Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Computers & Industrial Engineering 48 (2005) 409–425

computers & industrial engineering

www.elsevier.com/locate/dsw

An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems[☆]

Weijun Xia[*], Zhiming Wu

CIMS Laboratory, Department of Automation, 1416# Haoran High Technology Building, 1954# Huashan Road, Shanghai Jiao Tong University, Shanghai 200030, People's Republic of China

Abstract

Scheduling for the flexible job-shop is very important in both fields of production management and combinatorial optimization. However, it is quite difficult to achieve an optimal solution to this problem with traditional optimization approaches owing to the high computational complexity. The combining of several optimization criteria induces additional complexity and new problems. Particle swarm optimization is an evolutionary computation technique mimicking the behavior of flying birds and their means of information exchange. It combines local search (by self experience) and global search (by neighboring experience), possessing high search efficiency. Simulated annealing (SA) as a local search algorithm employs certain probability to avoid becoming trapped in a local optimum and has been proved to be effective for a variety of situations, including scheduling and sequencing. By reasonably hybridizing these two methodologies, we develop an easily implemented hybrid approach for the multi-objective flexible job-shop scheduling problem (FJSP). The results obtained from the computational study have shown that the proposed algorithm is a viable and effective approach for the multi-objective FJSP, especially for problems on a large scale.
© 2005 Elsevier Ltd. All rights reserved.

Keywords: Flexible job-shop scheduling; Particle swarm optimization; Simulated annealing; Multi-objective optimization; Combinatorial optimization

1. Introduction

Scheduling problems occur in all the economic domains, from computer engineering to manufacturing techniques. Most scheduling problems are complex combinatorial optimization problems

[☆] This manuscript was processed by Area Editor Maged Dessouky.
[*] Corresponding author. Tel.: +86 21 6293 2070/3428 24.
E-mail addresses: xwjemail@yahoo.com (W. Xia), weijunxia@sjtu.edu.cn (W. Xia), ziminwu@sjtu.edu.cn (Z. Wu).

0360-8352/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2005.01.018

and very difficult to solve. The job-shop scheduling problem (JSP) is a branch of production scheduling, which is among the hardest combinatorial optimization problems. The classical JSP consists in scheduling a set of jobs on a set of machines with the objective to minimize a certain criterion, subject to the constraint that each job has a specified processing order through all machines which are fixed and known in advance. It is well known that this problem is NP-hard (Garey, Johnson, & Sethi, 1976). That means with current algorithms even moderately sized problems cannot be solved to guaranteed optimality. Many different approaches have been applied to JSP.

Although an optimal solution algorithm for the classical JSP has not been developed, there is a trend in the research domain to solve a much more complex version of the problem. The problem is referred to as the flexible job-shop scheduling problem (FJSP). FJSP is an extension of the classical JSP which allows an operation to be processed by any machine from a given set. It incorporates all of the difficulties and complexities of its predecessor JSP and is more complex than JSP because of the addition need to determine the assignment of operations to machines. The scheduling problem of a FJSP consists of a routing sub-problem, that is, assigning each operation to a machine out of a set of capable machines and the scheduling sub-problem, which consists of sequencing the assigned operations on all machines in order to obtain a feasible schedule minimizing a predefined objective function. The FJSP mainly presents two difficulties. The first one is to assign each operation to a machine, and the second one is to schedule these operations in order to make a predefined objective minimal.

The literature of FJSP is considerably sparser than the literature of JSP. Bruker and Schlie (1990) were among the first to address this problem. They develop a polynomial algorithm for solving the flexible job-shop problem with two jobs. For solving the realistic case with more than two jobs, two types of approaches have been used: hierarchical approaches and integrated approaches. In hierarchical approaches assignment of operations to machines and the sequencing of operations on the resources or machines are treated separately, i.e. assignment and sequencing are considered independently, whereas in integrated approaches, assignment and sequencing are not differentiated. Hierarchical approaches are based on the idea of decomposing the original problem in order to reduce its complexity. This type of approach is natural for FJSP since the routing and the scheduling sub-problems can be separated. Brandimarte (1993) was the first to use this decomposition for the FJSP. He solved the routing sub-problem using some existing dispatching rules and then focused on the scheduling sub-problem, which is solved using a taboo search heuristic. Tung, Li, and Nagi (1999) developed a similar approach for scheduling a flexible manufacturing system. Recently, Kacem, Hammadi, and Borne (2002a,b) proposed a genetic algorithm controlled by the assigned model which is generated by the approach of localization (AL). And they used it to mono-objective and multi-objective FJSP. Integrated approaches were used by considering assignment and scheduling at the same time. Hurink, Jurisch, and Thole (1994) proposed a taboo search heuristic in which reassignment and rescheduling are considered as two different types of moves. The integrated approach which had been presented by Dauzere-Peres and Paulli (1997) was defined a neighborhood structure for the problem where there is no distinction between reassigning and resequencing an operation. A taboo search procedure is proposed based on the neighborhood structure. Mastrolilli and Gambardella (2002) improved Dauzere-Peres' taboo search techniques and presented two neighborhood functions. Most researchers were interested in applying taboo search techniques and genetic algorithms to FJSP in the past.

In this research paper a practical hierarchical solution approach is proposed for solving multi-objective FJSP. The proposed approach makes use of particle swarm optimization (PSO) to assign operations on machines and simulated annealing (SA) algorithm to schedule operations on each machine. The considered objective is to minimize makespan (maximal completion time), the total workload of machines, and the workload of the critical machine.

The remainder of the paper is organized as follows: the notation and problem description are introduced in Section 2. Section 3 describes standard PSO algorithm and how to apply it to multi-objective FJSP. Section 4 focuses on basic ingredients of SA, introducing some rules of parameters selection. The hybrid optimization algorithm is described in Section 5. In Section 6, computational experiments performed with the new hybrid optimization approach for three representative instances of FJSP are reported followed by the comparison to other heuristic methods. Some concluding remarks are made in Section 7.

2. Problem formulation

The FJSP problem may be formulated as follows. There is a set of n jobs that plan to process on m machines. The set machines is noted $M, M=\{M_1, M_2, \ldots M_m\}$. Each job j consists of a sequence of n_j operations (routing) $O_{j,1}, O_{j,2}, \ldots O_{j,nj}$. Each routing has to be performed to complete a job. The execution of each operation i of a job j (noted $O_{j,i}$) requires one machine out of a set of given machines called $M_{j,i} \subseteq M$. The problem is thus to both determine an assignment and a sequence of the operations on all machines that minimize some criteria. In this study, the following criteria are to be minimized:

F_1 makespan or maximal completion time of machines.
F_2 total workload of the machines, which represents the total working time of all machines.
F_3 critical machine workload, that is the machine with the biggest workload.

Many approaches have been developed in the domain of multi-objective meta-heuristic optimization. We focus our presentation on evolutionary approaches that can be classified into three types (Hsu, Dupas, Jolly, & Goncalves, 2002):

(i) The transformation towards a mono-objective problem consists of combining the different objectives into a weighted sum.
(ii) The non-Pareto approach utilizes operators for processing the different objectives in a separated way.
(iii) The Pareto approach is directly based on the Pareto optimization concept. It aims at satisfy two goals: first, converge to the Pareto front and also obtain diversified solutions scattered all over the Pareto front.

The objective function in this paper is based on the first type. The weighted sum of the above three objective values (F_1, F_2 and F_3) is taken as the objective function ($F(c)$).

Hypotheses considered in this paper are the following:

- Setting up times of machines and move times between operations are negligible,
- Machines are independent from each other,
- Jobs are independent from each other,
- At a given time, a machine can only execute one operation. It becomes available to other operations only if the operation which is processing is completed.
- There are no precedence constraints among the operations of different jobs.

3. Assignment algorithm: particle swarm optimization

Particle swarm optimization (PSO) is an evolutionary computation technique proposed by Kennedy and Eberhart in 1995 (Eberhart & Kennedy, 1995; Kennedy & Eberhart, 1995). The particle swarm concept was motivated from the simulation of social behavior. The original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock. The PSO algorithm mimics the behavior of flying birds and their means of information exchange to solve optimization problems. PSO has been introduced as an optimization technique in real-number spaces. But many optimization problems are set in a discrete space. Typical examples include problems that require ordering and route planning, such as in scheduling and routing problems. In this paper we introduce a method of converting domain to discrete domain for PSO and use it to assign operations to machines.

3.1. Standard PSO algorithm

PSO is an evolutionary algorithm which the system is initialized with a population (named swarm in PSO) of random solutions. Each individual or potential solution, named particle, flies in the D-dimensional problem space with a velocity which is dynamically adjusted according to the flying experiences of its own and its colleagues. During the past years, researchers have explored several models of PSO algorithm. In this paper, we use the global model equations as follows (Shi & Eberhart, 1999):

$$V_{id} = W * V_{id} + C1 * Rand() * (p_{id} - X_{id}) + C2 * rand() * (P_{gd} - X_{id}), \tag{1a}$$

and

$$X_{id} = X_{id} + V_{id}, \tag{1b}$$

where V_{id}, called the velocity for particle i, represents the distance to be travelled by this particle from its current position, X_{id} represents the particle position, P_{id} which is also called *pbest* (local best solution), represents ith particle's best previous position, and P_{gd}, which is also called *gbest* (global best solution), represents the best position among all particles in the swarm. W is inertial weight. It regulates the trade-off between the global exploration and local exploitation abilities of the swarm. The acceleration constants *C1* and *C2* represent the weight of the stochastic acceleration terms that pull each particle toward *pbest* and *gbest* positions. *Rand()* and *rand()* are two random functions in the range [0,1].

For Eq. (1a), the first part represents the inertia of previous velocity. The second part is the 'cognition' part, which represents the private thinking by itself. The third part is the 'social' part, which represents the cooperation among the particles (Kennedy, 1997). The process of implementing the PSO algorithm is as follows:

(1) Initialize a swarm of particles with random positions and velocities in the D-dimensional problem space.
(2) For each particle, evaluate the desired optimization fitness function.
(3) Compare particle's fitness value with particle's *pbest*. If current value is better than *pbest*, then set *pbest* value equal to the current value, and the *pbest* position equal to the current position in D-dimensional space.

(4) Compare fitness evaluation value with the best swarm's fitness obtained so far. If current value is better than *gbest*, then reset *gbest* to the current particle's fitness value.
(5) Change the velocity and position of the particle according to Eqs. (1a) and (1b) respectively.
(6) Loop to step (2) until a termination criterion is met, usually a sufficiently good fitness or a specified number of generations.

In PSO, each particle of the swarm shares mutual information globally and benefits from the discoveries and previous experiences of all other colleagues during the search process. PSO requires only primitive and simple mathematical operators, and is computationally inexpensive in terms of both memory requirements and time.

3.2. Encoding

The most important issue in applying PSO successfully to FJSP is to develop an effective 'problem mapping' and 'solution generation' mechanism. If these two mechanisms are devised successfully, it is possible to find good solutions for a given optimization problem in acceptable time. How to encode a schedule to a search solution (i.e. finding a suitable mapping between problem solution and PSO particle)? Firstly, we sequence the capable machines of an operation according to the increasing order of processing time. If one machine's processing time is equal to another, the lower order number of machine has priority. After this course, we get different priority levels for all machines which process the same operation. Then the particle position can be generated stochastically according to the order of operations of different jobs. Because of a different priority level corresponding to a different machine, a particle position is corresponding to a machine assignment of all operations.

To explain this approach, we choose the following example (with total flexibility, i.e. each machine can be selected for an operation.) in Table 1.

The problem is to execute three jobs on four machines. The left side of the table is original data, including jobs, operations, and processing times on different machines. On the right is the order of priority (i.e. priority level, 1, 2, 3, or 4, and priority $1>2>3>4$) of machines corresponding to each operation. The particle's position is a series of priority levels of assigned machines according to the order of operations. A stochastic particle position is represented in Fig. 1.

In general, initial particles' positions and initial particles' velocities in the swarm are generated at random. According to this approach to generate initial particles' positions, search space can be dwindled

Table 1
Example problem

		Machine				Priority order			
		M_1	M_2	M_3	M_4	1	2	3	4
J1	$O_{1,1}$	2	3	4	1	M_4	M_1	M_2	M_3
	$O_{1,2}$	3	1	8	2	M_2	M_4	M_1	M_3
J2	$O_{2,1}$	1	4	1	2	M_1	M_3	M_4	M_2
	$O_{2,2}$	5	3	2	9	M_3	M_2	M_1	M_4
	$O_{2,3}$	3	1	1	4	M_2	M_3	M_1	M_4
J3	$O_{3,1}$	7	6	3	5	M_3	M_4	M_2	M_1
	$O_{3,2}$	4	5	6	2	M_4	M_1	M_2	M_3

	Job1		Job2			Job3	
Operation	$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$	$O_{3,1}$	$O_{3,2}$
Particle Position	2	1	3	2	2	4	4
	↓	↓	↓	↓	↓	↓	↓
Processing Machine	M_1	M_2	M_4	M_2	M_3	M_1	M_3

Fig. 1. A stochastic particle position representation.

easily and this helps to improve the solution quality as well as search speed. For example, in Table 1 we define priority level less than 4, then the machine on the right of broken line cannot be selected for an operation, i.e. M_3 cannot be selected for $O_{1,1}$ and $O_{1,2}$, and M_2 cannot be selected for $O_{2,1}$, etc.

Each particle's position value represents priority level for each operation. So every bit of a particle's position should be integer. But after computation by Eqs. (1a) and (1b), some bits of a particle's position may appear real values such as 2.638, etc. It is meaningless for priority level. Therefore, in the algorithm we usually round off the real optimum value to its nearest integer number. By this way, we convert a continuous optimization problem to a discrete optimization problem.

3.3. Setting parameters

In Eq. (1a), inertial weight (W) is an important parameter to optimizing ability of PSO algorithm. A large inertial weight facilities searching new areas while a small inertial weight facilities fine-searching in the current search areas. Suitable selection of the inertial weight provides a balance between global exploration and local exploitation, and results in less iteration to find a sufficiently good solution on the average. Therefore, consider by linearly decreasing the inertial weight from a relatively large value to a relatively small value through the course of the PSO run, the PSO tends to have more global search ability at the beginning of the run while having more local search ability near the end of the run. For all computational experiments in this paper, the inertial weight is set to the following equation.

$$W = W_{\max} - \frac{W_{\max} - W_{\min}}{\text{iter}_{\max}}\text{iter}, \tag{2}$$

where

$W_{\max}$	initial value of weighting coefficient,
$W_{\min}$	final value of weighting coefficient,
$\text{iter}_{\max}$	maximum number of iterations or generation,
iter	current iteration or generation number.

The acceleration constants *C1* and *C2* in Eq. (1a) adjust the amount of 'tension' in the PSO system. Low values allow particles to roam far from target regions before being tugged back, while high values result in abrupt movement toward, or past, target regions (Eberhart & Shi, 2001).

By computation of Eqs. (1a) and (1b), the absolute value of V_{id} and X_{id} may be great. So the particle may overshoot the problem space. Therefore, V_{id} and X_{id} should be limited to maximum velocity $V_{\max}$ and maximum position $X_{\max}$, which are two parameters specified by the user. $V_{\max}$ serves as a constraint to control the global exploration ability of a particle swarm. In this paper, the maximum velocity $V_{\max}$ is set to maximal value of priority level (mpl), i.e. V_{id} is a value in the range $[-\text{mpl}, \text{mpl}]$. The maximum position $X_{\max}$ is also set to mpl. Because X_{id} represents priority level, X_{id} must be a positive integer. So X_{id} is an integer value in the range [1, mpl].

3.4. Fitness function

Fitness is used as the performance evaluation of particles in the swarm. Fitness is usually represented with a function $f{:}S\rightarrow R^{+}$ (S is the set of candidate schedules, and R^{+} is the set of positive real values). Mapping an original objective function value to a fitness value that represents relative superiority of particles is a feature of fitness function. The objective function ($F(c)$) can be represented as follows according to the description in Section 2.

$$F(c) = w_1 \times F_1(c) + w_2 \times F_2(c) + w_3 \times F_3(c).$$

We define the fitness function (fit(c)) is equal to the objective function, i.e.

$$\text{fit}(c) = F(c).$$

The objective is to minimize $F(c)$, i.e. fit(c). So particle with the lowest fitness will be superior to other particles and should be reserved in search process.

4. Scheduling algorithm: simulated annealing algorithm

Ever since its introduction, based on the joint efforts by Kirkpatrick, Gelatt, and Vecchi (1983), simulated annealing algorithm has produced good results for large combinatorial optimization problems, particularly in scheduling problems. On the one hand, the algorithm can be considered as a generalization of the well-known iterative improvement approach to combinatorial optimization problems, on the other hand, it can be viewed as an analogue of an algorithm used in statistical physics for computer simulation of the annealing of a solid to the state with minimum energy (Van Laarhoven, Aarts, and Lenstra, 1992).

SA approach can be viewed as an enhanced version of local search or iterative improvement, in which an initial solution is repeatedly improved by making small local alteration until no such alteration yields a better solution. SA randomizes this procedure in a way that allows occasional alterations that worsen the solution in an attempt to increase the probability of leaving a local optimum. The application of SA as a local search algorithm assumes a cost function (fitness function in this paper) calculated for each possible solution, a neighborhood comprising alternative solutions to a given solution, and a mechanism for generating possible solutions.

After an assignment process by PSO, the scheduling process can be completed via a SA algorithm with satisfactory scheduling results quickly.

4.1. SA algorithm

Starting from an initial solution, SA generates a new solution S' in the neighborhood of the original solution S. Then, the change of objective function values, $\Delta = f(S') - f(S)$, is calculated.

For a minimization problem, if $\Delta < 0$, the transition to the new solution is accepted. If $\Delta \geq 0$, then the transition to the new solution is accepted with probability, usually denoted by the function, $\exp(-\Delta/T)$, where T is a control parameter called the temperature. SA algorithm generally starts from a high temperature and then the temperature is gradually lowered. At each temperature, a search is carried out for a certain number of iterations, called the epoch length. When the termination condition is satisfied, the algorithm will stop.

4.2. Neighborhood solutions

In SA search algorithm, the choice of neighborhood can greatly influence algorithm performance. While choosing a rich neighborhood containing a large number of candidate solutions will increase the likelihood of finding good solutions, the computation time required to search from the available neighbors will also increase. As a simple method for generating neighborhood solutions, the pair-exchange method is used in a particle position according to the machine order in this paper. Assuming p operations assigned on a machine, the pair-exchange method is shown as follows:

$$(1 \leftrightarrow 2), (2 \leftrightarrow 3), (3 \leftrightarrow 4), \ldots (p-1 \leftrightarrow p)$$

Because the code of a particle's position is encoded according to the order of operation, it is not suitable for application in SA algorithm. The code should be switched from operation-based to machine-based. We also consider the example of Fig. 1. The switched code based machine order is shown in Fig. 2. By exchanging operations on the same machine of a particle and evaluating pair-exchange result each time, we can get satisfactory search results quickly.

4.3. Cooling schedule

SA process can be controlled by the cooling schedule. In general, the cooling schedule is specified by several parameters and methods, namely the initial temperature T_0, the epoch length L, the rule designated how to lower the temperature, and the termination condition.

A proper initial temperature should be high enough so that all possible solutions have equal chance of being visited. It produces much effect on computation time. Initial temperature T_0 in this paper is set by $T_0 = \Delta f_{\max}$, where $\Delta f_{\max}$ is the maximal difference in fitness value between any two neighboring solutions. And it should be adjusted experimentally.

The epoch length L denotes the number of moves made at the same temperature. According to the method of generating neighborhood solutions, the epoch length can be set as S_N, where S_N is the number of neighborhood solutions for a given solution. In our algorithm S_N is determined by the assignment of operations on each machine.

In SA algorithm, the temperature should be lowered in such a way that the cooling process would not take too long. The exponential cooling scheme specifies the temperature with $T_k = BT_{k-1}$, (during the kth epoch ($k = 1, 2, 3, \ldots$), where $B \in (0,1)$ is temperature decreasing rate) which is often believed to be

Machine	M_1		M_2		M_3		M_4
Operation	$O_{1,1}$	$O_{3,1}$	$O_{1,2}$	$O_{2,2}$	$O_{2,3}$	$O_{3,2}$	$O_{2,1}$

Fig. 2. Switched code based machine order in SA.

excellent in the current literature, since it provides a rather good compromise between a computationally fast schedule and the ability to reach low-energy state. The more the value of decreasing rate B is, the slower the dropping process of temperature goes on, and therefore, much more moves are required before the process is terminated.

As a criterion to terminate the algorithm, we use a simple and general way, in which a termination temperature T_{end} is set. If current temperature $T_k < T_{\text{end}}$, the algorithm will be terminated. T_{end}, with a value near zero, influences the final result of the algorithm directly. Smaller T_{end} implies finer search in problem space when algorithm termination is forthcoming.

5. Hybrid algorithm

Briefly, our hybrid algorithm can be summarized with the following pseudo-code in Fig. 3. It can be seen that during the hybrid search process PSO provides initial solutions for SA. Each particle's fitness value is computed by SA algorithm. In fact SA algorithm is only a sub-algorithm for the entire search process. PSO uses the solutions evaluated by SA to continue evolution.

6. Computational results

To illustrate the effectiveness and performance of algorithm proposed in this paper, three representative instances (represented by *problem* $n \times m$) based on practical data have been selected to compute. Three problem instances (*problem* 8×8, *problem* 10×10 and *problem* 15×10) are all taken from Kacem et al. (2002a,b). We have applied the hybrid algorithm (PSO + SA) to them with following parameters (Table 2).

6.1. Problem 8×8

This is an instance of partial flexibility, i.e. some operations are only achievable on a part of the available machines set. In the original data of Table 3, symbol 'X' indicates that the assignment is impossible. According to our approach, we only need to switch symbol 'X' to infinity, such as 9999. Applying this approach implicates the processing time on impossible machine is infinite and the priority level of machine is lower than other machines. So the machines with infinite processing time (such as 9999) cannot be selected for operations. By this way the partial flexibility problem can be converted to a total flexibility problem.

The obtained solutions by our hybrid algorithm are characterized by the following values:

$$\text{Soultion 1}: \; W_{\text{td}} = 75, \quad \text{Max}(W_k) = 12, \quad \text{Makespan} = 15$$

$$\text{Soultion 2}: \; W_{\text{td}} = 73, \quad \text{Max}(W_{\text{k}}) = 13, \quad \text{Makespan} = 16$$

where, W_{td} is used to represent total workload of all machines, and $\text{Max}(W_k)$ represents the critical machine workload. Figs. 4 and 5 show the results in the form of a Gantt chart. Numbers (in the form of [job, operation]) inside the blocks are the operations associated with the jobs, and blocks with bias represent the machine idle periods.

```
Begin
  Step 1.  Initialization
    *  Initialize parameters, including swarm size, maximum of generation, $W_{max}$, $W_{min}$, $C1$, $C2$;
    *  Determine $T_0$, $T_{end}$, $B$ by experiment.
  Step 2.  Assignment and scheduling
    generation := 0;
    Initialize particle's position and velocity stochastically;
    Evaluate each particle's fitness by SA algorithm subprogram;
    Initialize gbest position with the particle with the lowest fitness in the swarm;
    Initialize pbest position with a copy of particle itself;
    While (the maximum of generation is not met)
      Do {
          generation := generation+1;
          Generate next swarm by equation (1a) and (1b);
          Evaluate swarm {
              Compute each particle's fitness by SA algorithm subprogram;
              Find new gbest and pbest by comparison;
              Update gbest of the swarm and pbest of each particle;
          }
      }
    Step 3.  Output optimization results.
  End

  SA algorithm subprogram (for each particle $S$ of swarm) as follows:
  Code conversion;
  { $T_k = T_0$;
    While ( $T_k > T_{end}$ )
      Do {
          Generate a neighbor solution $S'$ from $S$;
          Compute fitness of $S'$;
          Evaluate $S'$ {
              $\Delta = f(S') - f(S)$;
              if ( min [1, $exp(-\Delta/T_k)$] > random[0, 1] )  { Accept $S'$; }
              Update the best solution found so far if possible;
          }
          $T_k = B*T_{k-1}$;
      }
  }
```

Fig. 3. Hybrid optimization algorithm with pseudo-code.

Table 2
Hybrid algorithm parameters

PSO algorithm		SA algorithm			
			$n\times m$		
			8×8	10×10	15×10
Swarm size	100				
Maximal generation	50				
$W_{\max}$	1.2	T_0	3	5	10
$W_{\min}$	0.4	T_{end}	0.01	0.01	0.01
$C1/C2$	2.0	B	0.9	0.9	0.95

Table 3
Problem 8×8 with 27 operation (partial flexibility)

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
J1	$O_{1,1}$	5	3	5	3	3	X	10	9
	$O_{1,2}$	10	X	5	8	3	9	9	6
	$O_{1,3}$	X	10	X	5	6	2	4	5
J2	$O_{2,1}$	5	7	3	9	8	X	9	X
	$O_{2,2}$	X	8	5	2	6	7	10	9
	$O_{2,3}$	X	10	X	5	6	4	1	7
	$O_{2,4}$	10	8	9	6	4	7	X	X
J3	$O_{3,1}$	10	X	X	7	6	5	2	4
	$O_{3,2}$	X	10	6	4	8	9	10	X
	$O_{3,3}$	1	4	5	6	X	10	X	7
J4	$O_{4,1}$	3	1	6	5	9	7	8	4
	$O_{4,2}$	12	11	7	8	10	5	6	9
	$O_{4,3}$	4	6	2	10	3	9	5	7
J5	$O_{5,1}$	3	6	7	8	9	X	10	X
	$O_{5,2}$	10	X	7	4	9	8	6	X
	$O_{5,3}$	X	9	8	7	4	2	7	X
	$O_{5,4}$	11	9	X	6	7	5	3	6
J6	$O_{6,1}$	6	7	1	4	6	9	X	10
	$O_{6,2}$	11	X	9	9	9	7	6	4
	$O_{6,3}$	10	5	9	10	11	X	10	X
J7	$O_{7,1}$	5	4	2	6	7	X	10	X
	$O_{7,2}$	X	9	X	9	11	9	10	5
	$O_{7,3}$	X	8	9	3	8	6	X	10
J8	$O_{8,1}$	2	8	5	9	X	4	X	10
	$O_{8,2}$	7	4	7	8	9	X	10	X
	$O_{8,3}$	9	9	X	8	5	6	7	1
	$O_{8,4}$	9	X	3	7	1	5	8	X

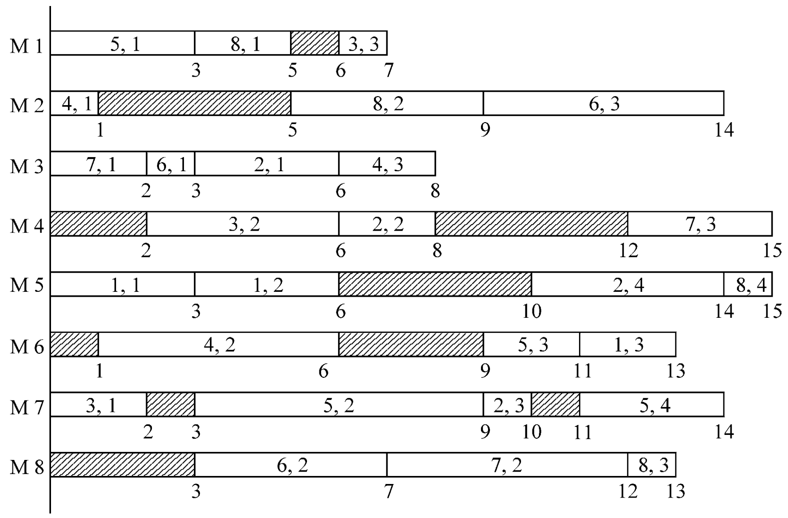

Fig. 4. Optimization solution 1 of problem 8×8 (W_{td}=75, Max(W_k)=12, Makespan=15).

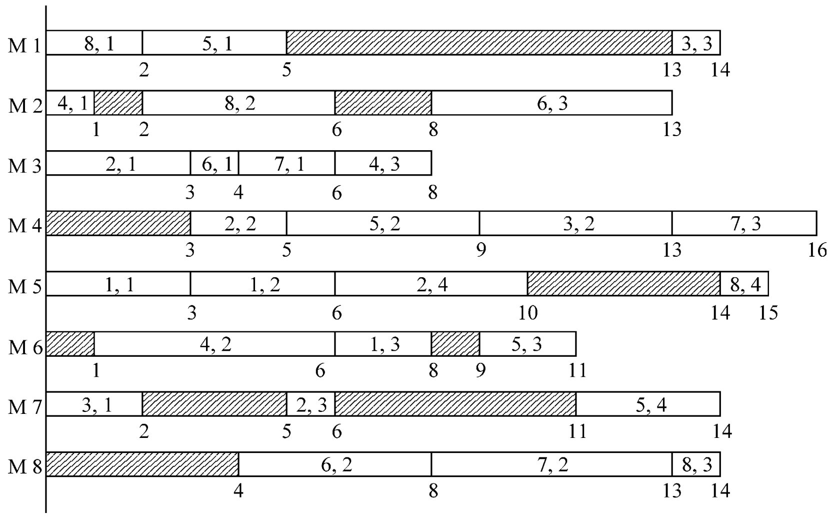

Fig. 5. Optimization solution 2 of problem 8×8 (W_{td}=73, Max(W_k)=13, Makespan=16).

Table 4 shows our hybrid algorithm's effectiveness by comparison with other algorithms. The column labeled 'Temporal Decomposition' refers to F. Chetouane's method (Kacem et al., 2002a) and the next column labeled 'classic GA' refers to classical genetic algorithm. 'Approach by Localization' and 'AL+CGA' are two algorithms by Kacem et al. (2002a,b).

6.2. Problem 10×10

To evaluate the effectiveness of our algorithm, we have applied it to the instance in Table 5.

The computational results by hybrid algorithm based on PSO and SA are shown in the following:

$$W_{\rm td} = 44, \quad {\rm Max}(W_{\rm k}) = 6, \quad {\rm Makespan} = 7$$

Its Gantt chart representation is shown in Fig. 6.

The comparison of our hybrid algorithm with other algorithms is shown in Table 6.

6.3. Problem 15×10

Larger scale instance is chosen to display the optimizing ability of our hybrid algorithm. This instance has 15 jobs with 56 operations that plan to be processed on 10 machines with total flexibility (see Table 7). It is a challenge to our algorithm indeed.

Table 4
Comparison of results on problem 8×8 with 27 operations

	Temporal decomposition	Classic GA	Approach by localization	AL+CGA		PSO+SA	
Makespan	19	16	16	15	16	15	16
W_{td}	91	77	75	79	75	75	73

Table 5
Problem 10×10 with 30 operations (total flexibility)

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J1	$O_{1,1}$	1	4	6	9	3	5	2	8	9	5
	$O_{1,2}$	4	1	1	3	4	8	10	4	11	4
	$O_{1,3}$	3	2	5	1	5	6	9	5	10	3
J2	$O_{2,1}$	2	10	4	5	9	8	4	15	8	4
	$O_{2,2}$	4	8	7	1	9	6	1	10	7	1
	$O_{2,3}$	6	11	2	7	5	3	5	14	9	2
J3	$O_{3,1}$	8	5	8	9	4	3	5	3	8	1
	$O_{3,2}$	9	3	6	1	2	6	4	1	7	2
	$O_{3,3}$	7	1	8	5	4	9	1	2	3	4
J4	$O_{4,1}$	5	10	6	4	9	5	1	7	1	6
	$O_{4,2}$	4	2	3	8	7	4	6	9	8	4
	$O_{4,3}$	7	3	12	1	6	5	8	3	5	2
J5	$O_{5,1}$	7	10	4	5	6	3	5	15	2	6
	$O_{5,2}$	5	6	3	9	8	2	8	6	1	7
	$O_{5,3}$	6	1	4	1	10	4	3	11	13	9
J6	$O_{6,1}$	8	9	10	8	4	2	7	8	3	10
	$O_{6,2}$	7	3	12	5	4	3	6	9	2	15
	$O_{6,3}$	4	7	3	6	3	4	1	5	1	11
J7	$O_{7,1}$	1	7	8	3	4	9	4	13	10	7
	$O_{7,2}$	3	8	1	2	3	6	11	2	13	3
	$O_{7,3}$	5	4	2	1	2	1	8	14	5	7
J8	$O_{8,1}$	5	7	11	3	2	9	8	5	12	8
	$O_{8,2}$	8	3	10	7	5	13	4	6	8	4
	$O_{8,3}$	6	2	13	5	4	3	5	7	9	5
J9	$O_{9,1}$	3	9	1	3	8	1	6	7	5	4
	$O_{9,2}$	4	6	2	5	7	3	1	9	6	7
	$O_{9,3}$	8	5	4	8	6	1	2	3	10	12
J10	$O_{10,1}$	4	3	1	6	7	1	2	6	20	6
	$O_{10,2}$	3	1	8	1	9	4	1	4	17	15
	$O_{10,3}$	9	2	4	2	3	5	2	4	10	23

The best solution obtained by proposed hybrid algorithm is shown by the following values:

$$W_{\mathrm{td}} = 91, \quad \mathrm{Max}(W_{\mathrm{k}}) = 11, \quad \mathrm{Makespan} = 12$$

The schedule corresponding to the best solution is represented by Gantt chart in Fig. 7.

We present the comparison with Kacem et al. (2002b) algorithm in Table 8. From Table 8 we know the new hybrid algorithm makes us reduce the makespan (12 instead of 23) and get the decline nearly 10% in terms of objective function value. It is more appropriate for solving problems on a large scale by our algorithm.

7. Conclusions

We have discussed a new approach based on the hybridization of PSO and SA for solving multi-objective flexible job-shop scheduling problems. Although it does not guarantee the optimality, such an

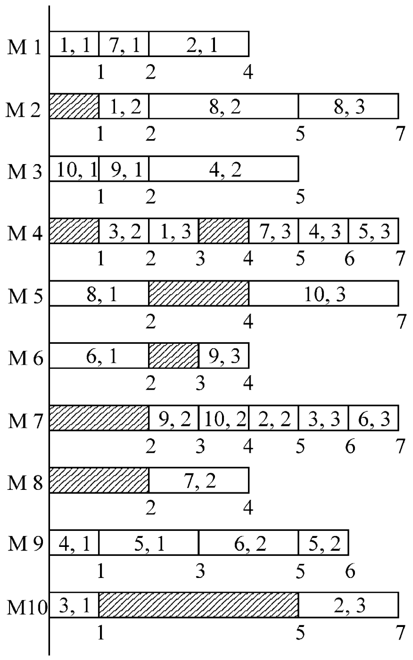

Fig. 6. Optimization solution of problem 10×10 (W_{td}=44, Max(W_k)=6, Makespan=7).

approach provides solutions with good quality in a reasonable time limit. The performance of the new approach is evaluated in comparison with the results obtained from other authors' algorithms for three representative instances. The obtained results show the effectiveness of the proposed approach.

Although there is a huge amount of literature on classical JSP, the FJSP does not have a rich literature. Therefore, there is still a need to develop effective approach for this complex problem. The proposed hybrid approach in this paper can be considered as effective mechanisms from this point of view.

There are a number of research directions that can be considered as useful extensions of this research. Although the proposed algorithm is tested with three representative instances, a more comprehensive computational study should be made to test the efficiency of proposed solution technique. Research on generalization of such an approach for solving other multi-objective optimization problems should be an interesting subject. Furthermore, applying PSO to other combinatorial optimization problems is also possible in further research.

Table 6
Comparison of results on problem 10×10 with 30 operations

	Temporal decomposition	Classic GA	Approach by localization	AL+CGA	PSO+SA
Makespan	16	7	8	7	7
W_{td}	59	53	46	45	44
Max(W_k)	16	7	6	5	6

Table 7
Problem 15×10 with 56 operations (total flexibility)

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J1	$O_{1,1}$	1	4	6	9	3	5	2	8	9	4
	$O_{1,2}$	1	1	3	4	8	10	4	11	4	3
	$O_{1,3}$	2	5	1	5	6	9	5	10	3	2
	$O_{1,4}$	10	4	5	9	8	4	15	8	4	4
J2	$O_{2,1}$	4	8	7	1	9	6	1	10	7	1
	$O_{2,2}$	6	11	2	7	5	3	5	14	9	2
	$O_{2,3}$	8	5	8	9	4	3	5	3	8	1
	$O_{2,4}$	9	3	6	1	2	6	4	1	7	2
J3	$O_{3,1}$	7	1	8	5	4	9	1	2	3	4
	$O_{3,2}$	5	10	6	4	9	5	1	7	1	6
	$O_{3,3}$	4	2	3	8	7	4	6	9	8	4
	$O_{3,4}$	7	3	12	1	6	5	8	3	5	2
J4	$O_{4,1}$	6	2	5	4	1	2	3	6	5	4
	$O_{4,2}$	8	5	7	4	1	2	36	5	8	5
	$O_{4,3}$	9	6	2	4	5	1	3	6	5	2
	$O_{4,4}$	11	4	5	6	2	7	5	4	2	1
J5	$O_{5,1}$	6	9	2	3	5	8	7	4	1	2
	$O_{5,2}$	5	4	6	3	5	2	28	7	4	5
	$O_{5,3}$	6	2	4	3	6	5	2	4	7	9
	$O_{5,4}$	6	5	4	2	3	2	5	4	7	5
J6	$O_{6,1}$	4	1	3	2	6	9	8	5	4	2
	$O_{6,2}$	1	3	6	5	4	7	5	4	6	5
J7	$O_{7,1}$	1	4	2	5	3	6	9	8	5	4
	$O_{7,2}$	2	1	4	5	2	3	5	4	2	5
J8	$O_{8,1}$	2	3	6	2	5	4	1	5	8	7
	$O_{8,2}$	4	5	6	2	3	5	4	1	2	5
	$O_{8,3}$	3	5	4	2	5	49	8	5	4	5
	$O_{8,4}$	1	2	36	5	2	3	6	4	11	2
J9	$O_{9,1}$	6	3	2	22	44	11	10	23	5	1
	$O_{9,2}$	2	3	2	12	15	10	12	14	18	16
	$O_{9,3}$	20	17	12	5	9	6	4	7	5	6
	$O_{9,4}$	9	8	7	4	5	8	7	4	56	2
J10	$O_{10,1}$	5	8	7	4	56	3	2	5	4	1
	$O_{10,2}$	2	5	6	9	8	5	4	2	5	4
	$O_{10,3}$	6	3	2	5	4	7	4	5	2	1
	$O_{10,4}$	3	2	5	6	5	8	7	4	5	2
J11	$O_{11,1}$	1	2	3	6	5	2	1	4	2	1
	$O_{11,2}$	2	3	6	3	2	1	4	10	12	1
	$O_{11,3}$	3	6	2	5	8	4	6	3	2	5
	$O_{11,4}$	4	1	45	6	2	4	1	25	2	4
J12	$O_{12,1}$	9	8	5	6	3	6	5	2	4	2
	$O_{12,2}$	5	8	9	5	4	75	63	6	5	21
	$O_{12,3}$	12	5	4	6	3	2	5	4	2	5
	$O_{12,4}$	8	7	9	5	6	3	2	5	8	4
J13	$O_{13,1}$	4	2	5	6	8	5	6	4	6	2
	$O_{13,2}$	3	5	4	7	5	8	6	6	3	2
	$O_{13,3}$	5	4	5	8	5	4	6	5	4	2
	$O_{13,4}$	3	2	5	6	5	4	8	5	6	4

(*continued on next page*)

Table 7 (*continued*)

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J14	$O_{14,1}$	2	3	5	4	6	5	4	85	4	5
	$O_{14,2}$	6	2	4	5	8	6	5	4	2	6
	$O_{14,3}$	3	25	4	8	5	6	3	2	5	4
	$O_{14,4}$	8	5	6	4	2	3	6	8	5	4
J15	$O_{15,1}$	2	5	6	8	5	6	3	2	5	4
	$O_{15,2}$	5	6	2	5	4	2	5	3	2	5
	$O_{15,3}$	4	5	2	3	5	2	8	4	7	5
	$O_{15,4}$	6	2	11	14	2	3	6	5	4	8

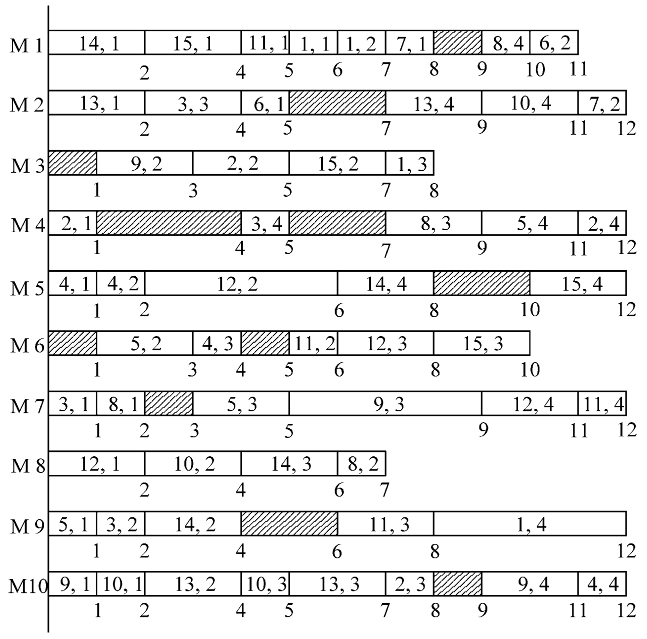

Fig. 7. Optimization solution of problem 15×10 (W_{td}=91, Max(W_k)=11, Makespan=12).

Table 8
Comparison of results on problem 15×10 with 56 operations

	AL+CGA		PSO+SA
Makespan	23	24	12
W_{td}	95	91	91
Max(W_k)	11	11	11

Acknowledgement

This project is supported by National Natural Science Foundation of China (No: 70071017).

References

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by taboo search. *Annals of Operations Research*, *41*, 157–183.

Bruker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, *45*, 369–375.

Dauzere-Peres, S., & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, *70*, 281–306.

Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In: *Proceedings of the sixth international symposium on micro machine and human science* (pp. 39–43).

Eberhart, R., & Shi, Y. (2001). Particle swarm optimization: Developments, applications and resources. In: *Proceedings of the 2001 IEEE international conference on evolutionary computation* (pp. 81–86).

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

Hsu, T., Dupas, R., & Jolly, D., Goncalves, G. (2002). Evalution of mutation heuristics for the solving of multiobjective flexible job shop by an evolutionary algorithm. In: *Proceedings of the 2002 IEEE international conference on systems, man and cybernetics* (pp. 6–9).

Hurink, E., Jurisch, B., & Thole, M. (1994). Tabu search for the job shop scheduling problem with multi-purpose machines. *Operations Research Spektrum*, *15*, 205–215.

Kacem, I., Hammadi, S., & Borne, P. (2002a). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, *32*(1), 1–13.

Kacem, I., Hammadi, S., & Borne, P. (2002b). Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, *60*, 245–276.

Kennedy, J. (1997). The particle swarm: social adaptation of knowledge. In: *Proceeding of the 1997 IEEE international conference on evolutionary computation* (pp. 303–308).

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In: *Proceeding of the 1995 IEEE international conference on neural network, IV* (pp. 1942–1948).

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Mastrolilli, M., & Gambardella, L. M. (2002). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, *3*(1), 3–20.

Shi, Y., & Eberhart, R. (1999). Empirical study of particle swarm optimization. In: *Proceedings of congress on evolutionary computation* (pp. 1945–1950).

Tung, L. F., Li, L., & Nagi, R. (1999). Multi-objective scheduling for the hierarchical control of flexible manufacturing systems. *The International Journal of Flexible Manufacturing Systems*, *11*, 379–409.

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