Applied Soft Computing 10 (2010) 888–896

Contents lists available at ScienceDirect

Applied Soft Computing

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

A Knowledge-Based Ant Colony Optimization for Flexible Job Shop Scheduling Problems

Li-Ning Xing*, Ying-Wu Chen, Peng Wang, Qing-Song Zhao, Jian Xiong

Department of Management Science and Engineering, College of Information System and Management, National University of Defense Technology, Changsha 410073, China

ARTICLE INFO

Article history:
Received 22 April 2008
Received in revised form
12 September 2009
Accepted 18 October 2009
Available online 24 October 2009

Keywords:
Combinatorial optimization
Ant Colony Optimization
Flexible Job Shop Scheduling

ABSTRACT

A Knowledge-Based Ant Colony Optimization (KBACO) algorithm is proposed in this paper for the Flexible Job Shop Scheduling Problem (FJSSP). KBACO algorithm provides an effective integration between Ant Colony Optimization (ACO) model and knowledge model. In the KBACO algorithm, knowledge model learns some available knowledge from the optimization of ACO, and then applies the existing knowledge to guide the current heuristic searching. The performance of KBACO was evaluated by a large range of benchmark instances taken from literature and some generated by ourselves. Final experimental results indicate that the proposed KBACO algorithm outperforms some current approaches in the quality of schedules.

1. Introduction

Scheduling problems have a vital role in recent years due to the growing consumer demand for variety, reduced product life cycles, changing markets with global competition and rapid development of new technologies. The Job Shop Scheduling Problem (JSSP) is one of the most popular scheduling models existing in practice, which is among the hardest combinatorial optimization problems [1]. Many approaches, such as, Simulated Annealing (SA) [2], Tabu Search (TS) [3], Genetic Algorithm (GA) [4], Ant Colony Optimization (ACO) [5], Neural Network (NN) [6], Evolutionary Algorithm (EA) [7] and other heuristic approach [8–10], have been successfully applied to JSSP.

In order to match today's market requirements, manufacturing systems need not only automated and flexible machines, but also flexible scheduling systems. The Flexible Job Shop Scheduling Problem extends JSSP by assuming that, for each given operation, it can be processed by any machine from a given set. Bruker and Schlie [11] were among the first to address this problem. The difficulties of FJSSP can be summarized as follows.

(1) Assignment of an operation to an appropriate machine;
(2) Sequencing the operations on each machine;
(3) A job can visit a machine more than once (called recirculation).

These three features significantly increase the complexity of finding even approximately optimal solutions.

Although EA has been applied to solve numerous applications, but there have two disadvantages for solving combinational optimization problems by a canonical EA.

(1) A canonical EA is a 'generation-evaluation' type of searching technique, which only uses fitness value or objective value to guide the evolutionary search [12].
(2) The optimization results obtained by the canonical EA are still limited due to the reliance on randomized natural selection and recombination [13].

For improving the performance of EA, several researches integrated some optimization strategies into the EA [14,15]. Also, the study of interaction between evolution and learning for solving optimization problems has been attracting much attention [16–19]. The diversity of these approaches has motivated our pursuit for a uniform framework called Knowledge-Based Heuristic Searching Architecture (KBHSA), which integrates knowledge model and heuristic searching model to search an optimal solution. We demonstrate the performance of this architecture in the instantiation of the Knowledge-Based Ant Colony Optimization (KBACO) which is applied to common benchmark problems. Experimental results show that KBACO algorithm outperforms previous approaches for solving the FJSSP.

The remainder of this paper is organized as follows. Section 2 reviews some recent works related to the FJSSP. Section 3 describes the proposed architecture (KBHSA) and its instantiation (KBACO).

* Corresponding author.
E-mail addresses: xln_2002@nudt.edu.cn, xinglining@gmail.com (L.-N. Xing), ywchen@nudt.edu.cn (Y.-W. Chen), phd2999@gmail.com (P. Wang), zqszqr@163.com (Q.-S. Zhao), xiongjian1984@hotmail.com (J. Xiong).

1568-4946/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2009.10.006

Section 4 presents and analyzes the performance of KBACO when applied to solve common benchmarks in literature and others generated by ourselves. Finally, Section 5 gives some concluding remarks and directions for future work.

2. Related works

There are three parts in this section. Section 2.1 gives the formal definition of FJSSP. Also, a practical model of the FJSSP is also described. Section 2.2 reviews recent related works for solving the FJSSP. Current studies of the interaction between evolution and learning for solving optimization problems are discussed in Section 2.3.

2.1. Problem formulation

Generally, the FJSSP can be formulated as follows [20].

(1) There is a set of n jobs that plan to process on m machines;
(2) Let $J = \{J_i\}_{1\le i\le n}$, indexed i, be a set of n jobs;
(3) Let $M = \{M_k\}_{1\le k\le m}$, indexed k, be a set of m machines;
(4) Each job J_i consists of a predetermined sequence of operations;
(5) Each operation O_{ij} (operation j of job i) can be processed without interruption on one of a set of machines $M_{ij}(M_{ij} \subseteq M)$. We denote p_{ijk} to be processing time of O_{ij} on machine M_k.
(6) The objective of FJSSP is to find a minimum makespan.

Hypotheses considered in this paper are listed as follows [21].

(1) Setting up times of machines and move times between operations are negligible;
(2) Machines are independent from each other;
(3) Jobs are independent from each other;
(4) At a given time, a machine can execute at most one operation. It becomes available to other operations only if the operation which is processing is completed;
(5) There are no precedence constraints among the operations of different jobs;
(6) No more than one operation of the same job can be executed at a time.

Kacem et al. [22] classified the FJSSP into two sub-problems as follows.

(1) Total FJSSP (T-FJSSP): each operation can be processed on any machine;
(2) Partial FJSSP (P-FJSSP): each operation can be processed on one machine of subset of M.

An instance of a FJSSP with two-job and three-machine is presented in Fig. 1. The operation sequence, machine alternatives and processing times are displayed in Table 1. The Gant chart of a feasible solution is presented in Fig. 1. The makespan for this solution is 53. This solution is not the optimal solution. The optimal solution must be found by the algorithm.

2.2. Previous works

Hierarchical approaches and integrated approaches are two types of approaches, which have been used to solve the realistic instance of FJSSP [21].

(1) In hierarchical approaches, assignment of operations to machines and the sequencing of operations on the resources or machines are treated separately. Hierarchical approaches are based on the idea of decomposing the original problem in order to reduce its complexity. Brandimarte [23] was the first to use the decomposition for the FJSSP. He solved the routing sub-problem using some existing dispatching rules and then focused on the scheduling sub-problem, which is solved using a TS heuristic. Tung et al. [24] developed a similar approach for scheduling a flexible manufacturing system. Kacem et al. [25] proposed a GA controlled by the assigned model which is generated by the approach of localization. Xia and Wu [21] present an effective hybrid optimization approach for the multi-objective FJSSP. They make use of Particle Swarm Optimization (PSO) to assign operations on machines and SA algorithm to schedule operations.
(2) Integrated approaches were used by considering assignment and scheduling at the same time. Hurink et al. [26] proposed a TS heuristic in which reassignment and rescheduling are considered as two different types of moves. Dauzere-Peres and Paulli presented a TS procedure based on their proposed neighborhood structure for the FJSSP [27]. In their approach, there is no distinction between reassigning and resequencing an operation. Mastrolilli and Gambardella [28] improved Dauzere-Peres' TS techniques and presented two neighborhood functions.

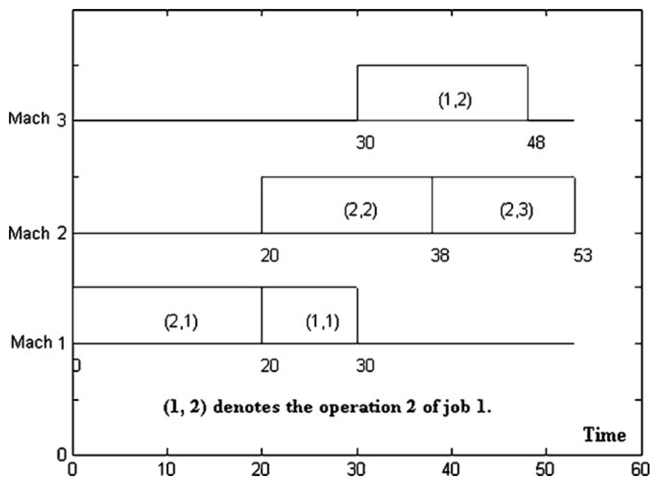


Fig. 1. The Gant chart of a feasible solution.

2.3. Learning for evolution

Generally, there are two approaches to implement the interaction between evolution and learning [20]. The first one is to keep good features of previous individuals to improve the fitness of individuals in the current generation. The second one is to keep bad features of previous individuals to avoid infeasibility in the current generation. Reynolds [16] implemented the interaction between

Table 1
Example of a two-job, three-machine FJSSP.

Jobs	Operation sequence	Operations	Machine alternative	Processing time (s)
J_1	O_{11}, O_{12}	O_{11}	M_1	10
			M_2	15
		O_{12}	M_2	12
			M_3	18
J_2	O_{21}, O_{22}, O_{23}	O_{21}	M_1	20
			M_3	25
		O_{22}	M_1	25
			M_2	18
		O_{23}	M_2	15
			M_3	25

evolution and learning by an external memory, which preserves the beliefs encoded in individuals (good traits from each generation). Similarly, Branke [17] uses a memory to keep good individuals. Louis and McDonnell [18] introduce case-based reasoning methods to select individuals from the case-based memory. Michalski [19] presents learnable evolution model (LEM), where machine learning is used purely to generate a new population. Ho et al. [20] proposed a learnable genetic architecture (LEGA) for learning and evolving of FJSSP. The above approaches for integrating learning into the evolution reveal an interaction between evolution and learning that is implemented in different ways. In general, they have used version space [13], case-based memory [18] or an AQ-learning system [19]. The reported performance results indicate that these approaches can significantly outperform the standard heuristic approach.

3. The proposed approach

The proposed KBHSA architecture and its instantiation (KBACO) are presented in this section. The KBHSA architecture is elaborated in Section 3.1 and its instantiation (KBACO) is described in Section 3.2. The implementation of KBACO algorithm is detailed elaborated in Section 3.3 to Section 3.7.

3.1. Knowledge-Based Heuristic Searching Architecture

Optimization problems are challenging as there are numerous feasible solutions that satisfy the constraints. When the number of solutions is too large to explicitly look at each one, several optimization strategies have been found to be extraordinary efficacious [29–31]. At the same time, an integration of genotypic knowledge representation, learning methodology and heuristic approach can yield significant improvements in solution quality and computational time [20]. Taking one with another, integrating knowledge model and heuristic searching model can be seen as a useful tool in the search of an optimal solution. The idea of using the hybrid models, which combined the knowledge model with heuristic searching model, for solving optimization problems is not just a theoretical concept and its practicality has already been demonstrated [32–34]. The diversity of the hybrid approaches has motivated our pursuit for a uniform framework for solving the combinatorial optimization problems.

The proposed KBSHA architecture is functionally divided into two modules, namely, knowledge model and heuristic searching model. The heuristic searching model takes charge of searching through the vast solution space and identifying an optimal solution. The knowledge model learns some available knowledge from the optimization, and then applies the existing knowledge to guide the current heuristic searching. The working mechanism of KBSHA architecture can be showed as Fig. 2.

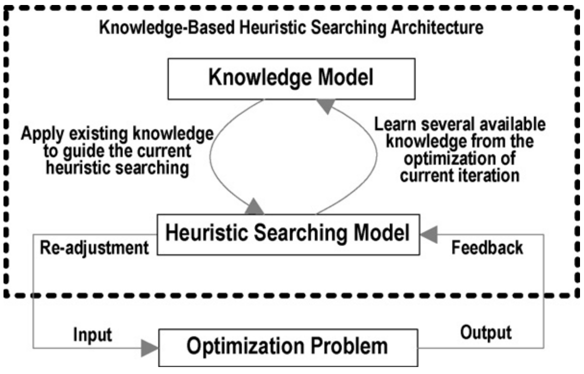

Knowledge-Based Heuristic Searching Architecture
Knowledge Model
Apply existing knowledge to guide the current heuristic searching
Learn several available knowledge from the optimization of current iteration
Heuristic Searching Model
Re-adjustment
Feedback
Input
Output
Optimization Problem

Fig. 2. The mechanism of the Knowledge-Based Heuristic Searching Architecture.

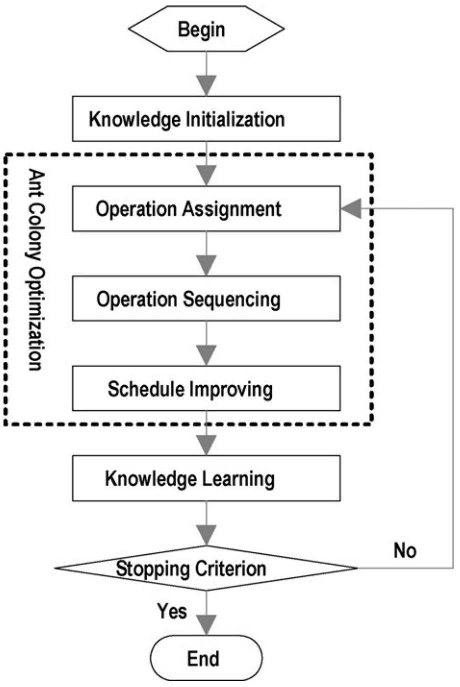


Fig. 3. The framework of the Knowledge-Based Ant Colony Optimization.

3.2. Knowledge-Based Ant Colony Optimization

We propose an instantiation of KBSHA called Knowledge-Based Ant Colony Optimization (KBACO) in this part. The framework of KBACO is displayed as Fig. 3. From Fig. 3, we can see that there are five modules in the proposed KBACO. The proposed KBACO algorithm can be briefly sketched as follows.

(1) The genotypic knowledge was initialized;
(2) A group of feasible solutions were constructed using ACO algorithm guided by the existing knowledge;
(3) The genotypic knowledge was updated by the optimization of current iteration.
(4) These processes were repeated till the stopping criterion satisfied.

3.3. Knowledge initialization

The structure of the knowledge model is shown in Fig. 4.

(1) *Elite solution knowledge*. After each iteration, predefined numbers of best solution are selected to insert into the elite solution set, and we called them as the elite solution knowledge. Among the elite solution set, the total amount of elite solution is fixed, and the worst solution will be replaced by a good one after each insertion. In the initialization phase, the elite solution

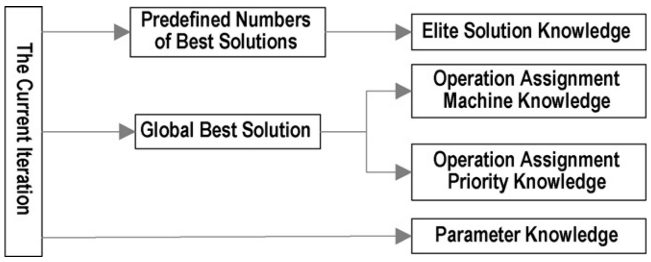

The Current Iteration
Predefined Numbers of Best Solutions
Elite Solution Knowledge
Global Best Solution
Operation Assignment Machine Knowledge
Operation Assignment Priority Knowledge
Parameter Knowledge

Fig. 4. Structure of knowledge model.

Table 2
An example of OAMK knowledge matrix.

Operation	Mach 1	Mach 2	Mach 3
O_{11}	0.8	0.5	0
O_{12}	0	0.6	0.3
O_{21}	0.5	0	0.6
O_{22}	0.3	0.6	0
O_{23}	0	0.7	0.4

The gray region denotes a practical OAMK knowledge matrix. In this matrix, *OAMK*(1, 2, 1) = 0 denotes the operation O_{12} cannot be assigned to machine 1, *OAMK*(1, 2, 2) = 0.6 denotes the operation O_{12} can be assigned to machine 2 with 0.6/(0.6 + 0.3) ≈ 66.7% probability, and *OAMK*(1, 2, 3) = 0.3 denotes O_{12} can be assigned to machine 3 with 0.3/(0.6 + 0.3) ≈ 33.3% probability. In the initialization phase, each elements of matrix *OAMK* was initialized using formula (1).

Table 3
An example of OAPK knowledge matrix.

Operation	Priority		
	1	2	3
O_{11}	0.9	0.6	0.1
O_{12}	0.4	0.8	0.4
O_{21}	0.6	0.7	0.3
O_{22}	0.3	0.8	0.2
O_{23}	0.3	0.7	0.9

The gray region denotes a practical OAPK knowledge matrix. In this matrix, *OAPK*(1, 2, 1) = 0.4 denotes O_{12} can be processed using the first priority rank with 0.4/(0.4 + 0.8 + 0.4) = 25% probability, *OAPK*(1, 2, 2) = 0.8 denotes the operation O_{12} can be processed using the second priority rank with 0.8/(0.4 + 0.8 + 0.4) = 50% probability, and *OAPK*(1, 2, 3) = 0.4 denotes O_{12} can be processed using the third priority rank with 0.4/(0.4 + 0.8 + 0.4) = 25% probability. In the initialization phase, a level τ_0 is initialized for all the elements of knowledge matrix *OAPK*, where τ_0 is a relatively small quantity.

knowledge will be initialized as null. In the knowledge learning phrase, the elite solution knowledge will be updated by the predefined numbers of best solution of each iteration.

(2) *Operation assignment machine knowledge* (OAMK). It is the accumulative knowledge of assigning the given operation to a more appropriate machine. OAMK is extracted from the global best solution (the obtained optimal solution from the beginning trials) of FJSSP. In this paper, we define a matrix *OAMK* with size $n \times num \times m$ for the OAMK, where n denotes the number of jobs, num denotes the maximal operation number of all jobs, and m denotes the number of machines. An example of OAMK matrix is showed as Table 2.

$$OAMK(i, j, k) = \begin{cases} 0 & \text{Operation } O_{ij} \text{ cannot be processed on Machine } k \\ 1/p_{ijk} & \text{Operation } O_{ij} \text{ can be processed on Machine } k \end{cases} \quad (1)$$

(3) *Operation assignment priority knowledge* (OAPK). OAPK is the accumulative knowledge of the more appropriate processing priority for the given operation. It is extracted from the global best solution of FJSSP. In this work, we define a matrix *OAPK* with size $n \times num \times rank$ for the OAPK, where n denotes the number of jobs, num denotes the maximal operation number of all jobs, and $rank$ denotes the number of priority (it can be predefined by the user). An example of OAPK matrix is showed as Table 3.

(4) *Parameters knowledge*. In order to decrease the sensitivity of parameters to the optimization, the parameters in KBACO were dynamically adjusted according to the optimization performance. In our work, if the global best solution is improved after one iteration, then this iteration will be called a successful iteration. The optimization performance can be understood as the times of the successful iteration. In the initialization phase, many parameter combinations were generated by the orthogonal design technology [35], and the optimization performance of each parameter combination is initialized as 1. In the beginning of each iteration, the parameter combination will be selected randomly according to the optimization performance. An executive example of parameters knowledge is showed as Fig. 5.

3.4. Operation assignment

Operation assignment is determining a feasible assignment of operations to machines. It is the first step to construct a feasible schedule for FJSSP. In the operation assignment phase, each ant will construct a feasible assignment based on the operation assignment machine knowledge. A simple example of constructing a feasible assignment is indicated as Table 4. The implementing flow of operation assignment is listed as follows.

Step 1 Select an arbitrary operation among all the operations which need to be assigned.

Step 2 Compute the probability of assigning the given operation to each machine based on the formula (2).

$$\Pr(i, j, k) = \frac{OAMK(i, j, k)}{\sum_{t=1}^{m} OAMK(i, j, t)} \quad (2)$$

where $\Pr(i,j,k)$ denotes the probability of assigning the operation O_{ij} to the machine k, $OAMK(i,j,k)$denotes the operation assignment machine knowledge of assigning the operation O_{ij} to the machine k.

Step 3 Select a machine randomly according to above probability, and assign the given operation to this selected machine.

Step 4 Repeat *step 1* to *step 3* till all the operations were assigned to the appropriate machines.

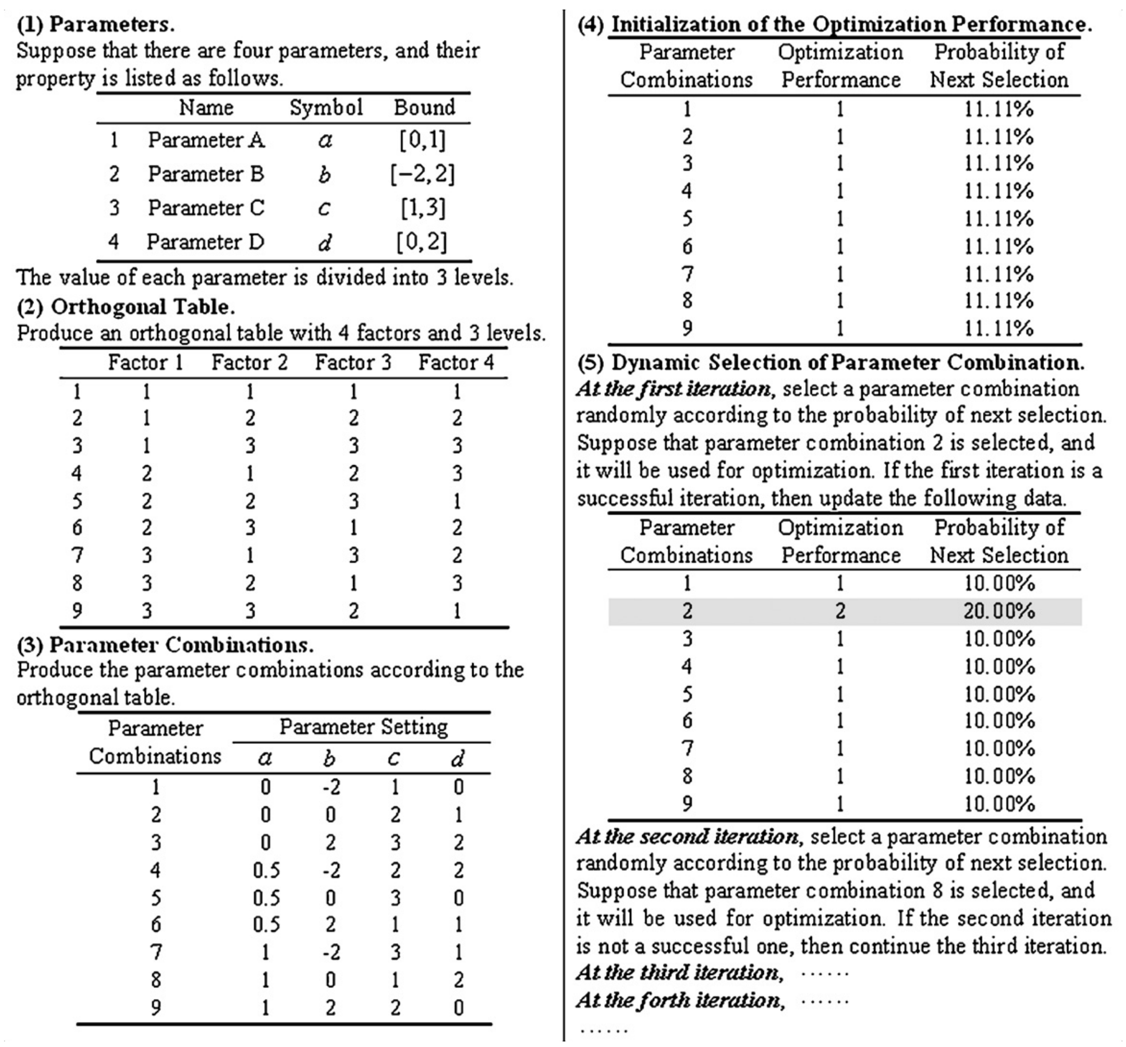
3.5. Operation sequencing

Operation sequencing is sequencing of operations on each machines. It is the second step to construct a feasible schedule for FJSSP. In the operation sequencing phase, each ant will construct

Table 4
A simple example of constructing a feasible assignment.

Oper	OAMK			Probability			Random number	Resulting machine[a]
	M 1	M 2	M 3	M 1	M 2	M 3		
O_{11}	0.8	0.5	0	0.62	0.38	0	0.9501	M 2
O_{12}	0	0.6	0.3	0	0.67	0.33	0.2311	M 2
O_{21}	0.5	0	0.6	0.45	0	0.55	0.6068	M 3
O_{22}	0.3	0.6	0	0.33	0.67	0	0.4859	M 2
O_{23}	0	0.7	0.4	0	0.64	0.36	0.8913	M 3

[a] The roulette approach was applied to select the resulting machine.



(1) Parameters.
Suppose that there are four parameters, and their property is listed as follows.

	Name	Symbol	Bound
1	Parameter A	a	[0,1]
2	Parameter B	b	[−2, 2]
3	Parameter C	c	[1,3]
4	Parameter D	d	[0, 2]

The value of each parameter is divided into 3 levels.

(2) Orthogonal Table.
Produce an orthogonal table with 4 factors and 3 levels.

	Factor 1	Factor 2	Factor 3	Factor 4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

(3) Parameter Combinations.
Produce the parameter combinations according to the orthogonal table.

Parameter Combinations	Parameter Setting			
	a	b	c	d
1	0	-2	1	0
2	0	0	2	1
3	0	2	3	2
4	0.5	-2	2	2
5	0.5	0	3	0
6	0.5	2	1	1
7	1	-2	3	1
8	1	0	1	2
9	1	2	2	0

(4) Initialization of the Optimization Performance.

Parameter Combinations	Optimization Performance	Probability of Next Selection
1	1	11.11%
2	1	11.11%
3	1	11.11%
4	1	11.11%
5	1	11.11%
6	1	11.11%
7	1	11.11%
8	1	11.11%
9	1	11.11%

(5) Dynamic Selection of Parameter Combination.
At the first iteration, select a parameter combination randomly according to the probability of next selection. Suppose that parameter combination 2 is selected, and it will be used for optimization. If the first iteration is a successful iteration, then update the following data.

Parameter Combinations	Optimization Performance	Probability of Next Selection
1	1	10.00%
2	2	20.00%
3	1	10.00%
4	1	10.00%
5	1	10.00%
6	1	10.00%
7	1	10.00%
8	1	10.00%
9	1	10.00%

At the second iteration, select a parameter combination randomly according to the probability of next selection. Suppose that parameter combination 8 is selected, and it will be used for optimization. If the second iteration is not a successful one, then continue the third iteration.
At the third iteration, ……
At the forth iteration, ……
……

Fig. 5. An implementation example of parameters knowledge.

a feasible schedule based on the operation assignment priority knowledge. The implementing flow of operation sequencing is listed as follows.

Step 1 Establish a comprehensive processing mechanism for the operation sequencing. Our processing mechanism is based on the simulation advance mechanism. This mechanism can be summarized as follows. If there are several idle machines, then it selects next operation from an allowed set (see *step 2*) based on the given rule (see *step 3*) for these machines. If all the machines are busy, then it advances the simulation time till one or several machines are idle. The aforementioned process will be executed till all the operations were finished.

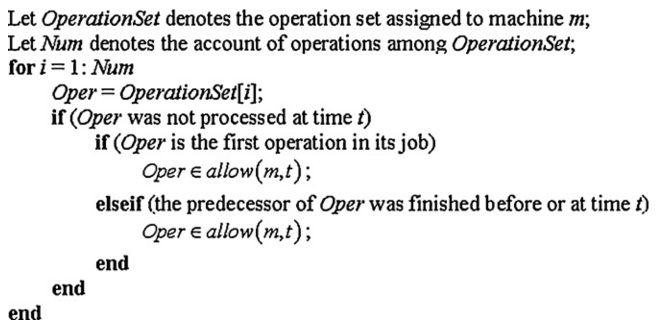
Step 2 Determine the allowed processing operation set. In this work, let *allow*(m, t) denotes the allowed processing operation set for machine m at time t, and it can be defined as Fig. 6.

Step 3 Select an operation from the allowed processing operation set based on the probability obtained from formula (3).

$$\forall O_{ij} \in allow(m,t),$$
$$\Pr(i,j,m,t) = \frac{[OAPK(i,j,m)]^{\alpha} \times [1/p_{i,j,m}]^{\beta}}{\sum_{O_{ij} \in allow(m,t)} \{[OAPK(i,j,m)]^{\alpha} \times [1/p_{i,j,m}]^{\beta}\}} \tag{3}$$

where $\Pr(i, j, m, t)$ denotes the probability of processing the operation O_{ij} on machine m at time t, *OAPK*(i, j, m) denotes the operation assignment priority knowledge of processing operation O_{ij} on machine m, α and β denotes the heuristic weight of OAPK and processing time, respectively.

Step 4 Construct a feasible schedule. According to the proposed processing mechanism of *step 1*, repeat *step 2* and *step 3* till all the operations were processed.



```
Let OperationSet denotes the operation set assigned to machine m;
Let Num denotes the account of operations among OperationSet;
for i = 1: Num
    Oper = OperationSet[i];
    if (Oper was not processed at time t)
        if (Oper is the first operation in its job)
            $Oper \in allow(m,t)$;
        elseif (the predecessor of Oper was finished before or at time t)
            $Oper \in allow(m,t)$;
        end
    end
end
```

Fig. 6. Definition of the allowed processing operation set.

A simple example of constructing a feasible schedule is indicated as Fig. 7.

The machine alternatives and processing times can see Table 1.
The operation assignment priority knowledge can refer to Table II.
The known operation assignment is listed as follows.

$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$
M1	M2	M1	M2	M3

$\alpha = 2, \beta = 3$.

(1) ***SimTime***=0

Machine 1 is idle.	Machine 2 is idle.	Machine 3 is idle.
$allow(1,0)=\{O_{11},O_{21}\}$;	$allow(2,0)=\{\ \}$;	$allow(3,0)=\{\ \}$;
$\Pr(1,1,1,0)=\left[0.9^2\times(1/10)^3\right]/\left[0.9^2\times(1/10)^3+0.6^2\times(1/20)^3\right]=0.95$		
$\Pr(2,1,1,0)=\left[0.6^2\times(1/20)^3\right]/\left[0.9^2\times(1/10)^3+0.6^2\times(1/20)^3\right]=0.05$		
Generate a random number, *Rand*=0.6895;		
Operation O_{11} is processed on machine 1 at time 0 (*roulette*).		

(2)***SimTime***=10

Machine 1 is idle.	Machine 2 is idle.	Machine 3 is idle.
$allow(1,10)=\{O_{21}\}$;	$allow(2,10)=\{O_{12}\}$;	$allow(3,10)=\{\ \}$;
$\Pr(2,1,1,10)=1$;	$\Pr(1,2,2,10)=1$;	
Operation O_{21} is processed on machine 1 at time 10. Operation O_{12} is processed on machine 2 at time 10.		

(3)***SimTime***=22

Machine 1 is busy.	Machine 2 is idle.	Machine 3 is idle.
	$allow(2,22)=\{\ \}$;	$allow(3,22)=\{\ \}$;

(4)***SimTime***=30

Machine 1 is idle.	Machine 2 is idle.	Machine 3 is idle.
$allow(1,30)=\{\ \}$	$allow(2,30)=\{O_{22}\}$;	$allow(3,30)=\{\ \}$;
	$\Pr(2,2,2,30)=1$;	
Operation O_{22} is processed on machine 2 at time 30.		

(5)***SimTime***=48

Machine 1 is idle.	Machine 2 is idle.	Machine 3 is idle.
$allow(1,48)=\{\ \}$	$allow(2,48)=\{\ \}$;	$allow(3,48)=\{O_{23}\}$;
		$\Pr(2,3,3,48)=1$
Operation O_{23} is processed on machine 3 at time 48.		

The resulting feasible schedule is shown as follows.

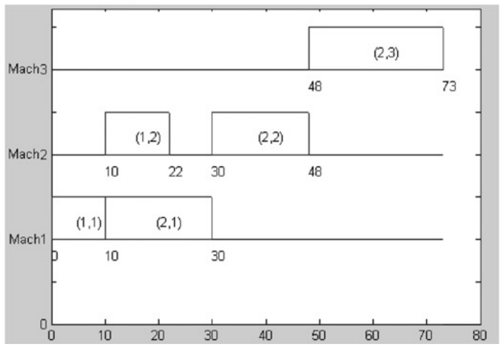


Fig. 7. A simple example of constructing a feasible schedule.

3.6. Schedule improving

In order to enhance the performance of the best schedule, the schedule improving operation was executed after each iteration based on the elite solution knowledge. That is, the solution in elite solution set will be improved by inter-learning operation. The inter-learning operator recombines the component of two solutions and produces new solutions such that the resultant solutions inherit a set of building blocks from each former solution. The implementing flow of inter-learning operation is listed as follows.

Solution=	Operation Order					Machine Selection				
	1	1	2	2	2	1	2	1	2	3

Fig. 8. Structure of the solution representation.

Solution A:	1	2	1	2	2
Solution B:	2	2	2	1	1
Two random positions:		*		*	
The substring from solution A:	1	2	1	2	2

The characters ofeach corresponding position in the substring are then found and deleted from solution B

The remainder of solution B:			2	1	

The substring in solution Ais inserted to solution B at the same position to create a new solution

Resultant solution1:	2	2	1	2	1

Another new solution can be obtained by the same way.

Resultant solution2:	2	2	2	1	1

Fig. 9. Inter-learning operation example of the operation order part.

(1) Solution representation

The solution representation has two components: operation order and machine selection (Fig. 8). More details of this representation can refer to the Ref. [20].

- *Operation order component*. The operation order representation from [20] was adopted in this work. Consider the problem in Fig. 1, job 1 (J_1) has 2 operations and job 2 (J_2) has 3 operations. One possible schedule could be ($O_{11}, O_{12}, O_{21}, O_{22}, O_{23}$), an individual is obtained from this schedule by replacing each operation by the corresponding job index (see Fig. 8). By reading the data from left to right and increasing operation index of each job, a feasible solution is always obtained.
- *Machine selection component*. We use an array of values to present machine selection. For the problem in Fig. 1, one possible encoding of the machine selection part is shown in Fig. 8.

(2) Inter-learning operation

In this work, inter-learning is applied on each part of the solution.

- *Operation order part*: Adopts the concepts of the two-point crossover operation, we propose a two-point inter-learning operation in this paper. Two-point inter-learning operator calls for two points to be selected on the existing solution organism strings. Everything between the two points is swapped between the existing solution organisms, rendering two resultant solution organisms. A simple inter-learning operation example of the operation order part is presented in Fig. 9.
- *Machine selection part*: Two partial parts of the solutions between the two random positions are exchanged. A simple inter-learning operation example of the machine selection part is presented in Fig. 10.

Solution A:	1	2	1	2	3
Solution B:	1	3	2	1	2
Two Random Positions:		*		*	
Resultant solution A:	1	3	2	1	3
Resultant solution B:	1	2	1	2	2

Fig. 10. Inter-learning operation example of the machine selection part.

Table 5
The scheme for these 31 FJSSP instances.

SN	Job	Mach	A-Oper	T-Oper	Remark	SN	Job	Mach	A-Oper	T-Oper	Remark
Case 1	4	5	3	12		Case 16	50	10	10	500	P-FJSP
Case 2	8	8	3.4	27		Case 17	50	10	10	500	T-FJSP
Case 3	10	7	3	29		Case 18	50	10	20	1000	P-FJSP
Case 4	10	10	3	30		Case 19	50	10	20	1000	T-FJSP
Case 5	15	10	3.7	56		Case 20	100	10	20	2000	P-FJSP
Case 6	10	6	5.5	55	MK 01	Case 21	100	10	20	2000	T-FJSP
Case 7	10	6	5.8	58	MK 02	Case 22	100	10	30	3000	P-FJSP
Case 8	15	8	10	150	MK 03	Case 23	100	10	30	3000	T-FJSP
Case 9	15	8	6	90	MK 04	Case 24	150	20	10	1500	P-FJSP
Case 10	15	4	7.1	106	MK 05	Case 25	150	20	10	1500	T-FJSP
Case 11	10	10	15	150	MK 06	Case 26	150	20	20	3000	P-FJSP
Case 12	20	5	5	100	MK 07	Case 27	150	20	20	3000	T-FJSP
Case 13	20	10	11.3	225	MK 08	Case 28	200	20	20	4000	P-FJSP
Case 14	20	10	12	240	MK 09	Case 29	200	20	20	4000	T-FJSP
Case 15	20	13	12	240	MK 10	Case 30	200	20	30	6000	P-FJSP
						Case 31	200	20	30	6000	T-FJSP

Please note: (1) "MK 01", "MK 02",. . ., "MK 10" denote the byname of given cases. (2) "T-FJSP" and "P-FJSP" denote the type of each FJSSP case.

3.7. Knowledge learning

(1) In knowledge learning phrase, the elite solution knowledge will be updated by the predefined numbers of best solution of every iteration.
(2) In this work, the knowledge depositing rule is only applied to the optimal schedule from the beginning of trials, and the knowledge evaporating rule is performed after every trial. Adopts the concepts of the Max-Min Ant System (MMAS) [36], knowledge on each solution component is limited to an interval $[\tau_{\min}, \tau_{\max}]$ to avoid stagnation state. MMAS is considered one of the best known ACO thanks to its ability to overcome stagnation by using an upper and a lower level of pheromone. The OAMK and OAPK will be updated by following rules.
   - Knowledge depositing rule

$$OAMK(i,j,k) = OAMK(i,j,k) \times Q_1 \tag{4}$$

$$OAPK(i,j,k) = OAPK(i,j,k) + Q_2 \tag{5}$$

   where Q_1 denotes the incremental coefficient of OAMK, Q_2 denotes the incremental level of OAPK.
   - Knowledge evaporating rule

$$OAMK(i,j,k) = \min\{\tau_{\max}, \max\{\tau_{\min}, (1-\rho)OAMK(i,j,k)\}\} \tag{6}$$

$$OAPK(i,j,k) = \min\{\tau_{\max}, \max\{\tau_{\min}, (1-\rho)OAPK(i,j,k)\}\} \tag{7}$$

   where ρ is the knowledge evaporating parameter.
(3) If the global best solution was improved, then the optimization performance of the current parameter combination will be added 1.

4. Computational results

Two sets of FJSSP instances (P-FJSSP and T-FJSSP) are used to evaluate the proposed KBACO. Test sample I includes 15 FJSSP instances taken from [20,22,23,25]. Test sample II includes 16 randomly generated FJSSP instances. The generated mechanisms for these random FJSSP instances can be listed as follows.

- The number of jobs ranges from 50 to 200.
- The number of machines ranges from 10 to 20.
- The number of operations for each job ranges from 10 to 30.
- The processing time for an operation ranges from 10 to 100 units.
- In the random P-FJSP instances, 50% machines are randomly selected to process an operation.
- The deviation of two machines to process one operation ranges from −10% to 10%.

Table 6
The parameters used in this section (I).

Parameter	ρ	α	β	Q_1	Q_2
Para-Comb 1	0.01	2	2	2	0.05
Para-Comb 2	0.01	4	4	4	0.1
Para-Comb 3	0.01	6	6	6	0.15
Para-Comb 4	0.01	8	8	8	0.2
Para-Comb 5	0.02	2	4	6	0.2
Para-Comb 6	0.02	4	6	8	0.05
Para-Comb 7	0.02	6	8	2	0.1
Para-Comb 8	0.02	8	2	4	0.15
Para-Comb 9	0.03	2	6	2	0.15
Para-Comb 10	0.03	4	8	4	0.2
Para-Comb 11	0.03	6	2	6	0.05
Para-Comb 12	0.03	8	4	8	0.1
Para-Comb 13	0.04	2	8	6	0.1
Para-Comb 14	0.04	4	2	8	0.15
Para-Comb 15	0.04	6	4	2	0.2
Para-Comb 16	0.04	8	6	4	0.05

The scheme for these 31 FJSSP instances was displayed as Table 5, where 'SN' denotes the sequence number, 'Job' denotes the number of jobs, 'Mach' denotes the number of machines, 'A-Oper' denotes the average number of operations for each job, and 'T-Oper' denotes the total number of operations in each FJSSP case. The proposed KBACO was implemented in Matlab on a Dell Precision 650 workstation with a Pentium IV 2.4G processor, and 1G RAM. The experimental results were averaged over 10 trials. The parameters used in this section can be summarized as follows (Tables 6 and 7). The termination criterion of KBACO occurs when one of the following conditions is satisfied.

(1) Maximal iteration *MI* was exhausted;

Table 7
The parameters used in this section (II).

Parameter	Value	
	Case 1–15	Case 16–31
Size[a]	200	20
Num[b]	20	10
Iter[c]	5	5
$\tau_{\min}$	0.01	0.01
$\tau_{\max}$	1	1
MI	200	200
SI	20	20

[a] *Size* denotes the number of ants.
[b] *Num* denotes the predefined numbers of best solution.
[c] *Iter* denotes the executing times of operation sequencing module at each iteration.

Table 8
Experimental results of the proposed KBACO for solving 15 FJSSP instances.

FJSSP instance	Makespan			Avg. time (min)
	Best	Avg.	SD	
Case 1	11	11	0	1.50
Case 2	14	14.3	0.46	6.47
Case 3	11	11	0	6.61
Case 4	7	7.4	0.48	11.07
Case 5	11	11.3	0.46	15.84
Case 6	39	39.8	0.43	7.82
Case 7	29	29.1	0.28	9.87
Case 8	204	204	0	75.92
Case 9	65	66.1	1.06	20.41
Case 10	173	173.8	1.08	16.22
Case 11	67	69.1	1.03	40.21
Case 12	144	145.4	1.42	28.53
Case 13	523	523	0	87.41
Case 14	311	312.2	1.81	78.36
Case 15	229	233.7	3.27	124.90

(2) The global best solution from the beginning trials was not improved in the successive *SI* iteration.

4.1. Experimental result

To these FJSSP instances presented in literatures, the final experimental results of our proposed KBACO averaged over 10 trials were displayed as Table 8. Four indexes were applied to describe the experimental results.

(1) *Best*: the best makespan among these 10 trials.
(2) *Avg.*: the average makespan of these 10 trials.
(3) *SD*: the standard deviation of the makespan obtained by 10 trials.
(4) *Avg. time*: the average computational times of these 10 trials.

The experimental results of Table 8 suggest that KBACO performed well on all fifteen FJSSP instances. It can obtain the near-optimal (optimal) solutions with a quick computational speed.

To these randomly generated FJSSP instances, the final experimental results of our proposed KBACO averaged over 10 trials were displayed as Table 9. Three methods were constructed to evaluate the contributions of particular ‘parts’ of the proposed KBACO algorithm.

Table 9
Experimental results of KBACO for solving 16 randomly generated FJSSP instances.

FJSSP instance	Optimal makespan			Avg. time (min)
	Method 1	Method 2	Method 3	
Case 16	2,811	2,789	2,719	16.31
Case 17	2,947	2,907	2,857	19.23
Case 18	5,709	5,700	5,682	38.54
Case 19	5,858	5,829	5,807	30.42
Case 20	11,058	11,013	10,995	150
Case 21	11,572	11,522	11,495	94.42
Case 22	17,465	17,147	17,112	135.68
Case 23	17,134	17,057	16,980	131.81
Case 24	4,457	4,448	4,436	155.09
Case 25	4,598	4,588	4,584	102.75
Case 26	9,067	8,770	8,730	292.44
Case 27	8,957	8,941	8,901	504.31
Case 28	11,696	11,366	11,322	722.26
Case 29	11,895	11,886	11,872	737.61
Case 30	17,755	17,627	17,574	878.14
Case 31	17,368	17,205	17,182	700.14

Table 10
Comparison results between A1–A5 and A8 in Case 1–Case 5 (optimal makespan)[a].

	Case 1	Case 2	Case 3	Case 4	Case 5
A1	–[b]	19	–	16	–
A2	–	16	–	7	–
A3	–	16	–	8	–
A4	–	15	–	7	23
A5	–	15	–	7	12
A8	11	14	11	7	11

[a] Please note, a release date for each job is considered in Ref. [25]. In other references, they did not consider the release date for each job.
[b] “–” denotes that this case was not tested in the given reference.

(1) *Method 1*: KBACO algorithm without schedule improving module and parameter learning module. It is to say, both the parameters knowledge and the elite solution knowledge was not used in method 1.
(2) *Method 2*: KBACO algorithm without schedule improving module, that is, the elite solution knowledge was not used in method 2.
(3) *Method 3*: KBACO algorithm.

Following indexes were applied to describe the experimental results in this part.

(1) *Optimal makespan*: the best makespan among 10 trials by each method.
(2) *Avg. time*: the average computational times of these 30 trials.

From the experimental results of Table 9, we can see that all the optimal makespan obtained by method 2 is better than that obtained by method 1. It is to say, the parameter learning module (parameters knowledge) is efficacious among the proposed KBACO algorithm. At the same time, we can see that all the optimal makespan obtained by method 3 is more excellent than that obtained by method 2. That is, the schedule improving module (elite solution knowledge) is efficacious among the proposed KBACO algorithm. Taking one with another, the proposed KBACO algorithm performed well on all sixteen randomly generated FJSSP instances.

4.2. Comparative studies

KBACO is compared with following published algorithms in the comparative studies. The final comparative results are displayed as Tables 10 and 11.

(1) *A1*: Temporal Decomposition [22];
(2) *A2*: Controlled Genetic Algorithm (CGA) [22];
(3) *A3*: Approach by Localization (AL) [22];
(4) *A4*: AL + CGA [22];
(5) *A5*: PSO + SA [21];
(6) *A6*: Tabu Search [23];

Table 11
Comparison results between A6, A7 and A8 in Case 6–Case 15 (optimal makespan).

	A6	A7	A8
Case 6	42	40	39
Case 7	32	29	29
Case 8	204	–	204
Case 9	81	67	65
Case 10	186	176	173
Case 11	86	67	67
Case 12	157	147	144
Case 13	523	523	523
Case 14	369	320	311
Case 15	296	229	229

(7) *A7*: GENACE [20];
(8) *A8*: KBACO (this paper).

Table 10 suggests that the optimal makespan of KBACO is superior among the six approaches for Case 1 to Case 5. Table 11 suggests that the optimal makespan of KBACO is superior among the three approaches for Case 6 to Case 15. To summarize, the results demonstrate that KBACO outperforms these eight published algorithms, and is competent for the Flexible Job Shop Scheduling Problems.

Before analyzing the computational time, it must be noted that the evaluations were made using different computers and languages (compilers) to those on which the comparison algorithms were run. The performance of different computers is hard to compare, and the running speed of different compilers is very different. Also, some of Refs. [21,22] have not reported the computational time in the experimental results, it makes very difficult for us to compare the computational time. For these reason, we cannot compare the computational time of different methods.

5. Conclusions

The contribution of this paper can be summarized as follows. It proposes a KBACO algorithm for the FJSSP. The performance of KBACO was largely improved by integrating the ACO model with knowledge model. In KBACO, some available knowledge was learned from the optimization of ACO by the knowledge model, at the same time, the existing knowledge was applied to guide the current heuristic searching of ACO. Final experimental results indicate that the proposed KBACO algorithm outperforms some published methods in the quality of schedules.

In the proposed KBACO, the knowledge model is simply a memory keeping good features from previous iteration. For this reason, we intend to focus on enhancing KBACO via explorations in machine learning so as to improve the efficiency of the knowledge model. Also, the extension to multi-objective FJSSP will be investigated in the near future.

Acknowledgments

The authors thank the anonymous reviewers for their invaluable comments and constructive suggestions at first, thank the editor in chief, associate editor and editors for their diligent working and helpful suggestions at second, thank the College of Information System and Management, National University of Defense Technology, China, for providing the excellent environment and the experiment facility.

References

[1] M.R. Garey, D.S. Johnson, R. Sethi, The complexity of flow shop and job-shop scheduling, Mathematics of Operations Research 1 (2) (1996) 117–129.
[2] M. Kolonko, Some new results on Simulated Annealing applied to the Job Shop Scheduling Problem, European Journal of Operational Research 113 (1) (1999) 123–136.
[3] F. Pezzella, E. Merelli, A Tabu Search method guided by shifting bottleneck for the Job Shop Scheduling Problem, European Journal of Operational Research 120 (2) (2000) 297–310.
[4] J.F. Goncalves, et al., A hybrid genetic algorithm for the Job Shop Scheduling Problem, European Journal of Operational Research 167 (1) (2005) 77–95.
[5] K.L. Huang, C.J. Liao, Ant colony optimization combined with taboo search for the job shop scheduling problem, Computers and Operations Research 35 (4) (2008) 1030–1046.
[6] D.J. Fonseca, D. Navaresse, Artificial neural networks for job shop simulation, Advanced Engineering Informatics 16 (4) (2002) 241–246.
[7] I.T. Tanev, T. Uozumi, Y. Morotome, Hybrid evolutionary algorithm-based real-world Flexible Job Shop Scheduling Problem: application service provider approach, Applied Soft Computing 5 (1) (2004) 87–100.
[8] H. Chen, P.B. Luh, An alternative framework to Lagrangian relaxation approach for Job Shop Scheduling, European Journal of Operational Research 149 (3) (2003) 499–512.
[9] W.Q. Huanh, A.H. Yin, An improved shifting bottleneck procedure for the Job Shop Scheduling Problem, Computers & Operations Research 31 (12) (2004) 2093–2110.
[10] K. Jansen, M. Mastrolilli, R. Solis-Oba, Approximation schemes for Job Shop Scheduling Problems with controllable processing times, European Journal of Operational Research 167 (2) (2005) 297–319.
[11] P. Bruker, R. Schlie, Job-shop scheduling with multi-purpose machines, Computing 45 (4) (1990) 369–375.
[12] L. Wang, A hybrid genetic algorithm-neural network strategy for simulation optimization, Applied Mathematics and Computation 170 (2) (2005) 1329–1343.
[13] R.G. Reynolds, An introduction to cultural algorithms, in: Proceedings of the Third Annual Conference on Evolutionary Programming, River Edge, NJ, World Scientific, Singapore, 1994, pp. 131–139.
[14] C.A. Santosa, J.A. Spimb, A. Garcia, Mathematical modeling and optimization strategies (genetic algorithm and knowledge base) applied to the continuous casting of steel, Engineering Applications of Artificial Intelligence 16 (5–6) (2003) 511–527.
[15] P. Vachhani, R. Rengaswamy, V. Venkatasubramanian, A framework for integrating diagnostic knowledge with nonlinear optimization for data reconciliation and parameter estimation in dynamic systems, Chemical Engineering Science 56 (6) (2001) 2133–2148.
[16] C.J. Chung, R.G. Reynolds, A testbed for solving optimization problems using cultural algorithm, in: Proceedings of the Fifth Annual Conference on Evolutionary Programming, MIT Press, Cambridge, 1996, pp. 225–236.
[17] J. Branke, Memory-enhanced evolutionary algorithms for dynamic optimization problems, in: Proceedings of the Congress on Evolutionary Computation, IEEE Press, Piscataway, 1999, pp. 1875–1882.
[18] S.J. Louis, J. McDonnell, Learning with case-injected genetic algorithms, IEEE Transactions on Evolutionary Computation 8 (4) (2004).
[19] R.S. Michalski, Learnable evolution model: evolution process guided by machine learning, Machine Learning 38 (1) (2000) 9–40.
[20] N.B. Ho, J.C. Tay, E.M.K. Lai, An effective architecture for learning and evolving flexible job-shop schedules, European Journal of Operational Research 179 (2) (2007) 316–333.
[21] W.J. Xia, Z.M. Wu, An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, Computers & Industrial Engineering 48 (2) (2005) 409–425.
[22] I. Kacem, S. Hammadi, P. Borne, Approach by localization and multi-objective evolutionary optimization for flexible job-shop scheduling problems, IEEE Transactions on Systems, Man, and Cybernetics, Part C 32 (1) (2002) 1–13.
[23] P Brandimarte, Routing and scheduling in a flexible job shop by Taboo search, Annals of Operations Research 41 (1) (1993) 157–183.
[24] L.F. Tung Lf, L. Li, R. Nagi, Multi-objective scheduling for the hierarchical control of flexible manufacturing systems, The International Journal of Flexible Manufacturing Systems 11 (4) (1999) 379–409.
[25] I. Kacem, S. Hammadi, P. Borne, Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic, Mathematics and Computers in Simulation 60 (3) (2002) 245–276.
[26] E. Hurink, B. Jurisch, M. Thole, Tabu Search for the Job Shop Scheduling Problem with multi-purpose machines, Operations Research Spektrum 15 (4) (1994) 205–215.
[27] S. Dauzere-Peres, J. Paulli, An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using Tabu Search, Annals of Operations Research 70 (3) (1997) 281–306.
[28] M. Mastrolilli, L.M. Gambardella, Effective neighborhood functions for the flexible job shop problem, Journal of Scheduling 3 (1) (2002) 3–20.
[29] S. Climer, W. Zhang, Cut-and-solve: an iteration search strategy for combinatorial optimization problems, Artificial Intelligence 170 (8–9) (2006) 714–738.
[30] L. Wang, D.Z. Zheng, An effective hybrid optimization strategy for job-shop scheduling problems, Computers and Operations Research 28 (6) (2001) 585–596.
[31] S.M. Yang, D.G. Shao, Y.J. Luo, A novel evolution strategy for multiobjective optimization problem, Applied Mathematics and Computation 170 (2) (2005) 850–873.
[32] A. Kusiak, Process planning: a knowledge-based and optimization perspective, IEEE Transactions on Robotics and Automation 7 (3) (1991) 257–266.
[33] A. Dutta, S. Mitra, Integrating heuristic knowledge and optimization models for communication network design, IEEE Transactions on Knowledge and Data Engineering 5 (6) (1993) 999–1017.
[34] R.G. Reynolds, S. Zhu, Knowledge-based function optimization using fuzzy cultural algorithms with evolutionary programming, IEEE Transactions on Systems, Man and Cybernetics-Part B: Cybernetics 31 (1) (2001) 1–18.
[35] Y.W. Leung, F. Wang, An orthogonal genetic algorithm with quantization for global numerical optimization, IEEE Transactions on Evolutionary Computation 5 (1) (2001) 41–53.
[36] T. Stuzle, H. Hoos, MAX-MIN Ant System, Journal of Future Generation Computer Systems 16 (8) (2001) 889–914.