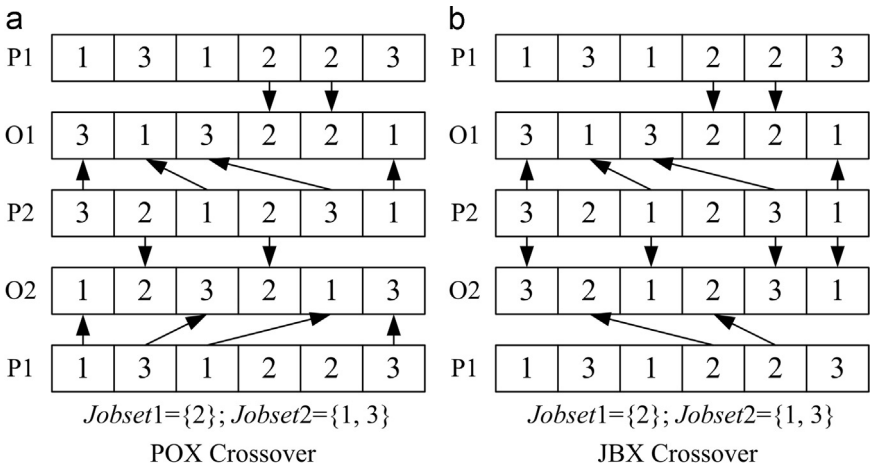

Fig. 3. Crossover operators for OS string.

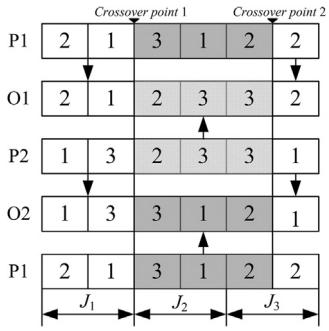

Fig. 4. Crossover operator for MS string.

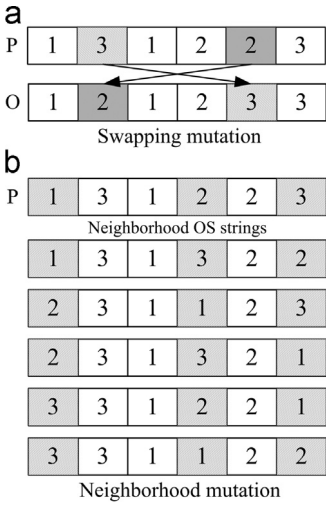

Fig. 5. Mutation operators for OS string.

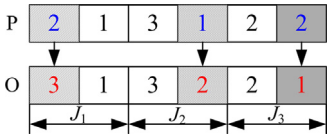

Fig. 6. Mutation operator for MS string.

generated by selecting the alternative machine randomly for each operation of every job. An example is provided in Fig. 2.

A solution is formed by one OS string and one MS string. The solutions can be decoded into semi-active, active, non-delay, and hybrid schedules. Because the makespan is a regular criterion, the active schedule is adopted here. The notations used to explain the decoding procedure are described below:

n	the total number of jobs
m	the total number of machines;
o_{ij}	the jth operation of the ith job;
as_{ij}	the allowable starting time of operation o_{ij};
s_{ij}	the earliest starting time of operation o_{ij};
k	the alternative machine corresponding to o_{ij};
t_{ijk}	the processing time of operation o_{ij} on machine k;
c_{ij}	the earliest completion time of operation o_{ij}, i.e. $c_{ij}=s_{ij}+t_{ijk}$;

The procedure of decoding is as follows:

Step 1: Generate the machine of each operation based on the MS string;
Step 2: Determine the set of operations for every machine: $m_a=\{o_{ij}\}1\le a\le m$;
Step 3: Determine the set of machines for every job: $Jm_d=\{machine\}1\le d\le n$;
Step 4: The allowable starting time for every operation: $as_{ij}=c_{i(j-1)}$ $(o_{ij}\in m_a)$, $c_{i(j-1)}$ is the completion time of the pre-operation of o_{ij} for the same job;
Step 5: Check the idle time of the machine of o_{ij}, and obtain the idle areas $[t\_s, t\_e]$, check these areas in turn (*if:* $\max(as_{ij}, t\_s)+t_{ijk}\le t\_e$, the earliest starting time is $s_{ij}=t\_s$, *else:* check the next area), if there is no area satisfying this condition:$s_{ij}=\max(as_{ij}, c(o_{ij}-1))$, $c(o_{ij}-1)$ is the completion time of the pre-operation of o_{ij} for the same machine;
Step 6: The completion time of every operation: $c_{ij}=s_{ij}+t_{ijk}$;
Step 7: Generate the sets of starting time and completion time for every operation of every job: $T_d(s_{ij}, c_{ij})1\le d\le n$;

Based on the above procedure, it can be obtained that the sets of starting time and completion time for each operation of every job. It is a schedule for the work shop.

Fig. 2 shows an example of encoding and decoding methods. Fig. 2(a) shows the data of this example. It contains 3 jobs and 3 machines (each job contains 2 operations). Fig. 2(b) shows a chromosome. It contains two strings. The OS string is an unpartitioned permutation with repetitions of job numbers. Because there are 3 jobs and each job contains 2 operations, the OS string is the permutation with two 1 s, two 2 s and two 3 s. Its length is 6. And, the MS string includes 3 parts because of 3 jobs. Its length is also equal to 6. Each part denotes the selected machine for the

corresponding operation for each job. For example, the first part denotes the selected machine for the corresponding operation for job 1. The first gene represents the selected machine from S_{11} for O_{11}. The third gene represents the selected machine from S_{21} for O_{21}. Every gene in this string denotes the selected machine for the fixed operations. The operation number does not change in the whole searching process. So, if the parents are feasible, this representation can ensure the offspring to be feasible. The chromosome in Fig. 2(b) (contains one OS string and one MS string) can be decoded to a schedule shown in Fig. 2(c).

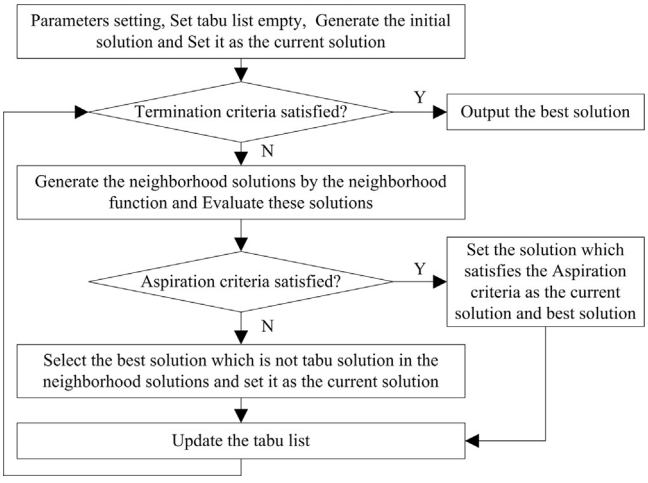

Fig. 7. Workflow of TS.

Table 2
The HA parameters.

Parameters	
The size of the population, *Popsize*	400
The total number of generations, *maxGen*	200
The permitted maximum step size with no improving, *maxStagnantStep*	20
Reproduction probability, p_r	0.005
Crossover probability, p_c	0.8
Mutation probability, p_m	0.1
The maximum iteration size of TS, *maxTSIterSize*	$800 \times (Gen/maxGen)$
Length of tabu list, *maxT*	9

4.3. Genetic operators

It is very important to employ good genetic operators which can effectively deal with the problem and efficiently generate excellent individuals in the population. The genetic operators can generally be divided into three classes: selection, crossover and mutation. In this paper, the chromosome contains two strings: OS string and MS string. Each string has its own genetic operators.

4.3.1. Selection

In GA, the selection operator is used to select the individuals according to the fitness. In this paper, two selection operators are adopted. The first one is the elitist selection scheme. This method reproduces $p_r \times Popsize$ (p_r is the reproduction probability; *Popsize* is the size of the population) individuals with good fitness in parents to the offspring. The other one is the tournament selection scheme. In tournament selection, a number of individuals are selected at randomly (dependent on the tournament size b, typically between 2 and 7, in this paper $b=2$) from the population and the individual with better fitness is selected. The tournament selection approach allows a tradeoff to be made between exploration and exploitation of the gene pool. This scheme can modify the selection pressure by changing the tournament size.

4.3.2. Crossover

In this paper, two crossover operators have been adopted for the OS string. In the procedure of HA, one crossover operator is selected randomly (50%) to crossover the OS string.

The first one is the precedence operation crossover (POX). The basic working procedure of POX is described as follows (two parents are denoted as P1 and P2; two offspring are denoted as O1 and O2):

Step 1: The Job set $J=\{J_1, J_2, J_3,\ldots, J_n\}$ is divided into two groups *Jobset*1 and *Jobset*2 randomly;
Step 2: Any element in P1 which belongs to *Jobset*1 are appended to the same position in O1 and deleted in P1; any element in P2 which belongs to *Jobset*1 are appended to the same position in O2 and deleted in P2;
Step 3: the remaining elements in P2 are appended to the remaining empty positions in O1 seriatim; and the remaining elements in P1 are appended to the remaining empty positions in O2 seriatim.

The second crossover operator for OS string is the job-based crossover (JBX). The basic working procedure of JBX is described as follows (two parents are denoted as P1 and P2; two offspring are denoted as O1 and O2):

Table 3
The experimental results of experiment 1.

Problem	ALCGA	PSOSA	PVNS	AIA[a]		HGVNA[b]		TSPCB[c]		HHS[d]		Heuristic[e]		TABC[f]		HGTS[g]		HA[h]	
$n \times m$				C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)
8×8	15	15	14*	14*	0.76	14*	22.4	14*	4.68	14*	0.00	15	0.094	14*	1.19	14*	2.2	14*	0.00
10×10	7*	7*	7*	7*	8.97	7*	43.1	7*	1.72	7*	0.01	7*	0.218	7*	1.4	7*	7.3	7*	0.01
15×10	14	12	12	11*	109.22	11*	112.2	11*	9.82	11*	0.42	12	0.532	11*	2.97	11*	1.3	11*	0.33

[a] The CPU time on a 2.0 GHz processor with 256 MB of RAM memory in C++.
[b] The CPU time on on a 3.0 GHz Pentium in Delphi.
[c] The CPU time on a Pentium IV 1.6 GHz processor with 512 MB of RAM memory in C++.
[d] The CPU time on an Intel 2.83 GHz Xeon processor with 15.9 GB of RAM memory in Java.
[e] The CPU time on a Pentium IV, 2.2 GHz processor with 2.0 GB of RAM memory in C.
[f] The CPU time on an Intel 2.0 GHz Core (TM) 2 Duo processor with 4.0 GB of RAM memory in C++.
[g] The CPU time on a Xeon E5520 processor with 24 GB of RAM memory in C++.
[h] The CPU time on an Intel 2.0 GHz Core (TM) 2 Duo processor with 8.0 GB of RAM memory in C++.

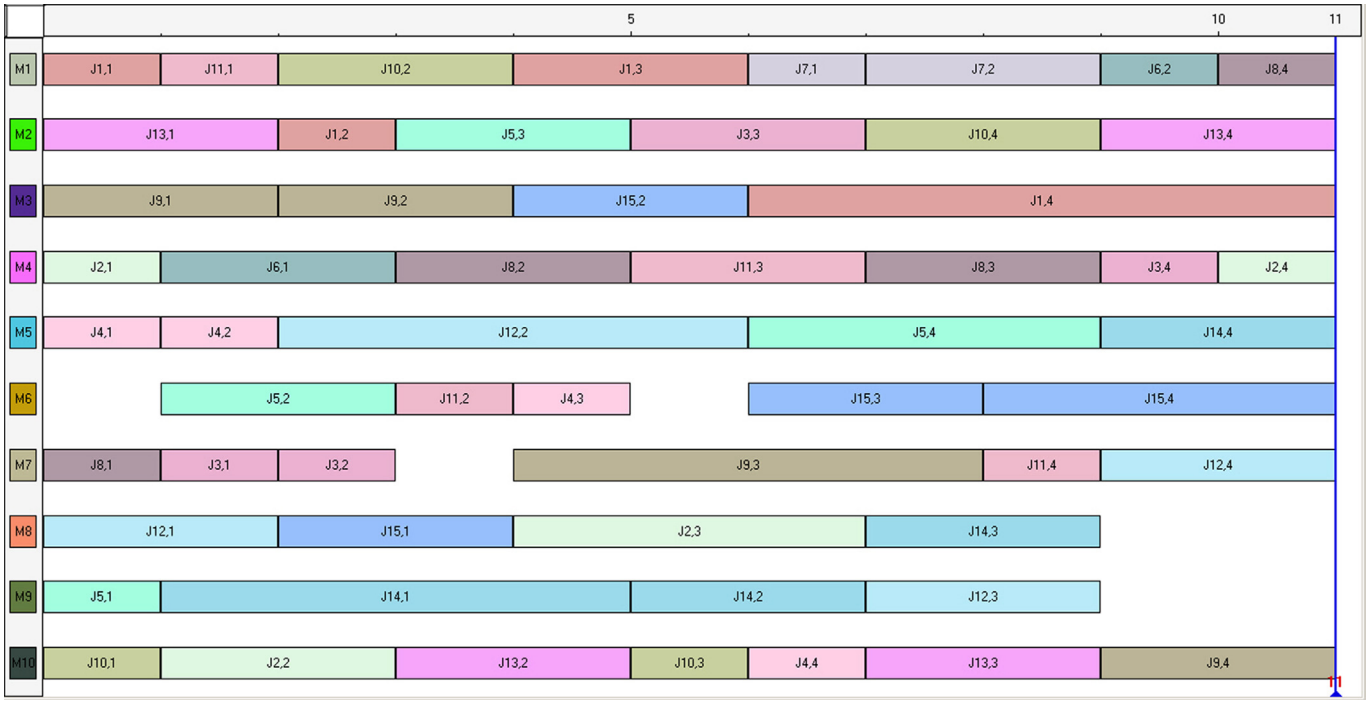

Fig. 8. Gantt chart of problem 15×10 in experiment 1.

Table 4
The experimental results of experiment 2.

Problem	$n \times m$	AIA[a]		HHS[b]		M2[c]		MILP[d]		HA[e]	
		C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)
SFJS01	2×2	66*	0.03	66*	0.00	66*	0.03	66*	0.00	66*	0.00
SFJS02	2×2	107*	0.03	107*	0.00	107*	0.10	107*	0.01	107*	0.00
SFJS03	3×2	221*	0.04	221*	0.00	221*	0.05	221*	0.05	221*	0.00
SFJS04	3×2	355*	0.04	355*	0.00	355*	0.04	355*	0.02	355*	0.00
SFJS05	3×2	119*	0.04	119*	0.00	119*	0.06	119*	0.04	119*	0.00
SFJS06	3×3	320*	0.04	320*	0.00	320*	0.28	320*	0.01	320*	0.00
SFJS07	3×5	397*	0.04	397*	0.00	397*	0.03	397*	0.00	397*	0.00
SFJS08	3×4	253*	0.05	253*	0.00	253*	0.16	253*	0.04	253*	0.00
SFJS09	3×3	210*	0.05	210*	0.00	210*	1.26	210*	0.01	210*	0.00
SFJS10	4×5	516*	0.05	516*	0.00	516*	0.06	516*	0.02	516*	0.00
MFJS01	5×6	468*	9.23	468*	0.01	468*	0.78	468*	0.26	468*	0.00
MFJS02	5×7	448*	9.35	446*	0.01	446*	49	446*	0.87	446*	0.00
MFJS03	6×7	468	10.06	466*	0.12	466*	191	466*	1.66	466*	0.02
MFJS04	7×7	554*	10.54	554*	0.06	564	1051	554*	27.43	554*	0.02
MFJS05	7×7	527	10.61	514*	0.02	514*	225	514*	4.55	514*	0.02
MFJS06	8×7	635	22.18	634*	0.01	634*	231	634*	52.48	634*	0.01
MFJS07	8×7	879*	24.82	879*	0.11	928	3600	879*	1890	879*	0.08
MFJS08	9×8	884*	26.94	884*	0.08	N/A	N/A	N/A	N/A	884*	0.06
MFJS09	11×8	1088	30.76	1055*	0.94	N/A	N/A	N/A	N/A	1055*	0.48
MFJS10	12×8	1267	30.94	1196*	0.69	N/A	N/A	N/A	N/A	1196*	0.59

N/A means the result was not given by the author.

[a] The CPU time on a 2.0 GHz processor with 256 MB of RAM memory in C++.
[b] The CPU time on an Intel 2.83 GHz Xeon processor with 15.9 GB of RAM memory in Java.
[c] The CPU time on a Core(TM) 2 Quard CPU 2.66 GHz processor with 4.0 GB of RAM memory in mathematical language GAMS and used CPLEX (for linear models) and SNOPT (for nonlinear models) solvers.
[d] The CPU time on an Intel Xeon E5440 2.83 GHz processor with 2.0 GB of RAM memory and used IBMILOG CPLEX 12.1 solver.
[e] The CPU time on an Intel 2.0 GHz Core (TM) 2 Duo processor with 8.0 GB of RAM memory in C++.

Step 1: The Job set $J=\{J_1, J_2, J_3,\ldots, J_n\}$ is divided into two groups *Jobset*1 and *Jobset*2 randomly;
Step 2: Any element in P1 which belongs to *Jobset*1 are appended to the same position in O1 and deleted in P1; any element in P2 which belongs to *Jobset*2 are appended to the same position in O2 and deleted in P2;
Step 3: The remaining elements in P2 are appended to the remaining empty positions in O1 seriatim; and the remaining elements in P1 are appended to the remaining empty positions in O2 seriatim.

For the OS string, Fig. 3(a) shows an example of the POX crossover operator and the Fig. 3(b) describes an example of the JBX crossover operator.

For the MS string, a two-point crossover has been adopted here as the crossover operator. In this operator, two positions are selected

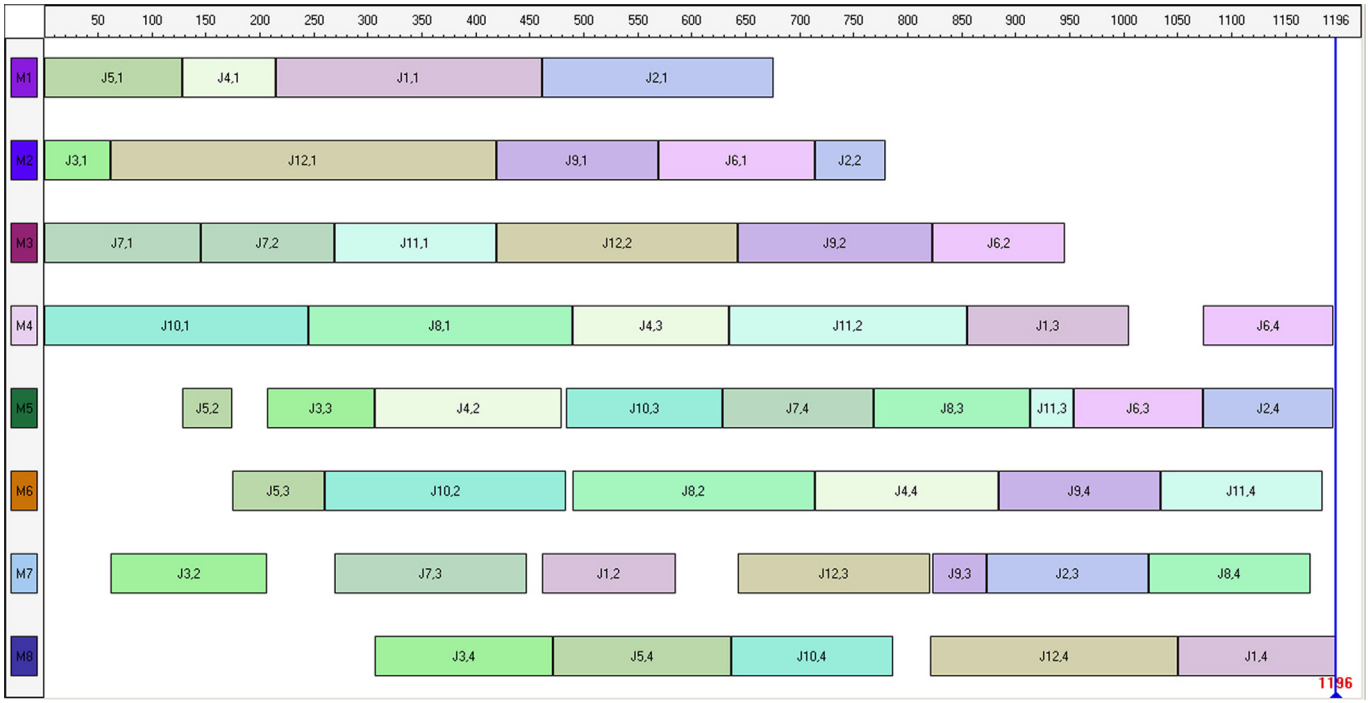

Fig. 9. Gantt chart of problem MFJS10 in experiment 2.

Table 5
The experimental results (minimizing makespan) of experiment 3.

Problem	$n \times m$	TS	LEGA	HO	GA	MAS	HGVNA	AIA	VNS	PVNS	HTS	DS	HAT	ABC	EA	HHS	HS	Heuristic	TABC	HGTS	MA2	HA
MK01	10 × 6	40*	40*	41	40*	40*	40*	40*	40*	40*	40*	40*	40*	40*	40*	40*	40*	42	40*	40*	40*	40*
MK02	10 × 6	26*	29	28	26*	32	26*	26*	26*	26*	26*	26*	26*	26*	26*	26*	26*	28	26*	26*	26*	26*
MK03	15 × 8	204*	N/A	204*	204*	N/A	204*	204*	204*	204*	204*	204*	204*	204*	204*	204*	204*	204*	204*	204*	204*	204*
MK04	15 × 8	60*	67	67	60*	67	60*	60*	60*	60*	62	60*	60*	60*	61	60*	60*	75	60*	60*	60*	60*
MK05	15 × 4	173	176	177	173	188	172*	173	173	173	172*	173	173	172*	173	172*	172*	179	173	172*	172*	172*
MK06	10 × 15	58	67	61	63	85	58	63	59	60	65	58	60	60	65	59	58	69	60	57*	59	57*
MK07	20 × 5	144	147	154	139*	154	139*	140	140	141	140	139*	140	139*	140	139*	139*	149	139*	139*	139*	139*
MK08	20 × 10	523*	523*	523*	523*	523*	523*	523*	523*	523*	523*	523*	523*	523*	523*	523*	523*	555	523*	523*	523*	523*
MK09	20 × 10	307*	320	321	311	N/A	307*	312	307*	307*	310	307*	307*	307*	311	307*	307*	342	307*	307*	307*	307*
MK10	20 × 15	198	229	219	212	N/A	197*	214	207	208	214	197*	205	208	225	202	205	242	202	198	202	197*

N/A means the result was not given by the author.

Table 6
The experimental results (computational time in terms of seconds) of experiment 3.

Problem	$n \times m$	HGVNA[a]	AIA[b]	HTS[c]	ABC[d]	HHS[e]	HS[e]	Heuristic[f]	TABC[g]	HGTS[h]	MA2[i]	HA[j]
MK01	10 × 6	1.47	97.21	2.80	3.23	3.87	0.07	0.09	3.36	5	20.16	0.06
MK02	10 × 6	3.44	103.46	19.31	35.63	5.79	0.74	0.17	3.72	15	28.21	0.59
MK03	15 × 8	17.47	247.37	0.98	1.19	36.60	0.01	0.52	1.56	2	53.76	0.16
MK04	15 × 8	3.42	152.07	40.82	38.94	13.30	1.04	0.20	66.58	10	30.53	0.49
MK05	15 × 4	6.82	171.95	20.23	19.29	35.78	7.47	0.20	78.45	18	36.36	4.57
MK06	10 × 15	5.30	245.62	27.18	66.61	111.65	60.73	0.45	173.98	63	80.61	53.82
MK07	20 × 5	6.23	161.92	35.29	131.84	26.16	10.59	0.39	66.19	33	37.74	20.01
MK08	20 × 10	8.44	392.25	4.65	2.33	171.10	0.02	0.66	2.15	3	77.71	0.02
MK09	20 × 10	18.71	389.71	70.38	91.21	172.24	0.39	0.94	304.43	24	75.23	0.86
MK10	20 × 15	19.36	384.54	89.83	237.11	437.69	373.01	1.20	418.19	104	90.75	33.21

[a] The CPU time on on a 3.0 GHz Pentium in Delphi.
[b] The CPU time on a 2.0 GHz processor with 256 MB of RAM memory in C++.
[c] The CPU time on a Pentium IV 1.6 GHz processor with 512 MB of RAM memory in C++.
[d] The CPU time on a 2.83 GHz processor with 3.21 GB of RAM memory in C++.
[e] The CPU time on an Intel 2.83 GHz Xeon processor with 15.9 GB of RAM memory in Java.
[f] The CPU time on a Pentium IV, 2.2 GHz processor with 2.0 GB of RAM memory in C.
[g] The CPU time on an Intel 2.4 GHz Core (TM) 2 Duo processor with 4.0 GB of RAM memory in C++.
[h] The CPU time on a Xeon E5520 processor with 24 GB of RAM memory in C++.
[i] The CPU time on an Intel Core i7-3520M 2.9 GHz processor with 8.0 GB of RAM memory in Java.
[j] The CPU time on an Intel 2.0 GHz Core (TM) 2 Duo processor with 8.0 GB of RAM memory in C++.

randomly at first. Then two offspring strings are generated by swapping all elements between the positions of the two parents' strings. Because every gene in this string denotes the selected machine for the fixed operations, the operation number does not change in the whole searching process. This crossover can ensure to generate feasible offspring if the selected parents are feasible. Fig. 4 shows an example of the crossover operator for MS string. Firstly, selecting two feasible parents (P1 and P2); then, selecting two positions randomly, in this example, the 2nd and 5th positions are selected; adopting the elements of P1 before the 2nd position and after the 5th position to the same positions in O1, and adopting the elements of P2 from the 3rd position to the 4th position to the same

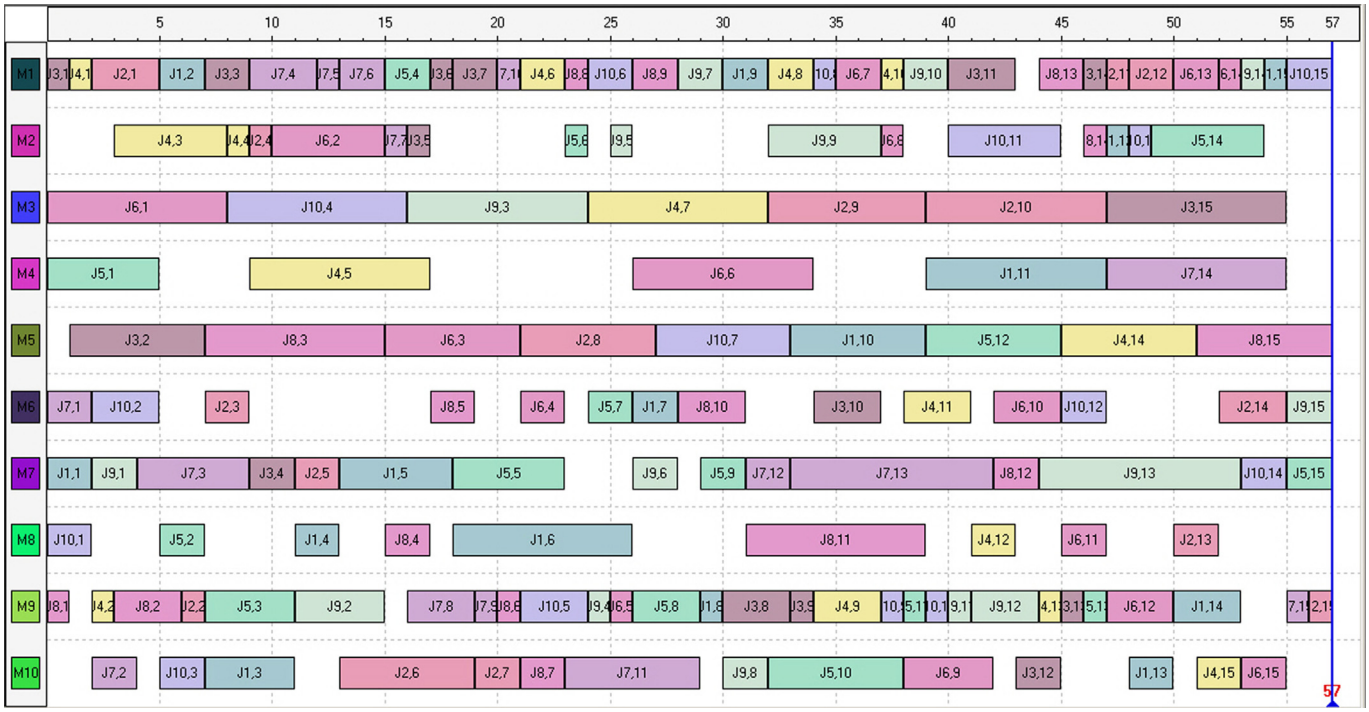

Fig. 10. Gantt chart of problem MK06 in experiment 3.

Table 7
The experimental results of experiment 4.

Problem	$n \times m$	TS	DS	TSBM[2]h	HGVNA[a]		HDEN$_2$[b]		GA[c]		HGTS[d]		HA[e]	
					C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)
mt10c1	10 × 11	928	928	927*	927*	12.87	927*	174.19	928	268.3	927*	13	927*	11.86
mt10cc	10 × 12	910	910	908*	910	12.24	908*	165.61	910	336.7	908*	13	908*	9.70
mt10x	10 × 11	918*	918*	922	918*	12.69	918*	179.22	918*	403	918*	15	918*	10.94
mt10xx	10 × 12	918*	918*	918*	918*	11.70	918*	179.84	918*	279	918*	12	918*	10.58
mt10xxx	10 × 13	918*	918*	918*	918*	11.52	918*	179.39	918*	352.4	918*	12	918*	10.79
mt10xy	10 × 12	906	906	905*	905*	12.24	905*	169.77	905*	310.8	905*	13	905*	11.17
mt10xyz	10 × 13	847*	849	849	849	10.71	847*	160.24	847*	401.4	847*	18	847*	9.44
setb4c9	15 × 11	919	919	914*	914*	45.99	914*	313.02	914*	358	914*	16	914*	15.45
setb4cc	15 × 12	909	909	907*	914	38.79	907*	316.89	909	348.8	907*	15	907*	14.87
setb4x	15 × 11	925*	925*	925*	925*	43.20	925*	338.30	925*	474.7	925*	15	925*	12.89
setb4xx	15 × 12	925*	925*	925*	925*	42.57	925*	336.24	925*	315.3	925*	14	925*	5.40
setb4xxx	15 × 13	925*	925*	925*	925*	36.09	925*	353.55	925*	420.4	925*	15	925*	8.90
setb4xy	15 × 12	916	916	910*	916	41.49	910*	330.18	916	335.2	910*	19	910*	12.21
setb4xyz	15 × 13	905	905	903*	905	39.15	903*	314.64	905	326.6	905	15	905	14.39
seti5c12	15 × 16	1174	1174	1174	1175	72.27	1171	1141.43	1174	297.4	1170*	41	1170*	31.02
seti5cc	15 × 17	1136*	1136*	1136*	1138	63.00	1136*	1222.53	1136*	510	1136*	34	1136*	16.82
seti5x	15 × 16	1201	1201	1198*	1204	66.78	1200	1112.77	1201	356.5	1199	38	1198*	27.32
seti5xx	15 × 17	1199	1199	1197*	1202	63.72	1197*	1078.60	1199	444.4	1197*	34	1197*	29.03
seti5xxx	15 × 18	1197*	1197*	1197*	1204	63.36	1197*	1087.12	1197*	396.7	1197*	31	1197*	19.28
seti5xy	15 × 17	1136*	1136*	1136*	1136*	63.18	1136*	1250.62	1136*	434.4	1136*	34	1136*	17.43
seti5xyz	15 × 18	1125*	1125*	1128	1126	57.96	1125*	1244.22	1125*	458.3	1125*	43	1125*	33.23

[a] The CPU time on a 2.0 GHz processor with 256 MB of RAM memory in C++.
[b] The CPU time on an Intel 2.83 GHz Xeon processor with 15.9 GB of RAM memory in Java.
[c] The CPU time on a Core(TM) 2 Quard CPU 2.66 GHz processor with 4.0 GB of RAM memory in C#.
[d] The CPU time on a Xeon E5520 processor with 24 GB of RAM memory in C++.
[e] The CPU time on an Intel 2.0 GHz Core (TM) 2 Duo processor with 4.0 GB of RAM memory in C++.

positions in O1, using the same process to generate the O2. Base on the whole process, we can see that if two parents are feasible, the offspring will be feasible.

4.3.3. Mutation

In this paper, two mutation operators have been adopted for the OS string. In the procedure of HA, one mutation operator is selected randomly (50%) to mutate the OS string.

The first one is the swapping mutation. The basic working procedure of swapping mutation is described as follows (one parent is denoted as P; one offspring is denoted as O):

Step 1: Select two positions in the P;
Step 2: Swap the elements in the selected positions to generate the O;

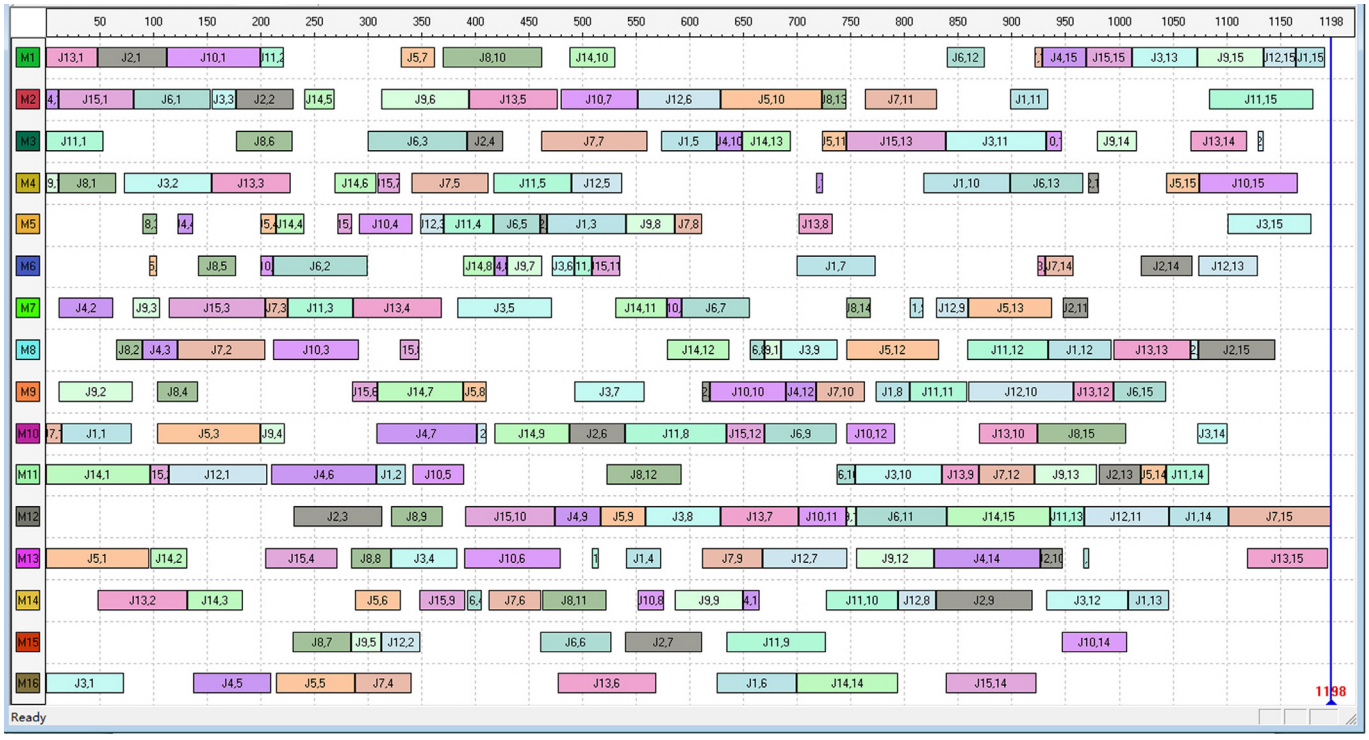

Fig. 11. Gantt chart of problem seti5x in experiment 4.

Table 8
The experimental results of experiment 5.

Problem	$n \times m$	IATS	TS	SA	GPSO	HO	DS	HGVNA[a]		HHS/LNS[b]		HGTS[c]		MA2[d]		HA[e]	
								C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)	C_{max}	CPU(s)
01a	10 × 5	2530	2518	2576	2539	2650	2518	2518	102.71	2505*	837.6	2505*	122	2521	185.72	2505*	107.68
02a	10 × 5	2244	2231	2259	2244	2323	2231	2231	140.98	2230*	972.6	2230*	205	2244	166.01	2230*	132.59
03a	10 × 5	2235	2229	2241	2232	2287	2229	2229	106.53	2228*	1164.6	2228*	181	2234	157.96	2229	97.31
04a	10 × 5	2565	2503*	2555	2523	2646	2503*	2515	95.93	2506	849.6	2503*	112	2513	87.25	2503*	87.00
05a	10 × 5	2229	2216	2258	2234	2353	2216	2217	143.95	2212	931.2	2214	208	2211*	117.03	2212	116.17
06a	10 × 5	2216	2203	2222	2218	2260	2196	2196	111.83	2187	1167.0	2193	260	2172*	135.07	2197	92.88
07a	15 × 8	2408	2283	2396	2361	2543	2283	2307	356.32	2288	1547.4	2270*	344	2365	215.18	2279	203.74
08a	15 × 8	2093	2069	2083	2086	2183	2069	2073	330.08	2067*	1905.6	2070	318	2087	164.33	2067*	184.44
09a	15 × 8	2074	2066	2098	2073	2118	2066	2066	327.49	2069	943.2	2067	376	2075	153.70	2065*	201.26
10a	15 × 8	2362	2291	2397	2362	2465	2291	2315	345.19	2297	1590	2247*	369	2327	180.87	2287	238.09
11a	15 × 8	2078	2063	2092	2083	2124	2063	2071	360.45	2061	1826.4	2064	294	2057*	163.37	2060	180.64
12a	15 × 8	2047	2034	2077	2050	2120	2031	2030	329.71	2027	914.4	2027	486	1992*	165.14	2027	150.81
13a	20 × 10	2302	2260	2343	2342	2492	2257	2257	462.85	2263	2900.3	2250	416	2311	196.45	2248*	292.93
14a	20 × 10	2183	2167	2188	2174	2319	2167	2167	587.13	2164*	3237.5	2170	396	2187	153.67	2167	210.17
15a	20 × 10	2171	2167	2179	2173	2259	2165	2165	669.92	2163*	2112.3	2168	523	2180	148.75	2163*	192.09
16a	20 × 10	2301	2255	2349	2324	2608	2256	2256	452.41	2259	2802.2	2246*	384	2293	194.71	2249	159.63
17a	20 × 10	2169	2141	2170	2162	2296	2140	2140	616.34	2137	3096.4	2142	483	2119*	203.10	2140	202.50
18a	20 × 10	2139	2137	2159	2157	2238	2127	2127	667.01	2124	2489.2	2129	650	2077*	191.23	2132	132.64

[a] The CPU time on a 2.0 GHz processor with 256 MB of RAM memory in C++.
[b] The CPU time on an Intel 2.83 GHz Xeon processor with 15.9 GB of RAM memory in Java.
[c] The CPU time on a Xeon E5520 processor with 24 GB of RAM memory in C++.
[d] The CPU time on an Intel Core i7-3520M 2.9 GHz processor with 8.0 GB of RAM memory in Java.
[e] The CPU time on an Intel 2.0 GHz Core (TM) 2 Duo processor with 8.0 GB of RAM memory in C++.

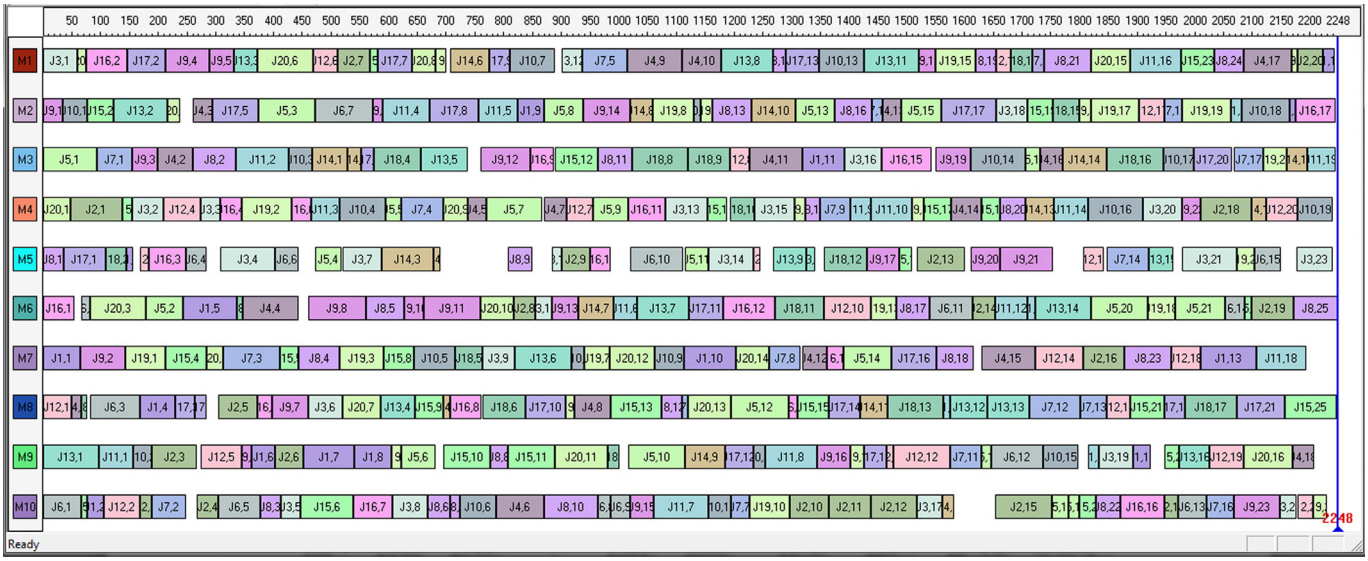

Fig. 12. Gantt chart of problem 13a in experiment 5.

Table 9
The experimental results of edata in experiment 6.

Problem	$n \times m$	TSN1	TSN2	IATS	TS	HA	HA CPU(s)[a]
mt06	6×6	57	57	55*	55*	55*	0.01
mt10	10×10	917	899	878	871*	871*	1.42
mt20	20×5	1109	1135	1106	1088*	1088*	7.38
la01	10×5	611	618	609*	609*	609*	0.22
la02	10×5	655*	656	655*	655*	655*	0.25
la03	10×5	573	566	554	550*	550*	0.92
la04	10×5	578	578	568*	568*	568*	0.89
la05	10×5	503*	503*	503*	503*	503*	0.19
la06	15×5	833*	833*	833*	833*	833*	0.27
la07	15×5	765	778	765	762	762*	4.98
la08	15×5	845*	845*	845*	845*	845*	0.34
la09	15×5	878*	878*	878*	878*	878*	0.33
la10	15×5	866*	866*	866*	866*	866*	0.35
la11	20×5	1106	1106	1103*	1103*	1103*	2.31
la12	20×5	960*	960*	960*	960*	960*	0.76
la13	20×5	1053*	1053*	1053*	1053*	1053*	0.39
la14	20×5	1151	1123*	1123*	1123*	1123*	1.01
la15	20×5	1111*	1121	1111*	1111*	1111*	1.23
la16	10×10	924	961	915	892*	892*	0.87
la17	10×10	757	757	707*	707*	707*	0.45
la18	10×10	864	864	843	842*	842*	1.65
la19	10×10	850	813	796*	796*	796*	2.28
la20	10×10	919	919	864	857*	857*	0.44
la21	15×10	1066	1085	1046	1017	1014*	29.41
la22	15×10	919	905	890	882	880*	25.44
la23	15×10	980	980	953	950*	950*	13.23
la24	15×10	952	952	918	909*	909*	17.35
la25	15×10	970	969	955	941*	941*	25.19
la26	20×10	1169	1149	1138	1125	1123*	31.82
la27	20×10	1230	1236	1215	1186	1184*	24.62
la28	20×10	1204	1197	1169	1149	1147*	39.28
la29	20×10	1210	1205	1157	1118	1115*	44.85
la30	20×10	1253	1286	1225	1204*	1204*	55.86
la31	30×10	1596	1593	1556	1539*	1541	41.35
la32	30×10	1769	1757	1698*	1698*	1698*	55.24
la33	30×10	1575	1575	1547	1547*	1547*	48.69
la34	30×10	1627	1636	1623	1599*	1599*	61.47
la35	30×10	1736	1736	1736	1736*	1736*	10.19
la36	15×15	1247	1235	1171	1162	1160*	61.78
la37	15×15	1453	1456	1418	1397*	1397*	20.61
la38	15×15	1185	1185	1172	1144	1143*	45.97
la39	15×15	1226	1226	1207	1184*	1184*	37.61
la40	15×15	1214	1236	1150	1150	1146*	43.65

[a] The CPU time on an Intel 2.0 GHz Core (TM) 2 Duo processor with 8.0 GB of RAM memory in C++.

The second mutation operator for OS string is the neighborhood mutation method. The basic working procedure of this method is described as follows (one parent is denoted as P; one offspring is denoted as O):

Step 1: Select 3 elements in the P (the values of these elements are different), and generate all the neighborhood OS strings;
Step 2: Choose the one in the neighborhood OS strings randomly and set it as the current OS string, it is the O.

For the OS string, Fig. 5(a) shows an example of the swapping mutation operator and the Fig. 5(b) describes an example of the neighborhood mutation operator. In Fig. 5(a), the 2nd and 5th positions are selected. The offspring is generated by swapping the elements in these positions. In Fig. 5(b), the 1st, 4th and 6th positions are selected. And then, generating 5 neighborhood OS strings, one of them is selected randomly as the offspring.

For the MS string, a mutation operator is designed as follows:

Step 1: Select r positions in the parent (r is the half of the length of the MS string);
Step 2: For each position (according to one operation), change the value of this selected position to the other machine in the machine set of the corresponding operation.

Fig. 6 shows an example of the mutation operator for MS string. In this example, r equals to 3 (the length is 6). And the 1st, 4th and 6th positions are selected. They represent the selected machine for O_{11}, O_{21} and O_{32} respectively. The offspring is generated by changing the value of these positions to the other machine in the machine set of the corresponding operation.

4.4. Local search by tabu search

Tabu search (Glover and Laguna, 1997) is a meta-heuristic method, which has been successfully applied in various combinatorial optimization problems including several scheduling problems. TS allows the searching process to explore solutions which do not decrease the objective function value if these solutions are not forbidden. It is usually obtained by keeping track of the final solution in terms of the action used to transform one solution to the next. It consists of several elements which contain the

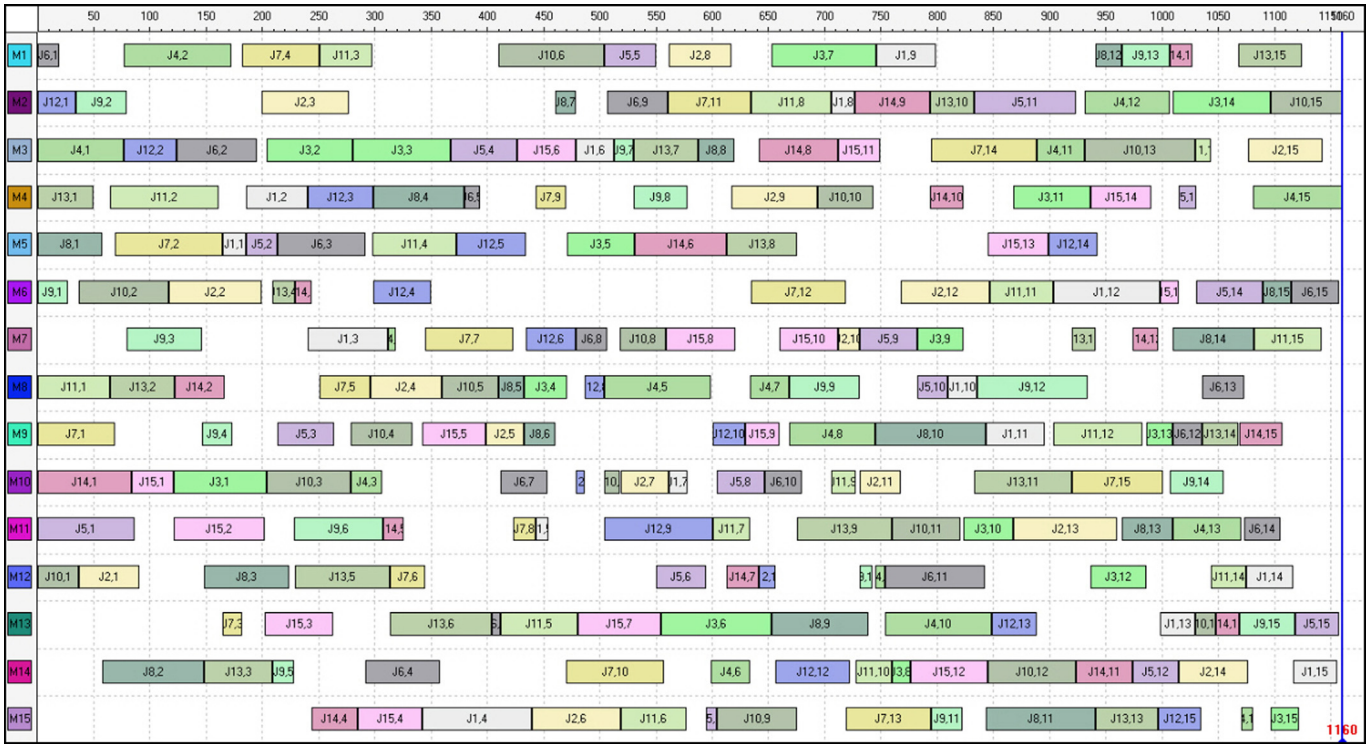

Fig. 13. Gantt chart of problem la36 of edata in experiment 6.

neighborhood structure, the move attributes, the tabu list, aspiration criteria and terminate criteria.

TS has emerged as one of the most efficient LS strategies for scheduling problems. In this study, it has been adopted as LS strategy in the proposed HA for FJSP. And the neighborhood structure proposed by Mastrolilli and Gambardella (2000) has been adopted here. This neighborhood structure was proved to be optimum connected (Mastrolilli and Gambardella, 2000). The basic workflow of TS is shown in Fig. 7. The overall procedure of the TS is described as follows:

Step 1: Set the parameters of TS, Set tabu list empty; Generate the initial solution and Set it as the current solution;
Step 2: Is the termination criteria satisfied?
If yes, go to *Step* 8;
Else, go to *Step* 3;
Step 3: Generate the new neighborhood solutions by the neighborhood function, and Evaluate these solutions;
Step 4: Is the aspiration criteria satisfied?
If yes, go to *Step* 6;
Else, go to *Step* 5;
Step 5: Select the best solution which is not tabu solution in the neighborhood solutions and set it as the current solution, and go to *Step* 7;
Step 6: Set the solution which satisfies the aspiration criteria as the current solution and best solution, and go to *Step* 7;
Step 7: Update the tabu list and go to *Step* 2;
Step 8: Output the best solution.

In the proposed HA, when an individual is to perform LS, it should be converted to a feasible schedule solution at first. And then this solution is used as the initial solution of TS. After the local search, the output solution of TS should be encoded to a feasible chromosome for genetic operators.

4.5. Terminate criteria

In this paper, the HA terminates when the number of generations reaches to the maximum value (*maxGen*) or the permitted maximum step size with no improving (*maxStagnantStep*); TS terminates when the number of iterations reaches to the maximum size (*maxTSIterSize*).

5. Experimental studies and discussions

5.1. Experimental results

The proposed HA procedure was coded in C++ and implemented on a computer with a 2.0 GHz Core (TM) 2 Duo processor with 8.0 GB of RAM memory. To illustrate the effectiveness and performance of the proposed HA, six famous benchmark instances from other papers are adopted here. These instances include 201 open problems for FJSP. Many researchers used them to evaluate their approaches. The objective of this paper is to minimize makespan. The comparisons among proposed HA and other state-of-the-art reported algorithms are also provided to show the performance of proposed method. To illustrate high efficiency of the proposed HA approach, the computational time (s) with some algorithms on several instances is also compared. Because the different computing hardware, programming platforms and coding skills used in each algorithm make these comparisons notoriously problematic (Yuan et al., 2013b), we also attach the CPU with RAM memory, the programming language, and the original computational time for the corresponding algorithms. The HA parameters for these instances are shown in Table 2.

In the proposed algorithm, the HA terminates when the number of generations reaches to the maximum value (*maxGen*) or the permitted maximum step size with no improving (*maxStagnantStep*); TS terminates when the number of iterations reaches to the

Table 10
The experimental results of rdata in experiment 6.

Problem	$n \times m$	TSN1	TSN2	IATS	TS	PBM^{2}h	TSM^{2}h	HA	HA CPU(s)[a]
mt06	6 × 6	47*	47*	47*	47*	N/A	N/A	47*	0.01
mt10	10 × 10	737	737	686*	686*	N/A	N/A	686*	2.42
mt20	20 × 5	1028	1028	1024	1022*	N/A	N/A	1022*	12.79
la01	10 × 5	577	577	574	571	572	574	570*	13.31
la02	10 × 5	535	535	532	530*	530*	532	530*	3.84
la03	10 × 5	481	486	479	478	478	482	477*	4.85
la04	10 × 5	509	506	504	502*	502*	509	502*	4.13
la05	10 × 5	460	458	458	457*	458	462	457*	4.23
la06	15 × 5	801	803	800	799*	799*	801	799*	3.54
la07	15 × 5	752	752	750	750	750	751	749*	7.35
la08	15 × 5	767	768	767	765*	765*	767	765*	9.89
la09	15 × 5	859	857	854	853*	853*	856	853*	11.93
la10	15 × 5	806	805	805	804*	805	807	804*	8.88
la11	20 × 5	1073	1073	1072	1071*	1071*	1072	1071*	2.73
la12	20 × 5	937	937	936*	936*	936*	937	936*	2.12
la13	20 × 5	1039	1039	1038*	1038*	1038*	1039	1038*	3.15
la14	20 × 5	1071	1071	1070*	1070*	1070*	1071	1070*	2.67
la15	20 × 5	1093	1093	1090*	1090*	1090*	1090*	1090*	9.73
la16	10 × 10	717*	717*	717*	717*	N/A	N/A	717*	0.98
la17	10 × 10	646*	646*	646*	646*	N/A	N/A	646*	0.58
la18	10 × 10	674	673	669	666*	N/A	N/A	666*	1.12
la19	10 × 10	725	709	703	700*	N/A	N/A	700*	1.67
la20	10 × 10	756*	756*	756*	756*	N/A	N/A	756*	0.59
la21	15 × 10	861	861	846	835*	N/A	N/A	835*	13.88
la22	15 × 10	790	795	772	760*	N/A	N/A	760*	10.31
la23	15 × 10	884	887	853	842	N/A	N/A	840*	25.04
la24	15 × 10	825	830	820	808	N/A	N/A	806*	22.32
la25	15 × 10	823	821	802	791	N/A	N/A	789*	25.09
la26	20 × 10	1086	1087	1070	1061*	N/A	N/A	1061*	38.35
la27	20 × 10	1109	1115	1100	1091	N/A	N/A	1089*	56.81
la28	20 × 10	1097	1090	1085	1080	N/A	N/A	1079*	55.02
la29	20 × 10	1016	1017	1004	998	N/A	N/A	997*	52.49
la30	20 × 10	1105	1108	1089	1078*	N/A	N/A	1078*	52.88
la31	30 × 10	1532	1533	1528	1521*	N/A	N/A	1521*	50.89
la32	30 × 10	1668	1668	1660	1659*	N/A	N/A	1659*	43.56
la33	30 × 10	1511	1507	1501	1499*	N/A	N/A	1499*	50.90
la34	30 × 10	1542	1543	1539	1536*	N/A	N/A	1536*	38.98
la35	30 × 10	1559	1559	1555	1550*	N/A	N/A	1550*	45.19
la36	15 × 15	1054	1071	1030	1030	N/A	N/A	1028*	56.64
la37	15 × 15	1122	1132	1082	1077	N/A	N/A	1074*	55.47
la38	15 × 15	1004	1001	989	962	N/A	N/A	960*	43.71
la39	15 × 15	1041	1068	1024*	1024*	N/A	N/A	1024*	9.28
la40	15 × 15	1009	1009	980	970*	N/A	N/A	970*	43.40

N/A means the result was not given by the author.

[a] The CPU time on an Intel 2.0 GHz Core (TM) 2 Duo processor with 8.0 GB of RAM memory in C++.

maximum size (*maxTSIterSize*). From the equation (*maxTSIterSize* $=800 \times (Gen/maxGen)$, *Gen* is the current generation of HA), the *maxTSIterSize* becomes bigger along with the *Gen* in the computation procedure. In the early stage of evolution of HA, because the GA cannot supply good initial individuals for TS, it is little possibility for TS to find the good solutions. The *maxTSIterSize* of TS is small. This can save the computational time of HA. In the late stage of evolution of HA, GA can provide good initial individual for TS. In this case, enlarging the *maxTSIterSize* of TS can help the TS to find good solutions. Therefore, maximum iteration size of TS is adaptive adjustment in the evolution process of HA. This can balance the exploitation and exploration of HA very well and save the computational time.

5.1.1. Experiment 1

The data of experiment 1 is adopted from Kacem et al. (2002). It contains 3 problems. Table 3 shows the experimental results and comparisons with other algorithms ($n \times m$ means that this problem contains n jobs and m machines, C_{max} means the makespan, CPU means the computational time in terms of seconds). HA represents the proposed algorithm. And the results of ALCGA, PSOSA, PVNS, AIA, TSPCB and HHS are adopted from Yuan et al. (2013b). The results of HGVNA, Heuristic, TABC and HGTS are adopted from Gao et al. (2008), Ziaee (2014), Gao et al. (2015) and Palacios et al. (2015b). The results marked by * are the best results among all these algorithms. From Table 3, the proposed HA obtains all the best results for three problems. Although seven algorithms obtain the same results, the proposed HA seems cost less computational time than other six algorithms. This means that the proposed approach has both good effectiveness and high efficiency for solving FJSP. Fig. 8 illustrates the Gantt chart of the problem 15 × 10.

5.1.2. Experiment 2

The data of experiment 2 is adopted from Fattahi et al. (2007) (http://www.ime.usp.br/~cris/fjs/benchmark/fmj/). It contains 20 problems. These problems are classified into two classes: small size FJSP (SFJS01-10), and medium and large size FJSP (MFJS01-10). Table 4 shows the experimental results and comparisons with other algorithms ($n \times m$ means that this problem contains n jobs and m machines, C_{max} means the makespan, CPU means the computational time in terms of seconds). HA represents the proposed algorithm. And the results of AIA and HHS are adopted from Yuan et al. (2013b). The results of M2 and MILP are adopted from

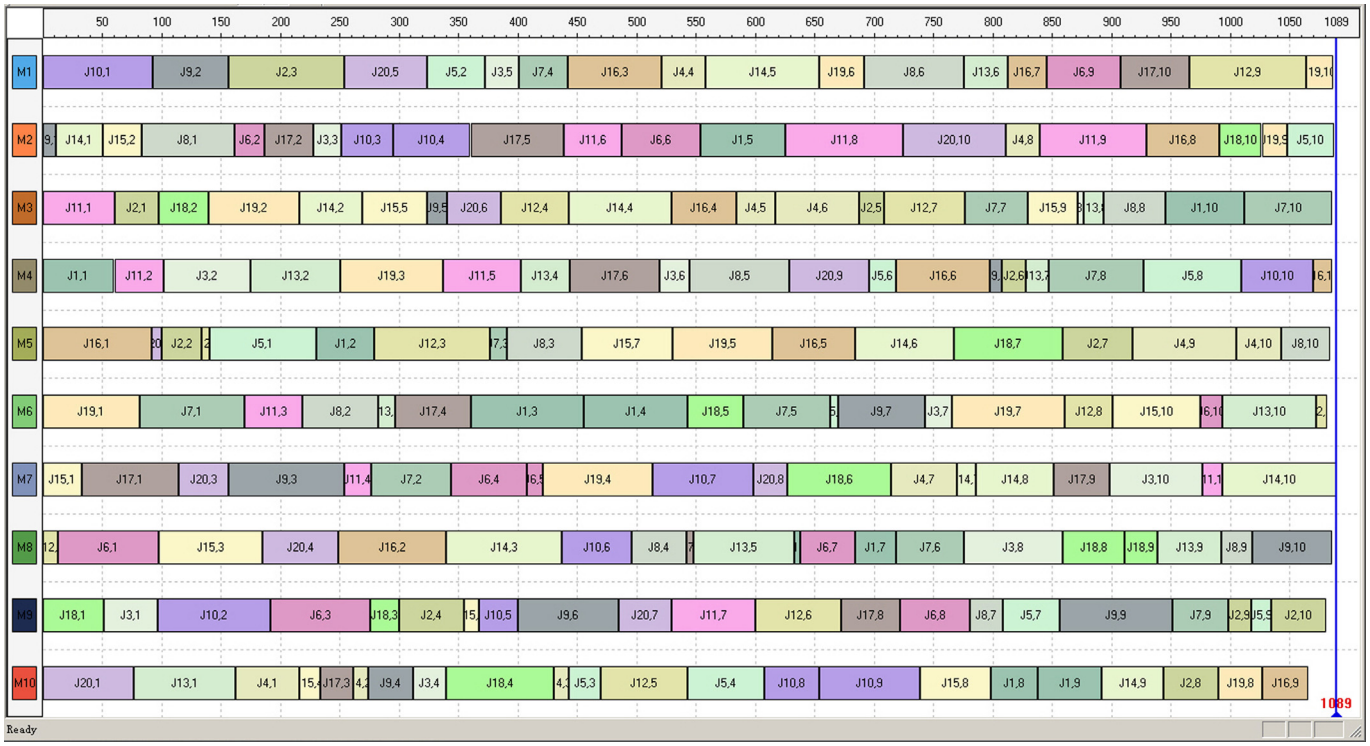

Fig. 14. Gantt chart of problem la27 of rdata in experiment 6.

Demir and Isleyen (2013) and Birgin et al. (2014). The results marked by * are the best results among all these algorithms. From Table 4, the proposed HA obtains all the best results for these problems. AIA, M2 and MILP had not obtained all the best results for these problems. Although both HHS and HA obtain the same results, the proposed HA seems cost less computational time than other algorithms. This means that the proposed approach has both good effectiveness and high efficiency for solving FJSP. Fig. 9 illustrates the Gantt chart of the problem MFJS10.

5.1.3. Experiment 3

The data of experiment 3 is adopted from Brandimarte (1993). It contains 10 problems. Table 5 shows the experimental results (minimizing makespan) and comparisons with the other algorithms ($n \times m$ means that this problem contains n jobs and m machines). Table 6 shows the computational time (s) and the comparisons with the other algorithms. HA represents the proposed algorithm. TS, LEGA, HO, GA, MAS, HGVNA, AIA, VNS, PVNS, HTS, DS and HAT represent the reported algorithms from the references Mastrolilli and Gambardella (2000), Ho et al. (2007), Zribi et al. (2007), Pezzella et al. (2008), Ennigrou and Ghedira, (2008), Gao et al. (2008), Bagheri et al. (2010), Amiri et al. (2010), Yazdani et al. (2010), Li et al. (2011), Hmida et al. (2010) and Tang et al. (2011). ABC, EA, HHS, HS, Heuristic, TABC, HGTS and MA2 represent the reported algorithms from the references Wang et al. (2012), Chiang and Lin (2013), Yuan and Xu (2013a), Yuan et al. (2013b), Ziaee (2014), Gao et al. (2015), Palacios et al. (2015b) and Yuan and Xu (2015). The results marked by * in Table 5 are the best results among these 21 algorithms. From Table 5, the proposed HA obtains all the best results for these problems. The other 20 comparing algorithms cannot obtain all the best results for these problems. This means that the proposed approach can obtain more and better results than the other approaches. From Table 6, the proposed HA seems cost less computational time than other algorithms except Heuristic. However, the results of Heuristic are the worst among all the 21 methods. The effectiveness of Heuristic is bad. Therefore, based on Tables 5 and 6, the results show that the proposed approach has both good effectiveness and high efficiency for solving FJSP. Fig. 10 illustrates the Gantt chart of the problem MK06.

5.1.4. Experiment 4

The data of experiment 4 is adopted from Barnes and Chambers (1996). It contains 21 problems. Table 7 shows the experimental results and comparisons with the other algorithms. ($n \times m$ means that this problem contains n jobs and m machines, C_{max} means the makespan, CPU means the computational time in terms of seconds). HA represents the proposed algorithm. And TS, HGVNA, DS, TSBM^{2}h, GA, HDEN$_2$ and HGTS represent the reported algorithms from the references Mastrolilli and Gambardella (2000), Gao et al. (2008), Hmida et al. (2010), Bozejko et al. (2010), Yuan and Xu (2013c), Demir and Isleyen (2014) and Palacios et al. (2015b). The results marked by * are the best results among these 8 algorithms. From Table 7, the proposed HA obtains 20 best results. TS obtained 11 best results; HGVNA obtained 10 best results; DS obtained 10 best results; TSBM^{2}h obtained 17 best results; HDEN$_2$ obtained 19 best results; GA obtained 13 best results; and HGTS obtained 19 best results. And the proposed HA seems cost less computational time than other algorithms. The experimental results show that the numbers of the best results obtained by HA are more than the other algorithms with less computational time. Therefore, based on Table 7, the results show that the proposed approach has both good effectiveness and high efficiency for solving FJSP. Fig. 11 illustrates the Gantt chart of the problem seti5x.

5.1.5. Experiment 5

The data of experiment 5 is adopted from Peres and Paulli (1997). It contains 18 problems. Table 8 shows the experimental results and comparisons with the other algorithms ($n \times m$ means that this problem contains n jobs and m machines, C_{max} means the makespan, CPU means the computational time in terms of seconds). HA represents the proposed algorithm. And IATS, TS, SA, GPSO, HO, HGVNA, DS, HHS/LNS, HGTS and MA2 represent the reported algorithms from the references Peres and Paulli (1997),

Table 11
The experimental results of vdata in experiment 6.

Problem	$n \times m$	TSN1	TSN2	IATS	TS	HA	HA CPU(s)[a]
mt06	6 × 6	47*	47*	47*	47*	47*	0.01
mt10	10 × 10	655*	655*	655*	655*	655*	0.22
mt20	20 × 5	1023	1023	1022*	1022*	1022*	5.91
la01	10 × 5	573	575	572	570*	570*	0.92
la02	10 × 5	531	530	529*	529*	529*	1.23
la03	10 × 5	482	481	479	477*	477*	1.29
la04	10 × 5	504	503	503	502*	502*	1.09
la05	10 × 5	464	461	460	457*	457*	10.47
la06	15 × 5	802	799*	800	799*	799*	2.95
la07	15 × 5	751	752	750	749*	749*	14.38
la08	15 × 5	766	766	766	765*	765*	16.75
la09	15 × 5	854	854	853	853*	853*	9.67
la10	15 × 5	805	805	805	804*	804*	2.54
la11	20 × 5	1073	1073	1071*	1071*	1071*	1.73
la12	20 × 5	940	940	936*	936*	936*	2.73
la13	20 × 5	1040	1041	1038*	1038*	1038*	4.34
la14	20 × 5	1071	1080	1070*	1070*	1070*	3.38
la15	20 × 5	1091	1091	1089*	1089*	1089*	17.51
la16	10 × 10	717*	717*	717*	717*	717*	0.44
la17	10 × 10	646*	646*	646*	646*	646*	0.56
la18	10 × 10	663*	663*	663*	663*	663*	0.48
la19	10 × 10	617*	617*	617*	617*	617*	1.22
la20	10 × 10	756*	756*	756*	756*	756*	0.41
la21	15 × 10	826	825	814	806	804*	44.19
la22	15 × 10	745	744	744	739	738*	41.23
la23	15 × 10	826	829	818	815	813*	35.37
la24	15 × 10	796	796	784	777*	777*	38.89
la25	15 × 10	770	769	757	756	754*	39.71
la26	20 × 10	1058	1058	1056	1054	1053*	41.43
la27	20 × 10	1088	1088	1087	1085*	1085*	35.29
la28	20 × 10	1073	1073	1072	1070*	1070*	42.17
la29	20 × 10	995	996	997	994*	994*	44.36
la30	20 × 10	1071	1070	1071	1069*	1069*	51.31
la31	30 × 10	1521	1521	1521	1520*	1520*	58.25
la32	30 × 10	1658*	1659	1659	1658*	1658*	65.78
la33	30 × 10	1498	1499	1498	1497*	1497*	53.82
la34	30 × 10	1536	1538	1537	1535*	1535*	62.55
la35	30 × 10	1553	1551	1551	1549*	1549*	67.02
la36	15 × 15	948*	948*	948*	948*	948*	4.57
la37	15 × 15	986*	986*	986*	986*	986*	4.88
la38	15 × 15	943*	943*	943*	943*	943*	2.13
la39	15 × 15	922*	922*	922*	922*	922*	4.33
la40	15 × 15	955*	955*	955*	955*	955*	4.61

[a] The CPU time on an Intel 2.0 GHz Core (TM) 2 Duo processor with 8.0 GB of RAM memory in C++.

Table 12
A comparison for problems la30, la35 and la40 of vdata in experiment 6.

Problem	$n \times m$	HA	HA CPU(s)[a]	MA2	MA2 CPU(s)[b]
la30	20 × 10	1069*	51.31	1072	53.43
la35	30 × 10	1549*	67.02	1550	83.03
la40	15 × 15	955*	4.61	955*	55.81

[a] The CPU time on an Intel 2.0 GHz Core (TM) 2 Duo processor with 8.0 GB of RAM memory in C++.
[b] The CPU time on an Intel Core i7-3520M 2.9 GHz processor with 8.0 GB of RAM memory in Java.

Mastrolilli and Gambardella (2000), Najid et al. (2002), Gao et al. (2006), Zribi et al. (2007), Gao et al. (2008), Hmida et al. (2010), Yuan and Xu (2013a), Palacios et al. (2015b) and Yuan and Xu (2015). The results marked by * are the best results among these 11 algorithms. From Table 8, the proposed HA obtains 7 best results. TS obtained 1 best results; DS obtained 1 best results; HHS/LNS obtained 6 best results; HGTS obtained 7 best results; and MA2 obtained 6 best results. And the proposed HA seems cost less computational time than other algorithms. The experimental results show that the numbers of the best results obtained by HA are more than the other algorithms (except HGTS) with less computational time. Although both HA and HGTS obtained the same number of best results, the computational time of HA is less than HGTS. And, some results of HA are the novel solutions for some benchmark problems which other algorithms did not find (problems 09a and 13a). Therefore, based on Table 8, the results show that the proposed approach has both good effectiveness and high efficiency for solving FJSP. Fig. 12 illustrates the Gantt chart of the problem 13a.

5.1.6. Experiment 6

The data of experiment 6 is adopted from Hurink et al. (1994). The problems were generated from the 43 classical JSP benchmark problems (mt06, mt10, mt20 and la01-la40). Hurink et al. (1994) generated three sets of benchmark problems: edata, rdata and vdata. Therefore, experiment 6 contains 129 problems.

For the problems in edata, Table 9 shows the experimental results and comparisons with the other algorithms ($n \times m$ means that this problem contains n jobs and m machines, CPU means the computational time in terms of seconds). HA represents the proposed algorithm. And TSN1 and TSN2 represent the methods from the reference Hurink et al. (1994); IATS and TS represent the reported algorithms from the references Peres and Paulli (1997) and Mastrolilli and Gambardella (2000). We have not found other papers which published their results on this problem set (edata). The results marked by * are the best results among these 5 algorithms. Because other algorithms did not provide the computational time. We cannot compare the computational time among proposed HA and other algorithms, and only give the computational time of proposed HA. From Table 9, the proposed HA obtains 42 best results. TSN1 obtained 9 best results; TSN2 obtained 8 best results; IATS obtained 17 best results and TS obtained 34 best results. And the computational time of proposed HA seems to be little. The experimental results show that the numbers of the best results obtained by HA are more than the other algorithms with less computational time. And, some results of HA are the novel solutions for some benchmark problems which other algorithms did not find (problems la21, la22, la26–29, la36, la38 and la40). Therefore, based on Table 9, the results show that the proposed approach has both superior effectiveness and high efficiency for solving FJSP. Fig. 13 illustrates the Gantt chart of the problem la36 of edata in experiment 6.

For the problems in rdata, Table 10 shows the experimental results and comparisons with the other algorithms ($n \times m$ means that this problem contains n jobs and m machines, CPU means the computational time in terms of seconds). HA represents the proposed algorithm. And TSN1 and TSN2 represent the methods from the reference Hurink et al. (1994); IATS and TS represent the reported algorithms from the references Peres and Paulli (1997) and Mastrolilli and Gambardella (2000); PBM^{2}h and TSM^{2}h represent the reported algorithms in the reference Bozejko et al. (2010). The results marked by * are the best results among these 7 algorithms. Because other algorithms did not provide the computational time. We cannot compare the computational time among proposed HA and other algorithms, and only give the computational time of proposed HA. From Table 10, the proposed HA obtains the best results for all problems. TSN1 obtained 4 best results; TSN2 obtained 4 best results; IATS obtained 10 best results; TS obtained 31 best results; PBM^{2}h obtained 10 best results among 15 problems and TSM^{2}h obtained 1 best result among 15 problems. And the computational time of proposed HA seems to be little. The experimental results show that the numbers of the best results obtained by HA are more than the other algorithms with less computational time. And, some results of HA are the novel solutions for some benchmark problems which other algorithms did not find (problems la01, la03, la07, la15, la23–25, la27–29, la36–38). Therefore, based on Table 10, the results show

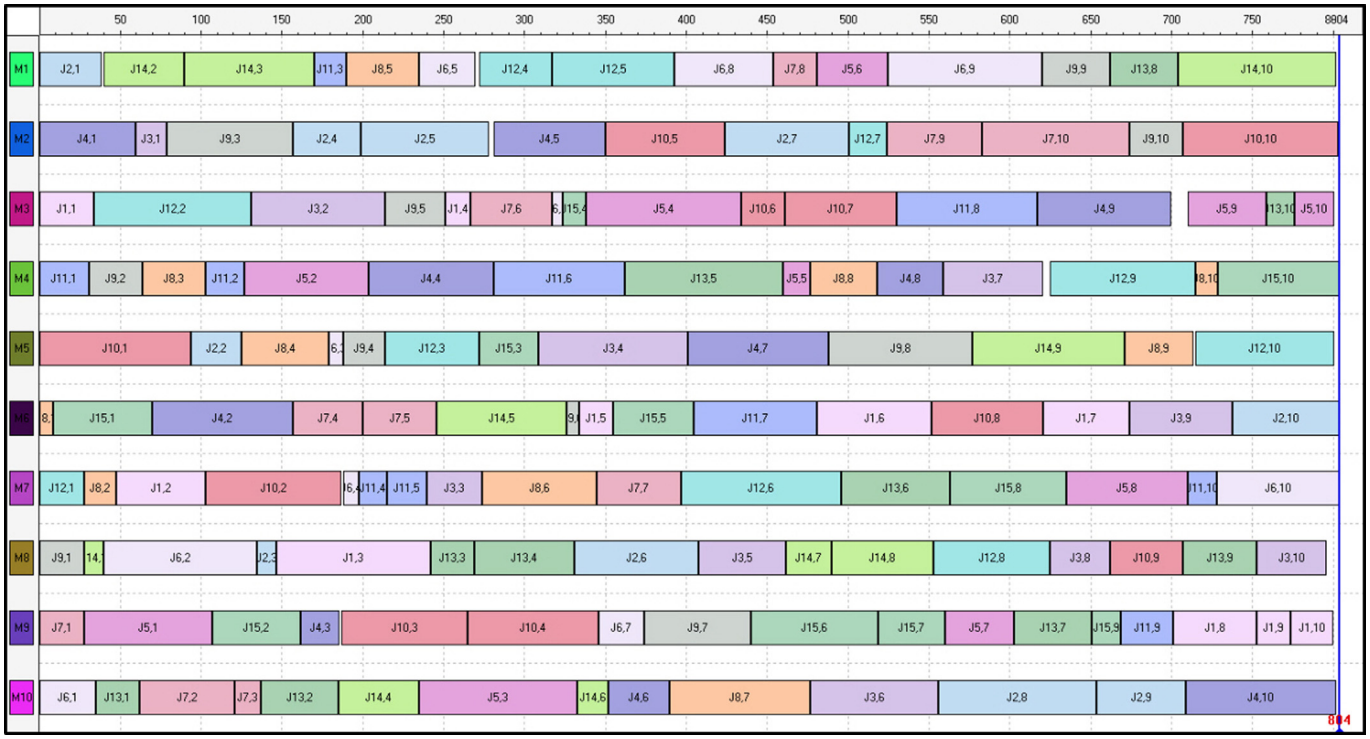

Fig. 15. Gantt chart of problem la21 of vdata in experiment 6.

that the proposed approach has both superior effectiveness and high efficiency for solving FJSP. Fig. 14 illustrates the Gantt chart of the problem la27 of rdata in experiment 6.

For problems in vdata, Table 11 shows the experimental results and comparisons with the other algorithms ($n \times m$ means that this problem contains n jobs and m machines, CPU means the computational time in terms of seconds). HA represents the proposed algorithm. And TSN1 and TSN2 represent the methods in the reference Hurink et al. (1994); IATS and TS represent the reported algorithms from the references Peres and Paulli (1997), Mastrolilli and Gambardella (2000). The results marked by * are the best results among these 5 algorithms. Because other algorithms did not provide the computational time. We cannot compare the computational time among proposed HA and other algorithms, and only give the computational time of proposed HA. From Table 11, the proposed HA obtains the best results for all problems. TSN1 obtained 13 best results; TSN2 obtained 13 best results; IATS obtained 19 best results and TS obtained 38 best results. And the computational time of proposed HA seems to be little. The experimental results show that the numbers of the best results obtained by HA are more than the other algorithms with less computational time. And, some results of HA are the novel solutions for some benchmark problems which other algorithms did not find (problems la21–23, la25–26). For the problem la21 of vdata in experiment 6, the result of the HGVNA proposed by Gao et al. (2008) was 805. It was worse than 804 obtained by the proposed HA. And for the problems la30, la35 and la40 of vdata, a comparison between proposed HA and MA2 adopted from Yuan and Xu (2015) is reported in Table 12 ($n \times m$ means that this problem contains n jobs and m machines, CPU means the computational time in terms of seconds). The results in Table 12 show that the proposed HA is better than MA2 regardless of the solution accuracy and the computational time. Therefore, based on Tables 11 and 12, the results show that the proposed approach has both superior effectiveness and high efficiency for solving FJSP. Fig. 15 illustrates the Gantt chart of the problem la21 of vdata in experiment 6.

5.2. Discussions

From the results of each experiment, the proposed HA can obtain the best results for the most problems. And for many problems, the solutions of HA are preferable to the results of other algorithms. Therefore, the above experimental results indicate that the proposed HA can solve the FJSP effectively and obtain many good and useful results. And, some of them are new solutions for some benchmark problems which were not found previously. The computational time of proposed HA also has been compared with other algorithms, and the proposed HA seems cost less computational time than other algorithms. Therefore, the above experimental results show that the proposed approach has both superior effectiveness and high efficiency for solving FJSP.

The reasons are as follows. Firstly, the proposed HA is the hybrid algorithm of GA and TS. It combines the global search and local search by using GA to perform exploration and TS to perform exploitation. The GA has powerful global searching ability and TS has vabluable local searching ability. Secondly, in order to solve the FJSP effectively, effective encoding method, genetic operators and neighborhood structure are employed and designed in this method. Finally, in the parameters setting, maximum iteration size of TS is adaptive adjustment in the evolution process of HA. This can save the computational time. Therefore, the proposed approach has very good searching ability with little computational time and can keep good balance between intensification and diversification.

6. Conclusions and future works

In this paper, by hybridizing the population based evolutionary searching ability of GA with local improvement ability of TS to balance exploration and exploitation, an effective HA has been proposed for FJSP with the objective to minimize the makespan. Six famous benchmark instances (including 201 open problems) have been used to test the performance of HA. The experimental

results demonstrate that the proposed HA has achieved significant improvement for solving FJSP regardless of the solution accuracy and the computational time. And, the proposed method obtains the new best solutions for several benchmark problems.

The contributions of this research include:

- A HA which hybridizes the GA and TS has been proposed to solve the FJSP. The proposed HA combines the advantages of GA and TS together. And based on the features of FJSP, we design and employ all the parts of the HA approach. This algorithm can reflect the essential characteristics of FJSP. The experimental results show that the proposed HA has been successfully used to solve the FJSP. It is a promising and very effective method on the research of the FJSP.
- The proposed algorithm combines the advantages of evolutionary algorithm and LS method. It provides a new way to solve other scheduling problems in the manufacturing field, such as JSP, the integrated process planning and scheduling problem and so on. And the experimental results of the FJSP show that this algorithm may solve these problems effectively, because these scheduling problems contain several similar aspects with the FJSP.

In the future, the following works can be made. Firstly, we can combine other algorithms in evolutionary algorithms and other LS methods to construct more effective hybrid algorithm to solve the scheduling problems. After all, some benchmark instances have not found the optimum solutions. The current approaches are still far from being enough. Secondly, the proposed HA can solve the mono-objective FJSP effectively. We can extend its use by combining this method with some methods which deal with multi-objective to solve the multi-objective FJSP or other scheduling problems in the manufacturing field.

Acknowledgment

The authors would like to thank the editor and anonymous referees whose comments helped a lot in improving this paper. This research work is supported by the Natural Science Foundation of China (NSFC) under Grant nos. 51435009 and 51375004, National Key Technology Support Program under Grant no. 2015BAF01B04, and Youth Science & Technology Chenguang Program of Wuhan under Grant no. 2015070404010187.

References

Amiri, M., Zandieh, M., Yazdani, M., Bagheri, A., 2010. A variable neighbourhood search algorithm for the flexible job shop scheduling problem. Int. J. Prod. Res. 48 (19), 5671–5689.

Bagheri, A., Zandieh, M., Mahdavi, I., Yazdani, M., 2010. An artificial immune algorithm for the flexible job shop scheduling problem. Future Gener Comput. Syst. 26, 533–541.

Barnes J.W., Chambers J.B., 1996. Flexible job shop scheduling by tabu search Graduate Program in Operations ReseArch. and Industrial Engineering, The University of Texas at Austin; Technical report series: ORP96-09.

Baykasoglu, A., Ozbakir, L., 2010. Analyzing the effect of dispatching rules on the scheduling performance through grammar based flexible scheduling system. Int J. Prod. Econ. 124, 369–381.

Birgin, E.G., Feofiloff, P., Fernandes, C.G., Melo, E.L., Oshiro, M.T.I., Ronconi, D.P., 2014. A MILP model for an extended version of the Flexible Job Shop Problem. Optim. Lett. 8, 1417–1431.

Birgin, E.G., Ferreira, J.E., Ronconi, D.P., 2015. List scheduling and beam search methods for the flexible job shop scheduling problem with sequencing flexibility. Eur. J. Oper. Res. 247, 421–440.

Bozejko, W., Uchronski, M., Wodecki, M., 2010. Parallel hybrid metaheuristics for the flexible job shop problem. Comput.Ind. Eng. 59, 323–333.

Brandimarte, P., 1993. Routing and scheduling in a flexible job shop by taboo search. Ann. Oper. Res. 41 (3), 157–183.

Brucker, P., Schile, R., 1990. Job-shop scheduling with multi-purpose machines. Computing 45 (4), 369–375.

Calleja, G., Pastor, R., 2014. A dispatching algorithm for flexible job-shop scheduling with transfer batches: an industrial application. Prod. Plan. Control. 25 (2), 93–109.

Chaudhry, I.A., Khan, A.A., 2016. A research survey: review of flexible job shop scheduling techniques. Int. Trans. Oper. Res. 23, 551–591.

Chen, H., Ihlow, J., Lehmann, C., 1999. A genetic algorithm for flexible job shop scheduling. In: Proceedings of IEEE International conference on Robotics and Automation, 2; pp. 1120–1125.

Chen, J.C., Wu, C.C., Chen, C.W., Chen, K.H., 2012. Flexible job shop scheduling with parallel machines using genetic algorithm and grouping genetic algorithm. Exp. Syst. Appl. 39, 10016–10021.

Chiang, T.S., Lin, H.J., 2013. A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling. Int. J. Prod. Econ. 141, 87–98.

Demir, Y., Isleyen, S.K., 2013. Evaluation of mathematical models for flexible job shop scheduling problems. Appl. Math. Model. 37, 977–988.

Demir, Y., Isleyen, S.K., 2014. An effective genetic algorithm for flexible job shop scheduling with overlapping in operations. Int. J. Prod. Res. 52 (13), 3905–3921.

Driss, I., Mouss, K.N., Laggoun, A., 2015. A new genetic algorithm for flexible job-shop scheduling problems. J. Mech. Sci. Technol. 29 (3), 1273–1281.

Ennigrou, M., Ghedira, K., 2008. New local diversification techniques for flexible job shop scheduling problem with a multi-agent approach. Auton. Agent Multi-Agent Syst. 17, 270–287.

Fattahi, P., Meharabad, M.S., Jolai, F., 2007. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. J. Intell. Manuf. 18, 331–342.

Gao, L., Peng, C.Y., Zhou, C., Li, P.G., 2006. Solving flexible job shop scheduling problem using general particle swarm optimization. In: Proceedings of the 36th CIE Conference on Computers and Industrial Engineering, pp. 3018–3027.

Gao, J., Sun, L.Y., Gen, M., 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput. Oper. Res. 35, 2892–2907.

Gao, K.Z., Suganthan, P.N., Pan, Q.K., Chua, T.J., Cai, T.X., Chong, C.S., 2014. Pareto based grouping discrete harmony search algorithm for multi-objective flexible job shop scheduling. Inform. Sci. 289, 76–90.

Gao, K.Z., Suganthan, P.N., Chua, T.J., Chong, C.S., Cai, T.X., Pan, Q.K., 2015. A two-stage artificial bee colony algorithm scheduling flexible job shop scheduling problem with new job insertion. Exp. Syst. Appl. 42, 7652–7663.

Giovanni, L.D., Pezzella, F., 2010. An improved genetic algorithm for the distributed and flexible job shop scheduling problem. Eur. J. Oper. Res. 200, 395–408.

Girish, B.S., Jawahar, N., 2009. Scheduling job shop associated with multiple routings with genetic and ant colony heuristics. Int. J. Prod. Res. 47 (14), 3891–3917.

Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers Norwell, MA, USA.

Gomes, M.C., Barbosa-Povoa, A.P., Novais, A.Q., 2005. Optimal Scheduling for Flexible Job Shop Operation. Int. J. Prod. Res. 43 (11), 2323–2353.

Graey, M.R., Jonmson, D.S., Sethi, R., 1976. The complexity of flow shop and job shop scheduling. Math. Oper. Res 1, 117–129.

Guevara, G., Pereira, A.I., Ferreira, A., Barbosa, J., Leitao, P., 2014. Genetic algorithm for flexible job shop scheduling problem – a case study. In: Proceedings of the International Conference on Numerical Analysis and Applied Mathematics 2014 (ICNAAM-2014) 1648, 140006-1–140006-4.

Hmida, A.B., Haouari, M., Huguet, M.J., Lopez, P., 2010. Discrepancy search for the flexible job shop scheduling problem. Comput. Ope. Res. 37, 2192–2201.

Ho, N.B., Tay, J.C., Lai, E.M.K., 2007. An effective architecture for learning and evolving flexible job shop schedules. Eur. J. Oper. Res. 179, 316–333.

Ho, N.B., Tay, J.C., 2008. Solving multiple objective flexible job shop problems by evolution and local search. IEEE Trans Syst, Man, Cybern. – Part C: Appl. Rev. 38 (5), 674–685.

Huang, R.H., Yang, C.L., Cheng, W.C., 2013. Flexible job shop scheduling with due window-a two-pheromone ant colony approach. Int. J. Prod. Econ. 141, 685–697.

Hurink, J., Jurisch, B., Thole, M., 1994. Tabu search for the job shop scheduling problem with multi-purpose machines. OR Spektrum 15, 205–215.

Jensen, M.T., 2003. Generating robust and flexible job shop schedules using genetic algorithms. IEEE Trans. Evol. Comput. 7 (3), 275–288.

Jia, S., Hu, Z.H., 2014. Path-relinking tabu search for the multi-objective flexible job shop scheduling problem. Comput. Oper. Res. 47, 11–26.

Kacem, I., Hammadi, S., Borne, P., 2002. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Trans. Syst., Man, Cybern., Part C: Appl. Rev. 32 (1), 1–13.

Karimi, H., Rahmati, S.H.A., Zandieh, M., 2012. An efficient knowledge-based algorithm for the flexible job shop scheduling problem. Knowl.-Based Syst. 36, 236–244.

Li, J.Q., Pan, Q.K., Gao, K.Z., 2011a. Pareto based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. Int. J. Adv. Manuf. Technol. 55, 1159–1169.

Li, J.Q., Pan, Q.K., Suganthan, P.N., Chua, T.J., 2011b. A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. Int. J. Adv. Manuf. Technol. 52 (5–8), 683–697.

Liou, C.D., Hsieh, Y.C., 2015. A hybrid algorithm for the multi-stage flow shop group scheduling with sequence-dependent setup and transportation times. Int. J. Prod. Econ. 170, 258–267.

Mastrolilli, M., Gambardella, L.M., 2000. Effective neighbourhood functions for the flexible job shop problem. J. Sched. 3, 3–20.

Mati, Y., Lahlou, C., Peres, S.D., 2010. Modelling and solving a practical flexible job shop scheduling problem with blocking constraints. Int. J. Prod. Res. 49 (8), 2169–2182.
Moslehi, G., Mahnam, M., 2011. A pareto approach to multi-objective flexible job shop scheduling problem using particle swarm optimization and local search. Int. J. Prod. Econ. 129, 14–22.
Najid, N.M., Peres, S.D., Zaidat, A., 2002. A modified simulated annealing method for flexible job shop scheduling problem. In: Proceedings of 2002 International Conference on Systems, Man, and Cybernetics, Piscataway, pp. 6–12.
Ozguven, C., Ozbakir, L., Yavuz, Y., 2010. Mathematical models for job shop scheduling problems with routing and process plan flexibility. Appl. Math. Model. 34, 1539–1548.
Palacios, J.J., Gonzalez-Rodriguez, I., Vela, C.R., Puente, J., 2015a. Coevolutionary makespan optimisation through different ranking methods for the fuzzy flexible job shop. Fuzzy Sets Syst. 278 (1), 81–97.
Palacios, J.J., Gonzalez, M.A., Vela, C.R., Rodriguez, I.G., Puente, J., 2015b. Genetic tabu search for the fuzzy flexible job shop problem. Comput. Oper. Res. 54, 74–89.
Paulli, J., 1995. A hierarchical approach for the FMS scheduling problem. Eur. J. Oper. Res. 86 (1), 32–42.
Pezzella, F., Morganti, G., Ciaschetti, G., 2008. A genetic algorithm for the flexible job shop scheduling problem. Comput. Oper. Res. 35, 3202–3212.
Peres, S.D., Paulli, J., 1997. An integrated approach for modeling and solving the general multiprocessor job shop scheduling using tabu search. Ann. Oper. Res. 70, 281–306.
Roshanaei, V., Azab, A., ElMaraghy, H., 2013. Mathematical modelling and a meta-heuristic for flexible job shop scheduling. Int. J. Prod. Res. 51 (20), 6247–6274.
Rossi, A., Dini, G., 2007. Flexible job shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. Robot. Comput. Integr. Manuf. 23, 503–516.
Rossi, A., 2014. Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships. Int. J. Prod. Econ. 153, 253–267.
Seebacher, G., Winkler, H., 2014. Evaluating flexibility in discrete manufacturing based on performance and efficiency. Int. J. Prod. Econ. 153, 340–351.
Tang, J.C., Zhang, G.J., Lin, B.B., Zhang, B.X., 2011. A hybrid algorithm for flexible job shop scheduling problem. Proced. Eng. 15, 3678–3683.
Tay, J.C., Ho, N.B., 2008. Evolving dispatching rules using genetic programming for solving multi-objective flexible job shop problems. Comput. Ind. Eng. 54, 453–473.
Torabi, S.A., Karimi, B., Ghomi, S.M.T.F., 2005. The common cycle economic lot scheduling in flexible job shops: the finite horizon case. Int. J. Prod. Econ. 97, 52–65.
Vilcot, G., Billaut, J.C., 2011. A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem. Int. J. Prod. Res. 49 (23), 6963–6980.
Wang, S.J., Yu, J.B., 2010. An effective heuristic for flexible job shop scheduling problem with maintenance activities. Comput. Ind. Eng 59, 436–447.
Wang, L., Zhou, G., Xu, Y., Wang, S.Y., Liu, M., 2012. An effective artificial bee colony algorithm for the flexible job shop scheduling problem. Int J. Adv. Manuf. Technol. 60, 303–315.
Wang, X.L., Cheng, T.C.E., 2015. A heuristic for scheduling jobs on two identical parallel machines with a machine availability constraint. Int.l J. Prod. Econ. 161, 74–82.
Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1 (1), 67–82.
Xiong, J., Xing, L.N., Chen, Y.W., 2013. Robust scheduling for multi-objective flexible job shop problems with random machine breakdowns. Int. J. Prod. Econ. 141, 112–126.
Yazdani, M., Amiri, M., Zandieh, M., 2010. Flexible job shop scheduling with parallel variable neighborhood search algorithm. Exp. Syst. Appl. 37, 678–687.
Yuan, Y., Xu, H., 2013a. An integrated search heuristic for large-scale flexible job shop scheduling problems. Comput. Oper. Res. 40, 2864–2877.
Yuan, Y., Xu, H., Yang, J.D., 2013b. A hybrid harmony search algorithm for the flexible job shop scheduling problem. Appl. Soft Comput. 13, 3259–3272.
Yuan, Y., Xu, H., 2013c. Flexible job shop scheduling using hybrid differential evolution algorithms. Comput. Ind. Eng. 65, 246–260.
Yuan, Y., Xu, H., 2015. Multiobjective flexible job shop scheduling using memetic algorithms. IEEE Trans. Autom. Sci. Eng. 12 (1), 336–353.
Zhang, G.H., Gao, L., Shi, Y., 2011. An effective genetic algorithm for the flexible job shop scheduling problem. Exp. Syst. Appl. 38, 3563–3573.
Zhang, Q., Manier, H., Manier, M.A., 2012. A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times. Comput. Oper. Res. 39, 1713–1723.
Ziaee, M., 2014. A heuristic algorithm for solving flexible job shop scheduling problem. Int. J. Adv. Manuf. Technol. 71, 519–528.
Zribi, N., Kacem, I., Kamel, A.E., Borne, P., 2007. Assignment and scheduling in flexible job shops by hierarchical optimization. IEEE Trans. Syst., Man, Cybern.-Part C: Appl. Rev. 37 (4), 652–661.