Information Sciences 181 (2011) 4804–4822

Contents lists available at ScienceDirect

Information Sciences

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

An effective shuffled frog-leaping algorithm for multi-mode resource-constrained project scheduling problem

Ling Wang [*], Chen Fang

Tsinghua National Laboratory for Information Science and Technology (TNList), Department of Automation, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Article history:
Received 5 July 2010
Received in revised form 6 May 2011
Accepted 6 June 2011
Available online 12 June 2011

Keywords:
Multi-mode resource-constrained project scheduling
Shuffled frog-leaping algorithm
Multi-mode serial schedule generation scheme
Multi-mode permutation based local search
Multi-mode forward–backward improvement
Combined local search

ABSTRACT

In this paper, an effective shuffled frog-leaping algorithm (SFLA) is proposed for solving the multi-mode resource-constrained project scheduling problem (MRCPSP). In the SFLA, the virtual frogs are encoded as the extended multi-mode activity list (EMAL) and decoded by the multi-mode serial schedule generation scheme (MSSGS). Initially, the mode assignment lists of the population are generated randomly, and the activity lists of the population are generated by the regret-based sampling method and the latest finish time (LFT) priority rule. Then, virtual frogs are partitioned into several memeplexes that evolve simultaneously by performing the simplified two-point crossover (STPC). To enhance the exploitation ability, the combined local search including the multi-mode permutation based local search (MPBLS) and the multi-mode forward–backward improvement (MFBI) is further performed in each memeplex. To maintain the diversity of each memeplex, virtual frogs are periodically shuffled and reorganized into new memeplexes. Computational results based on the well-known benchmarks and comparisons with some existing algorithms demonstrate the effectiveness of the proposed SFLA.

1. Introduction

The resource-constrained project scheduling problem (RCPSP) is concerned with scheduling project activities over time and resources. The multi-mode resource-constrained project scheduling problem (MRCPSP) is an extension of the RCPSP. Generally, the MRCPSP is much closer to reality. In a multi-mode resource-constrained project, each activity has several execution modes. Each mode has related duration and nonrenewable and renewable resource requirements. The MRCPSP is more complex than the RCPSP. Kolisch and Drexl [13] have proven that even finding a feasible solution of the MRCPSP is NP-complete if there is more than one non-renewable resource. Until now, only the MRCPSP with no more than 20 activities can be solved optimally using the exact algorithms with acceptable computational time [26]. In practice, near-optimal schedule is usually enough under the condition that the computational time is limited. Thus, heuristics for the MRCPSP have gained increasing attention during the past decades.

Kolisch and Drexl [13] proposed a local search approach, which contained a construction phase to generate an initial solution, a local search phase to perform a neighbourhood search on the feasible mode-assignment set, and an intensification phase to improve the solution.

As for genetic algorithm (GA), Hartmann [10] developed a GA to solve the MRCPSP, in which single-pass improvement and multi-pass improvement were adopted to enhance the local search. Alcaraz et al. [1] also proposed a GA to solve the

* Corresponding author. Tel.: +86 10 62783125; fax: +86 10 62786911.
E-mail addresses: wangling@tsinghua.edu.cn, wangling@mail.tsinghua.edu.cn (L. Wang).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.06.014

MRCPSP. Different from the GA developed by Hartmann, a forward/backward gene was adopted in the solution representation, and a new fitness function was developed to improve the performance of the GA.

As for simulated annealing (SA), Józefowska et al. [12] proposed a SA algorithm to solve the MRCPSP, in which two versions of SA were discussed: SA without penalty function and SA with penalty function. In both cases, three neighbourhood generation mechanisms were applied: neighbourhood shift, mode change and combination of neighbourhood shift and mode change. Bouleimen and Lecocq [2] proposed a SA algorithm for both the RCPSP and the MRCPSP. For the RCPSP, activity list was used as the representation, and the search was based on alternated activity and time incrementing process. For the MRCPSP, they used the activity list and mode list to represent a solution, and they also introduced two embedded search loops alternating activity and mode neighborhood exploration.

As for particle swarm optimization (PSO), Zhang et al. [29] proposed a PSO based approach, in which priority combination and mode combination were adopted to represent a particle and a procedure checking and adjusting infeasible particle-represented solutions was adopted to transform the infeasible solution into a feasible one. Jarboui et al. [11] proposed a combinatorial PSO approach, in which a unique representation was adopted.

In addition, Van Peteghem and Vanhoucke [21] proposed an artificial immune system (AIS) to solve the MRCPSP. In the AIS, the mode assignment list generation was translated to a multi-choice multi-dimensional knapsack problem. Wauters et al. [28] proposed a multi-agent learning approach, where an agent was placed each activity node to decide the order to visit its successors and the mode to execute the activity. Damak et al. [4] proposed a differential evolution (DE) approach to solve the MRCPSP. Elloumi and Fortemps [7] proposed a hybrid rank-based evolutionary algorithm that transformed the MRCPSP to a bi-objective problem to cope with the potential violation of the nonrenewable resource constraints.

The shuffled frog-leaping algorithm (SFLA) is one of the latest meta-heuristic methods. The SFLA combines the benefits of genetic-based memetic algorithm (MA) and the social behavior-based PSO algorithm [8]. Different from PSO algorithm, the SFLA considers both gene evolution and meme evolution. Elbeltagia et al. [6] provided a brief comparison among five meta-heuristics including PSO and SFLA. In the SFLA, each individual is called a frog, and the whole population is divided into a number of memeplexes. The different memeplexes are considered as different cultures of frogs, and each memeplex performs a local search by moving around in the search space with the help of neighborhood frogs' social experiences in the same memeplex. So far, the SFLA has been applied to many optimization problems, such as water distribution network design [9], mixed-model assembly line sequencing problem [23], bi-criteria permutation flow-shop problem [22,24] and general large-scale water supply system [3].

To the best of our knowledge, there is no published work solving the MRCPSP by the SFLA. In this paper, we propose an effective SFLA for solving the MRCPSP with the criterion to minimize the makespan. In particular, we propose a new encoding scheme based on the extended multi-mode activity list (EMAL) and a decoding scheme based on the multi-mode serial schedule generation scheme (MSSGS). In the initial population, the mode assignment lists are generated randomly and the activity lists are generated by the regret-based sampling method and the latest finish time (LFT) priority rule. We propose the simplified two-point crossover (STPC) to evolve the population. Moreover, the combined local search including the multi-mode permutation based local search (MPBLS) and the multi-mode forward–backward improvement (MFBI) is performed in each memeplex to enhance the exploitation ability. To maintain the diversity of each memeplex, the frogs are periodically reorganized into new memeplexes to exchange information. Computational results based on benchmarks and comparisons with some existing algorithms demonstrate the effectiveness of the SFLA.

The remainder of the paper is organized as follows: in Section 2, the MRCPSP is described and formulated. In Section 3, the basic SFLA is introduced. The SFLA for the MRCPSP is proposed in detail in Section 4. The computational results and comparisons are provided in Section 5. Finally we end the paper with some conclusions and future work in Section 6.

2. Multi-mode resource-constrained project scheduling problem

Generally, the MRCPSP can be stated as follows. A project consists of J activities labeled as $j = 1, \ldots, J$, and the number of modes in which activity j can be performed is denoted as M_j. The duration of activity j performed in mode m is denoted by d_{jm}. There exist precedence relationships between some activities of the project. The precedence relationships are given by sets of immediate predecessors θ_j indicating that an activity j may not be started before each of its predecessors $i \in \theta_j$ is completed. The number of renewable resources is referred as K^ρ. For each resource $k = 1, 2, \ldots, K^\rho$, the per-period-availability is assumed to be constant R_k^ρ. Activity j requires r_{jmk} units of renewable resource k in each period of its non-preemptable duration. The number of nonrenewable resources is referred as K^ν. For each $k = 1, 2, \ldots K^\nu$, the amount of availability is R_k^ν. Activity j requires n_{jmk} units of nonrenewable resource k. The activities $j = 0$ and $j = J + 1$ are dummy activities, which represent the start and end of the project, respectively. It assumes that the dummy activities do not request any resource and their durations equal to zero. The set of all activities including the dummy activities is denoted as $J^+ = \{0, \ldots, J + 1\}$. The commonly used objective is to minimize the makespan of the project, which is also considered in this paper.

The MRCPSP can be formulated as follows:

$$\text{Minimize} \quad \sum_{t=EF_{J+1}}^{LF_{J+1}} t \cdot x_{J+1,1,t} \tag{1}$$

$$\text{Subject to} \quad \sum_{m=1}^{M_j} \sum_{t=EF_j}^{LF_j} x_{jmt} = 1, \quad j \in J^+ \tag{2}$$

$$\sum_{m=1}^{M_i} \sum_{t=EF_i}^{LF_i} t \cdot x_{imt} \leqslant \sum_{m=1}^{M_j} \sum_{t=EF_j}^{LF_j} (t - d_j) x_{jmt}, \quad j \in J^+;\ i \in \theta_j \tag{3}$$

$$\sum_{j=1}^{J} \sum_{m=1}^{M_j} r_{jmk} \sum_{b=\max\{t,EF_j\}}^{\min\{t+d_{jm}-1,LF_j\}} x_{jmb} \leqslant R_k^{\rho}, \quad k = 1,2,\ldots,K^{\rho};\ t = 1,2,\ldots,T \tag{4}$$

$$\sum_{j=1}^{J} \sum_{m=1}^{M_j} n_{jmk} \sum_{t=EF_j}^{LF_j} x_{jmt} \leqslant R_k^{\nu}, \quad k = 1,2,\ldots,K^{\nu} \tag{5}$$

$$x_{jmt} = \begin{cases} 1, & \text{if activity } \ j \text{ is performed in mode } m \ \text{ and finished at time } t \\ 0, & \text{otherwise} \end{cases}, \quad j \in J^+ \tag{6}$$

where LF_i is the latest finish time of activity i; EF_i is the earliest finish time of activity i; T is a feasible upper bound on the project makespan. Eq. (1) defines that the objective is to minimize the makespan; Eq. (2) guarantees each activity is executed exactly once; Eq. (3) confirms the precedence constraints are satisfied; Eqs. (4) and (5) represent the renewable and nonrenewable resource constraints, respectively.

A simple example which contains a project with 11 activities (including two dummy activities $j = 0$ and $j = 10$) is illustrated in Fig. 1. The project is represented by an activity on node (AoN) network, in which a node represents an activity and an arc represents a precedence constraint between two activities. In this example, one kind of renewable resource and one kind of nonrenewable resource are considered (i.e. $K^{\rho} = K^{\nu} = 1$), and the maximum amounts of renewable and nonrenewable resources are 2 and 28, respectively (i.e. $R^{\rho} = 2$, $R^{\nu} = 28$). Each activity contains two execution modes. The resource requirements and corresponding durations of each activity in different modes are listed in Table 1. For example, if activity 5 is executed in mode 2, its duration is 4 and it needs 2 units of renewable resource and 3 units of nonrenewable resource.

In Fig. 2, a schedule of this project is shown. The execution modes of the activities $j = 1, 2, \ldots, J$ are 2, 1, 2, 1, 2, 2, 2, 2, and 1, respectively. We can see that the precedence constraints are satisfied. For example, activity 5, as a successor of activity 2, starts at time instant 2, at which time activity 2 has already been finished. Furthermore, both renewable and nonrenewable resource constraints have been satisfied, since the amount of nonrenewable resource consumed (NRC) is 25 which is less than R^{ν} and the renewable resource consumed (RRC) is also less than R^{ρ} at each period. As a result, this is a feasible schedule with makespan 18.

3. Basic shuffled frog leaping algorithm

The shuffled frog-leaping algorithm is a population-based evolution algorithm originally proposed by Eusuff et al. [8] for solving discrete decision problems. Inspired by the process that frogs hunt for food, the SFLA stresses the balance between the exploration and exploitation in the evolution process. In the SFLA, an initial population is formed by a set of randomly generated solutions called virtual frogs, each of which is represented by a vector. First, the whole population is partitioned into *sfla_m* subsets called memeplexes. Different memeplexes, each of which contains *sfla_n* frogs, are considered as different cultures of frogs and evolve independently in the whole searching space. Each memeplex is sorted so that frogs are arranged

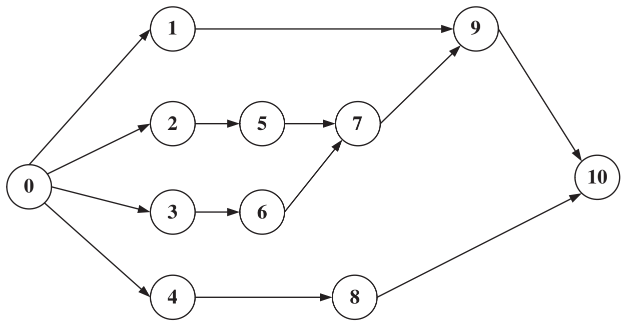


Fig. 1. An example of the MRCPSP.

Table 1
Resource requirements and corresponding durations.

Activity	Mode 1			Mode 2		
	r_{jm}	n_{jm}	d_{jm}	r_{jm}	n_{jm}	d_{jm}
1	2	4	2	1	2	3
2	1	5	2	1	2	4
3	1	3	2	1	1	3
4	1	2	1	1	1	2
5	2	5	2	2	3	4
6	1	4	2	1	3	3
7	2	2	1	2	1	2
8	2	3	1	1	2	3
9	1	3	4	1	1	6

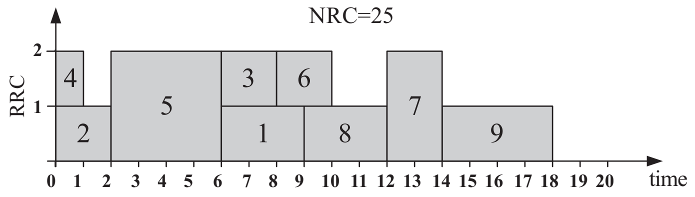


Fig. 2. A feasible schedule of the example in Fig. 1.

in order of decreasing performance ($j = 1,\ldots,sfla\_n$). In each memeplex, a subset of the memeplex called submemeplex is constructed. Each submemeplex contains *sfla_q* frogs that are selected randomly from their memeplex according to triangular probability distribution, i.e. $p_j = 2(sfla\_n + 1 - j)/[sfla\_n(sfla\_n + 1)]$, $j = 1, \ldots, sfla\_n$. In each submemeplex, the worst frog P_w will leap according to its own experience as well as the experience from the best frog P_B in this memeplex. The leaping step size is calculated as follows:

$$S_B = \begin{cases} \min\{\text{int}[rand \cdot (P_B - P_w)], S_{\max}\} & \text{for a positive step} \\ \max\{\text{int}[rand \cdot (P_B - P_w)], -S_{\max}\} & \text{for a negative step} \end{cases} \tag{7}$$

where *rand* is a random real number between 0 and 1, and $S_{\max}$ is the maximum leaping step size.

Then, the new position of the worst frog is calculated by

$$P'_w = P_w + S_B \tag{8}$$

If the new position is better then the old one, replace P_w with P'_w; else, another new position P''_w of this frog will adjust according to the best frog P_G in the entire population with the following leaping step size:

$$S_G = \begin{cases} \min\{\text{int}[rand \cdot (P_G - P_w)], S_{\max}\} & \text{for a positive step} \\ \max\{\text{int}[rand \cdot (P_G - P_w)], -S_{\max}\} & \text{for a negative step} \end{cases} \tag{9}$$

$$P''_w = P_w + S_G \tag{10}$$

If the new position P''_w is better than the old one, replace P_w with P''_w; else, the worst frog is replaced by a randomly generated frog.

After evolving *sfla_N* steps, all the frogs are shuffled and divided into *m* new memeplexes. Such frog leaping and shuffling processes are repeated until the stopping condition is satisfied. Clearly, the SFLA can be regarded as a kind of PSO algorithm if only one memeplex exists (i.e. $sfla\_m = 1$). The pseudo code of the basic SFLA is summarized in Fig. 3.

4. The proposed SFLA for the MRCPSP

4.1. General overview

In this section, a SFLA for the MRCPSP is proposed. The procedure of the proposed SFLA is illustrated in Fig. 4.

First, the preprocessing procedure is adopted to reduce the searching space and the regret-based biased random sample method with LFT priority rule is used to generate an initial population. Then the MFBI operation is applied to improve the initial population. For every generation, the whole population is partitioned into *sfla_m* memeplexes. In each memeplex, a submemeplex is generated and then the STPC is performed to generate a child to replace *Pw*. In particular, the STPC is first applied to the best frog *PB* and worst frog *Pw* of the submemeplex to generate a child; if the child is worse than P_W, the STPC is applied to *PW* and *PG* (the best frog in the whole population) to generate a child once again; if the new child is still worse

```
BEGIN
   Randomly generate an initial population;
   DO
Divide the whole population into $m$ memeplexes, and each memeplex contains $n$ frogs;
Construct a submemeplex for each memeplex;
Calculate the new position of the worst frog in each submemeplex by Eqs. (7) and (8);
Set $i$=0;
FOR ($i$<$sfla\_N$)
   IF ( $P_w'$ is not better than $P_w$ )
      Calculate the new position of the worst frog by Eqs. (9) and (10);
      IF ( $P_w''$ is not better than $P_w$ )
         Randomly generate a new frog to instead the old one;
      ELSE
         $P_w = P_w''$;
   ELSE
      $P_w = P_w'$;
   i++;
Shuffle the whole population;
WHILE (stopping condition is not met);
END
```

Fig. 3. Procedure of the basic SFLA.

than P_W, Pw will be replaced by a random generated frog. For the child generated by the STPC, the MPBLS and MFBI are applied for further improvement. After *sfla_N* iterations of crossover and local search, all the frogs of all the memeplexes are shuffled to perform the next generation of evolution.

4.2. Preprocessing

At the beginning of the SFLA, we utilize the reduction procedure developed by Sprecher et al. [27] to reduce the search space. This procedure can exclude the inefficient/non-executable modes as well as redundant resources. A mode m' is called inefficient if there exists another mode m with $d_{jm} \leqslant d_{jm'}$ and $r_{jmk} \leqslant r_{jm'k}$ for each renewable resource $k \in K^{\rho}$ and $n_{jmk} \leqslant n_{jm'k}$ for each nonrenewable resource $k \in K^{v}$. A mode m' is called non-executable if $\sum_{\substack{i=1 \\ i\neq j}}^{J} \min_m n_{imk} + n_{jm'k} > R_k^{v}$ for each nonrenewable resource $k \in K^{v}$. A nonrenewable resource $k \in K^{v}$ is called redundant if $\sum_{j=1}^{J} \max_m n_{jmk} \leqslant R_k^{v}$. The procedure of preprocessing is summarized in Fig. 5.

4.3. Encoding scheme

Ideally, an individual representation for the RCPSP should meet the following conditions [20]: (1) The transformation between solutions should be computationally fast; (2) For each solution in the original space, there is a solution in the encoded space; (3) Each encoded solution corresponds to one feasible solution in the original space; (4) All solutions in the original space should be represented by the same number of encoded solutions; (5) Small changes in the encoded solution should result in small changes in the solution itself. Since the MRCPSP is the generalization of the RCPSP, the five conditions also should be considered for the MRCPSP. Kolisch and Hartmann [14] concluded by experimental tests that procedures based on activity list representation outperformed other procedures. In the MRCPSP, the mode assignment list is needed to assign every activity an execution mode. A list with the start (finish) time of every activity will be helpful when the multi-mode backward (forward) schedule is performed.

According to all these considerations, we propose an extended multi-mode activity list (EMAL) which contains four parts: (1) an activity list (AL) $\{\pi_0, \pi_1, \pi_2, \ldots, \pi_{J+1}\}$; (2) a mode assignment list (ML) $\{\lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_{J+1}\}$; (3) a list with the start time of every activity (SL) $\{st_0, st_1, st_2, \ldots, st_{J+1}\}$; and (4) a list with the finish time of every activity (FL) $\{ft_0, ft_1, ft_2, \ldots, ft_{J+1}\}$.

Clearly, such an EMAL based encoding scheme meets the above five conditions. Consider again the example illustrated in Fig. 1, the EMAL representation for the schedule (shown in Fig. 2) is given in Fig. 6, where the mode, start time and finish time of each activity can be seen clearly from the EMAL representation. For example, for activity 3, the mode is 2, the start time is 6, and the finish time is 8.

In Section 4.8.2, we will show that the local search MFBI can be speeded up with the help of the EMAL.

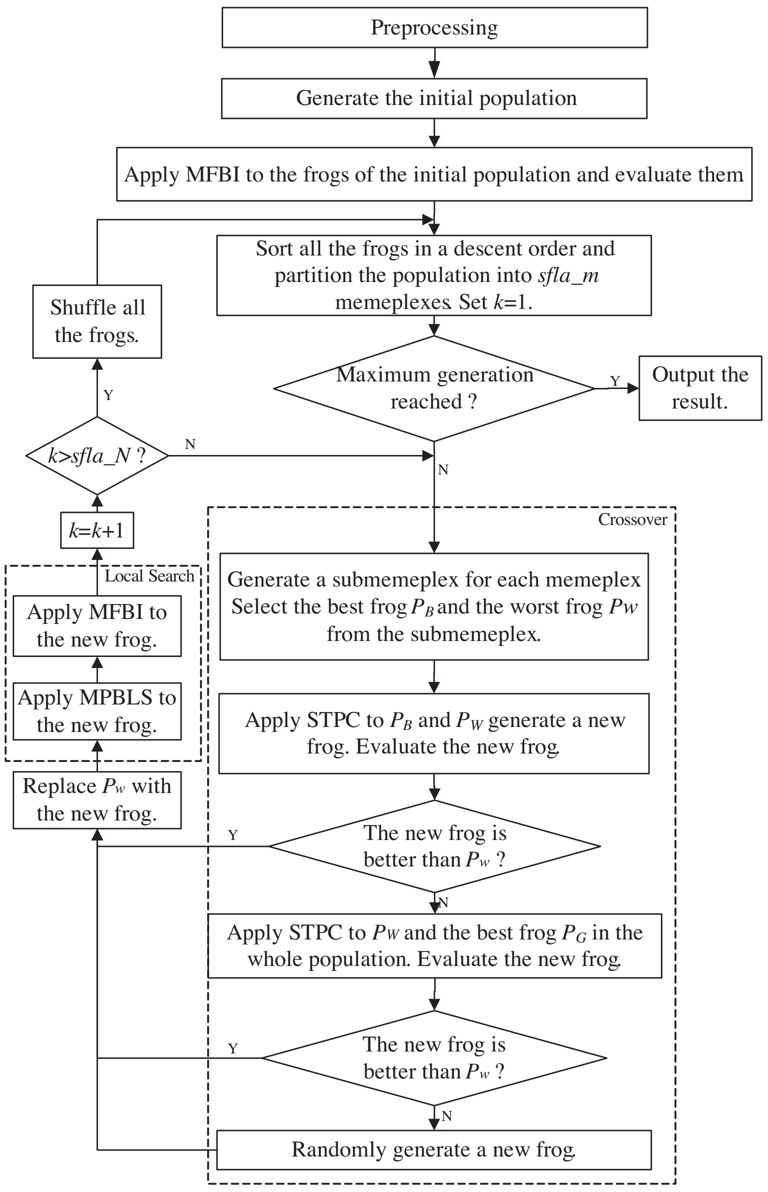

Preprocessing
Generate the initial population
Apply MFBI to the frogs of the initial population and evaluate them
Sort all the frogs in a descent order and partition the population into sfla_m memeplexes. Set k=1.
Shuffle all the frogs.
Maximum generation reached ?
Y
Output the result.
Y
k>sfla_N ?
N
N
k=k+1
Crossover
Local Search
Apply MFBI to the new frog.
Apply MPBLS to the new frog.
Replace Pw with the new frog.
Generate a submemeplex for each memeplex Select the best frog PB and the worst frog Pw from the submemeplex.
Apply STPC to PB and PW generate a new frog. Evaluate the new frog.
The new frog is better than Pw ?
Y
N
Apply STPC to Pw and the best frog PG in the whole population. Evaluate the new frog.
The new frog is better than Pw ?
Y
N
Randomly generate a new frog.

Fig. 4. Procedure of the proposed SFLA for the MRCPSP.

Step1: Remove all non-executable modes;

Step2: Delete the redundant nonrenewable resources;

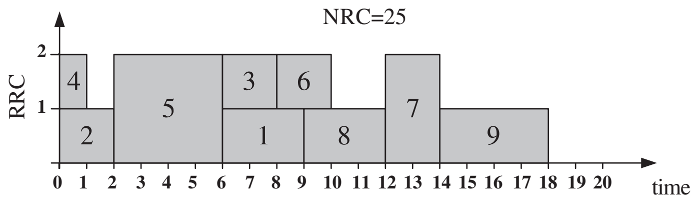
Step3: Eliminate all inefficient modes;

Step4: If any mode has been erased within Step 3, go to Step2.

Fig. 5. Procedure of preprocessing.

4.4. Multi-mode serial schedule generation scheme

Schedule generation scheme (SGS) is the core of most heuristic procedures for the RCPSP, which is used to decode a solution representation to a schedule. There are two kinds of schedule generation schemes frequently used in literature: the serial SGS based on activity incrementation and the parallel SGS based on time incrementation [15].



AL	2	4	5	1	3	6	8	7	9
ML	1	1	2	2	2	2	2	2	1
ST	0	0	2	6	6	8	9	12	14
FT	2	1	6	9	8	10	12	14	18

Fig. 6. The EMAL representation for the schedule shown in Fig. 2.

For the MRCPSP, it is more complicated to transform a solution representation to a schedule. The SGS cannot be used for the MRCPSP, since there is no conception of nonrenewable resource. That is, the SGS cannot deal with nonrenewable resource infeasible representation. There are two more things need to be considered for the MRCPSP. First, we need to indicate a suitable fitness value for the nonrenewable resources infeasible representation. Second, we need to fully utilize the nonrenewable resources to minimize the fitness value for the nonrenewable resources feasible representation.

In this paper, we propose a multi-mode version serial schedule generation scheme (MSSGS) for the EMAL. For an EMAL with ML $\{\lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_{J+1}\}$, the nonrenewable infeasible degree is calculated by $ERR(ML) = \sum_{k=1}^{K^v} \left(\max\left(0, \sum_{j=1}^{J} n_{j\lambda_j k} - R_k^v\right)\right)$. It is noticeable that $ERR(ML) = 0$ if the EMAL is nonrenewable resources feasible.

The MSSGS contains two procedures: infeasible tackling procedure to tackle infeasible representation, and feasible tackling procedure to tackle feasible representation.

The infeasible tackling procedure chooses an activity randomly and changes its mode to a new one with lower nonrenewable resources consumption. The procedure is repeated until the representation becomes feasible or the maximum iteration number *SGS_adj* is reached. In our procedure, we adopt the number of activities *J* as the maximum iteration number, i.e., *SGS_adj=J*.

This feasible tackling procedure is based on the multi-mode left shift bounding rule developed by Sprecher et al. [27]. A multi-mode left shift is a rule to improve the finish time of activity one by one at the every decision point of the partial schedule by changing its mode without changing the modes and delaying the finish times of any other activities. There is a weakness of the multi-mode left shift bounding rule: activities scheduled earlier will occupy more amounts of nonrenewable resources, so the activities scheduled later will not have desirable nonrenewable resources to use. In our procedure, we develop a probability based selecting scheme controlled by a threshold *SGS_Pper* to ensure that all the activities have an equal opportunity to use nonrenewable resources. The pseudo code of MSSGS is illustrated in Fig. 7.

In the procedure of MSSGS, we calculate nonrenewable infeasible degree $ERR(ML)$ first. If $ERR(ML) > 0$, the infeasible tackling procedure will be processed to reduce the infeasible degree; if $ERR(ML) = 0$, feasible tackling procedure will be processed to calculate the makespan.

With the help of the proposed MSSGS, when a new EMAL is generated (i.e. the AL and ML parts of the EMAL are changed by searching operation), the other two parts of the EMAL will be updated automatically. MSSGS can not only reduce the infeasible degree of an infeasible representation but also improve the fitness value of a feasible representation by fully utilizing the nonrenewable resources. In a word, the MSSGS will reduce the nonrenewable resource consumption for infeasible representations and increase the nonrenewable resource consumption for feasible representations.

4.5. Population initialization

In our SFLA, the regret-based biased random sample method [18] with priority rule is adopted to generate the initial population. First, the ML of the EMAL is generated randomly. Second, with generated ML, we obtain an AL with feasible precedence by repeatedly taking an activity from the eligible set which contains activities whose predecessors have been selected into AL but themselves are still not selected. During the process of constructing the activity list, the latest finish time (LFT) priority rule [5] is employed to derive the probabilities to decide which activity is selected as the next activity at each decision stage.

Let η_j be the probability to choose activity j. And η_j can be calculated as follows:

$$\eta_j = \frac{(\mu_j + \varepsilon)^\alpha}{\sum_{i \in D} (\mu_j + \varepsilon)^\alpha} \tag{11}$$

$$\mu_j = \max_{i \in D} LFT_i - LFT_j \tag{12}$$

```
Procedure MSSGS($EMAL$)
BEGIN
Initialization $j$=1,  $\pi R_{kt} = R_k^\rho$;
Calculate the ERR($ML$);
IF ERR($ML$)>0
  DO
    Randomly select an activity $j$;
    Select a new mode for activity $j$, create a new mode assignment list $ML'$
    IF ERR ( $ML'$ ) < ERR($ML$)
      $ML = ML'$;
    $k$++;
  WHILE $k$<$SGS\_adj$ && ERR($ML$)==0;
IF ERR($ML$)>0
  $\text{Makespan}(EMAL) = \sum_{j=1}^{J} d_{\pi_j \lambda_j} + \text{ERR}(ML)$;
ELSE
  WHILE $j$<J
    Calculate the remaining capacity of resource r in period t  $\pi R_{rt} = R_k^\rho - \sum_{j \in A_t} r_{j\lambda_j k}$;
    Calculate the earliest precedence feasible finish time  $EFT_{\pi_j} = \max\{FT_i \mid i \in P_{\pi_j}\} + d_{\pi_j \lambda_j}$;
    Randomly generate $q$ where 0<$q$<1;
    IF $q$<$SGS\_Pper$
      Calculate the all the finish time with different modes  $m = 1, \ldots, M_j$:
      $FT_{\pi_j m} = \min\{t \mid EFT_{\pi_j} \le t, r_{\pi_j mk} \le \pi R_{k\tau}, \tau = t - d_{\pi_j m} + 1, \ldots, t, k \in K^\rho\}$;
      Select the mode with the minimized finish time of activity $j$:  $\lambda_j = \arg \min_{\substack{m=1,\ldots,M_j \\ ERR(m)=0}} FT_{\pi_j m}$;
      $FT_{\pi_j} = FT_{\pi_j \lambda_j}$;
    ELSE
      Calculate the finish time:
      $FT_{\pi_j} = \min\{t \mid EFT_{\pi_j} \le t, r_{\pi_j \lambda_j k} \le \pi R_{k\tau}, \tau = t - d_{\pi_j \lambda_j} + 1, \ldots, t, k \in K^\rho\}$;
    $ST_{\pi_j} = FT_{\pi_j} - d_{\pi_j}$;
    Record  $ST_{\pi_j}$  and  $FT_{\pi_j}, j = 1, \ldots, J$;
    $j$=$j$+1;
  $\text{Makespan}(EMAL) = FT_{\pi_J}$
END
```

Fig. 7. Procedure of the MSSGS.

where LFT_j is the latest finish time of activity j, and D denotes the eligible set. In [16], Kolisch pointed out that the regret-based biased random sample method was of good performance when $\varepsilon = \alpha = 1$. So, we also set $\varepsilon = \alpha = 1$ in our SFLA.

4.6. Population partition

In the SFLA, the entire population is partitioned into *sfla_m* memeplexes, and then each memeplex with *sfla_n* frogs performs an independent memetic search according to its own search experience. First, all the frogs are sorted in a descent order according to their objective values (i.e., makespan). Then the whole population is partitioned as follows: the $((i-1) \cdot sfla_n + j)$th frog enters the jth memeplex, $i, j = 1, \ldots, sfla\_m$. Suppose $sfla\_m = 3$, $sfla\_n = 4$, then the rank 1, rank 4, rank 7, and rank 10 frogs form memeplex 1; the rank 2, rank 5, rank 8, and rank 11 frogs form memeplex 2; the rank 3, rank 6, rank 9, and rank 12 frogs form memeplex 3.

4.7. Frog-leaping process (crossover)

In the SFLA, the frogs in each memeplex perform an evolution process to improve their positions and try to drive other frogs towards the global optima. Inspired by the two-point crossover of traditional binary encoded GA and the multi-mode two-point forward–backward crossover (MMTPFBC) proposed by Alcaraz et. al. [1], we proposed the simplified two-point crossover (STPC) to perform the frog-leaping process.

```
Procedure STPC
BEGIN
FOR $i = 0,\dots,q_1 \cup q_2\dots,J+1$
    $\pi_i^C = \pi_i^F$;
    $\lambda_i^C = \lambda_i^F$;
FOR $i = q_1+1$ to $q_2-1$
    $j = \min\{j \mid 0 \le j \le J+1, \pi_j^M \notin \pi^C\}$;
    $\pi_i^C = \pi_j^M$;
    $\lambda_i^C = \lambda_j^M$;
END
```

Fig. 8. Procedure of the STPC.

Suppose two frogs are selected, where the worse one is denoted as *M* and the better one is denoted as *F*. The STPC performs as follows.

First, we randomly choose two integers q_1 and q_2 (called crossover points), $0 \leqslant q_1 < q_2 \leqslant J + 1$. Then, the new frog *C* is generated as follows: The positions from 0 to q_1 and the positions from q_1 to $J + 1$ in *C* are inherited from the father, exactly in the same order; the positions between q_1 and q_2 are inherited from the mother preserving their relative positions in the mother's sequence. As a result, the EMAL of the new frog *C* is precedence feasible if *F* and *M* are precedence feasible. The procedure of STPC is shown in Fig. 8.

The MMTPFBC actually contains two crossover operators: one for forward scheduling mode, and the other for backward scheduling mode. The MMTPFBC needs to adopt different crossover operators when the parents have different scheduling modes. The STPC simplifies the MMTPFBC by combining the two crossover operators into one. As a result, no matter what the scheduling modes of parents are, the STPC can treat them indistinguishably. This character makes the STPC more convenient to use.

4.8. Local search

In our SFLA, a combined local search including the multi-mode permutation based local search (MPBLS) and the multi-mode forward–backward improvement (MFBI) is developed to enhance the exploitation ability. In particular, the MPBLS is applied to the new child generated by crossover, and then the MFBI is used for further improvement.

4.8.1. MPBLS

To enhance the exploitation ability, a multi-mode permutation-based local search strategy MPBLS controlled by a threshold *sfla_Pper* is proposed to explore the neighborhood of a frog. This local search is performed by swapping the adjacent activities, but it happens if and only if there is no precedence relationship between the adjacent activities. As a result, the MPBLS does not break the feasibility of the EMAL if it is feasible before performing local search. The procedure of MPBLS is described in Fig. 9.

4.8.2. MFBI

The forward–backward improvement (FBI) is an effective technique for the RCPSP introduced by Li and Willis [19]. Basically, the procedure iteratively employs SGS forward and backward scheduling until no further improvement in the makespan can be found. But the FBI cannot be used for the MRCPSP, since FBI cannot tackle the representation which is nonrenewable resource infeasible.

```
Procedure MPBLS
BEGIN
Randomly generate $q$ where $0<q<1$;
FOR $i=1,2,\dots,n$
  IF ($q<sfla\_Pper$)
    Randomly change the mode of activity $j$;
    IF ($\pi_i$ is not the predecessor for $\pi_{i+1}$)
      Exchange $\pi_i$ and $\pi_{i+1}$;
      Exchange $\lambda_i$ and $\lambda_{i+1}$;
Evaluation the new EMAL and update the SL and FL of EMAL.
END
```

Fig. 9. Procedure of the MPBLS.

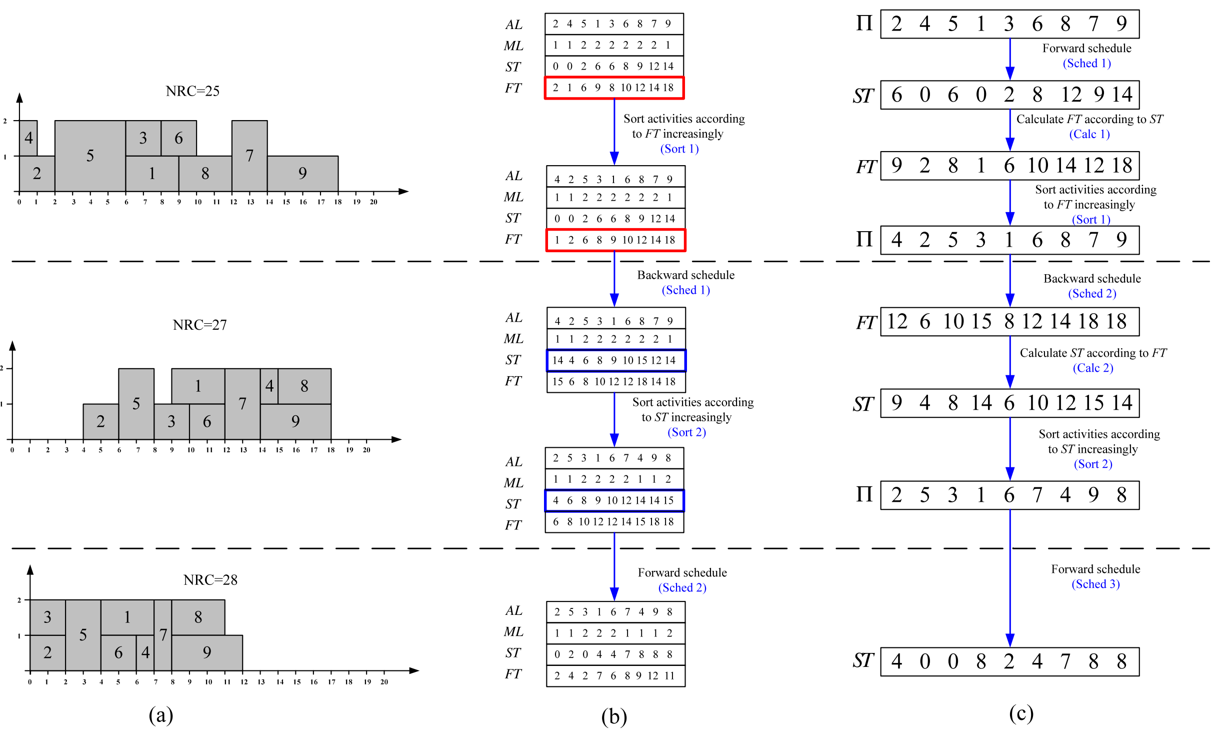

NRC=25
NRC=27
NRC=28
AL
ML
ST
FT
Π
Forward schedule (Sched 1)
Calculate FT according to ST (Calc 1)
Sort activities according to FT increasingly (Sort 1)
Backward schedule (Sched 1)
Backward schedule (Sched 2)
Calculate ST according to FT (Calc 2)
Sort activities according to ST increasingly (Sort 2)
Forward schedule (Sched 2)
Forward schedule (Sched 3)
(a)
(b)
(c)

Fig. 10. A single iteration step of the MFBI.

In this paper, we propose a multi-mode version forward–backward improvement (MFBI) for the MRCPSP. The MFBI is based on the MSSGS. Two characters make the MFBI a promising way to improve the schedules: first, the MFBI can improve the fitness value not only by changing the sequence of AL but also by changing the modes of activities with the help of the MSSGS; second, the MFBI can operate on infeasible representation, even changing it to a feasible one. With these characters, the MFBI can be easily incorporated into the SFLA to improve the solution quality.

The finish time list (part 4 of the EMAL) of a forward schedule determines the activity priorities for the next backward schedule. Similarly, the start time list (part 3 of the EMAL) of a backward schedule determines the activity priorities for the forward schedule. In Fig. 10, a single iteration step is used to illustrate procedure of the MFBI.

First, the multi-mode backward improvement is used by shifting each activity (in the descent order of activity finish time) to right as much as possible. Activity 9 and activity 7 cannot be scheduled later. Activity 8 can be right shifted to start at time 15. Activity 6 can be shifted two time units and start at time 10. Since right shifting activity 8 makes some additional resources available, activity 1 can be shifted three time units to start at time 9. Activity 3 can be right shifted two time units. With the help of the MSSGS, activity 5 is changed to mode 1, and its duration reduces to 2. The NRC also increases from 25 to 27. So activity 5 starts at time 6 and activity 2 starts at time 4. Finally, activity 4 is shifted to time 14. In this way, we obtain a schedule with makespan equals to 14. Further improvement of the schedule is possible by shifting activities to left as much as possible. As illustrated in Fig. 10(a), this process reduces the makespan by two further time unit, and increases the NRC from 27 to 28. Clearly, the MFBI is of certain ability to improve the quality of the individual and guarantee the precedence feasibility of the generated individual.

Table 2
Combinations of parameter values.

Parameters	Factor level				
	1	2	3	4	5
Popsize	100	200	300	400	500
sfla_m	1	2	4	5	10
sfla_N	1	2	5	10	20
sfla_q	2	4	6	8	10
sfla_Pper	0.1	0.3	0.5	0.7	0.9
SGS_Pper	0.1	0.3	0.5	0.7	0.9

With the help of the EMAL, the MFBI procedure can be speeded up. The speed-up procedure is illustrated in Fig. 10(b) and (c).

It can be seen from Fig. 10(b) that only 2 quick sort and 2 serial SGS operations are needed to complete a single iteration step of the MFBI based on the EMAL. However, 2 quick sort, 3 serial SGS, and 2 computing *FT* (*ST*) operations are needed if based on AL (Fig. 10(c)). Since the time complexities of quick sort, serial SGS, and computing *FT* (*ST*) are $O(n\log n)$, $O(n^2)$, and $O(n)$, respectively, the computational time of the MFBI with the EMAL will be reduced to about two thirds of the computational time of the FBI with activity list and mode list. The SFLA does not operate on AL but on the EMAL representation. This character makes it more convenient and effective to solve the MRCPSP.

5. Computational experiments

5.1. Test problems

We code and run the procedure in Visual C++ 2005 on IBM ThinkPad T61 with a Core 2 T7500 2.2 GHz processor and use the standard MRCPSP test dataset J10, J12, J14, J16, J18, J20 and J30 of the well-known PSPLIB. The PSPLIB are generated by the problem generator ProGen designed by Kolisch and Sprecher [17]. The optimal solutions for J10–20 sets are available, however no optimal solutions for J30 set have been found. Each dataset contains 640 instances, some of which are infeasible. We exclude the infeasible instances from our tests. Hence, we have 536 instances for J10, 547 instances for J20, 551 instances for J14, 550 instances for J16, 552 instances for J18, and 554 instances for J20. For the set J30, only 552 instances have found a feasible solution. Each instance of J10–20 and J30 contains three modes, two renewable and two non-renewable resources.

5.2. Parameters setting

First, we use Taguchi method of design of experiment (DOE) to determine a set of suitable parameters for the SFLA. Since the J20 is the hardest dataset of J10–20 sets whose optimal solutions are known, we choose the dataset J20 to carry out the DOE test.

The SFLA contains six key parameters: the population size of each generation (*popsize*), the number of memeplexes (*sfla_m*), the evolution steps (*sfla_N*), the number of individuals in each submemeplex (*sfla_q*), the MPBLS acceptation rate (*sfla_Pper*), and the threshold of MSSGS (*SGS_Pper*). Combinations of different parameter values are shown in Table 2.

For each instance, we set the maximum number 50,000 of the schedules as the stopping condition of the SFLA. The average response variable (*ARV*) value is the following average deviation value for *R* instances, respectively.

Table 3
Orthogonal table for the SFLA.

Experiment number	Factors						ARV
	popsize	*sfla_m*	*sfla_N*	*sfla_q*	*Sfla_Pper*	*SGS_Pper*	
1	1	1	1	1	1	1	0.022162
2	1	2	2	2	2	2	0.012031
3	1	3	3	3	3	3	0.012235
4	1	4	4	4	4	4	0.006690
5	1	5	5	5	5	5	0.006473
6	2	1	2	3	4	5	0.011954
7	2	2	3	4	5	1	0.028620
8	2	3	4	5	1	2	0.009931
9	2	4	5	1	2	3	0.005576
10	2	5	1	2	3	4	0.005509
11	3	1	3	5	2	4	0.011376
12	3	2	4	1	3	5	0.006678
13	3	3	5	2	4	1	0.028301
14	3	4	1	3	5	2	0.014720
15	3	5	2	4	1	3	0.008561
16	4	1	4	2	5	3	0.013229
17	4	2	5	3	1	4	0.006237
18	4	3	1	4	2	5	0.005523
19	4	4	2	5	3	1	0.024312
20	4	5	3	1	4	2	0.012031
21	5	1	5	4	3	2	0.019594
22	5	2	1	5	4	3	0.011146
23	5	3	2	1	5	4	0.006210
24	5	4	3	2	1	5	0.007401
25	5	5	4	3	2	1	0.021513

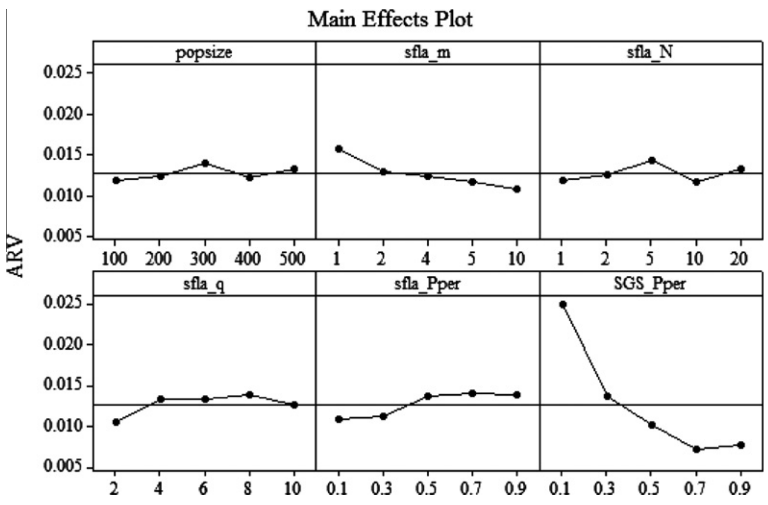


Fig. 11. Factor level trend of the SFLA.

$$ARV = \frac{1}{R}\sum_{i=1}^{R}\frac{(Makespan_i - OPT_i)}{OPT_i} \tag{13}$$

where $Makespan_i$ is the makespan of the ith instance in J20 obtained by the SFLA; OPT_i is the optimal makespan of ith instance in J20; R is the number of feasible instants in J20 (i.e. $R = 547$).

According to the number of parameters and factor levels, we choose the orthogonal array $L_{25}(5^6)$. That is, the total number of treatment is 25, the number of parameters is 6, and the number of factor levels is 5. The orthogonal array for J20 is listed in Table 3.

According to the orthogonal table, we illustrate the trend of each factor level in Fig. 11. Then, we can figure out how much the ARV value changes for each parameter to analyze their significance rank. The results are listed in Table 4.

From Table 4, it can be seen that: *SGS_Pper* is the most significant parameter among the six parameters (Delta = 0.017777); *sfla_m* is the second most significant parameter (Delta = 0.004846); the other four parameters are not very significant (Delta < 0.0033). So, it is concluded that most parameters of proposed SFLA have slight impact on the performance of the algorithm. As a result, the proposed SFLA can be used without too much trial and error for parameters tuning.

According to the factor level trend, the best combination of parameter values for the proposed SFLA is listed in Table 5.

5.3. Impact of different operators of SFLA

There are three unique operators in the proposed SFLA. First, the whole population is partitioned into different memeplexes that evolve simultaneously. This operator is the key operator of the SFLA that introduces the memetic evolution into the social behavior-based PSO algorithm. If the number of memeplexes equals to 1 (i.e. *sfla_m* = 1), the SFLA can be regarded as a special PSO algorithm. Second, the MPBLS is adopted to enhance the exploitation ability. Third, the MFBI is adopted to improve the solution. In this subsection, we try to analyze the impact of these operators of the SFLA.

The parameter values of Table 5 are used as default values. Table 6 shows the results of different combinations of operators with different maximum number of generated schedules as stopping conditions. Shortly after the beginning of

Table 4
Response table for ARV.

Level	*popsize*	*sfla_m*	*sfla_N*	*sfla_q*	*sfla_Pper*	*SGS_Pper*
1	0.011918	0.015663	0.011812	0.010531	0.010858	0.024982
2	0.012318	0.012942	0.012614	0.013294	0.011204	0.013661
3	0.013927	0.012440	0.014333	0.013332	0.013666	0.010149
4	0.012266	0.011740	0.011608	0.013798	0.014024	0.007204
5	0.013173	0.010817	0.013236	0.012648	0.013850	0.007606
Delta	0.002009	0.004846	0.002724	0.003266	0.003166	0.017777
Rank	6	2	5	3	4	1

Table 5
The best combination of parameter values for the SFLA.

	popsize	*sfla_m*	*sfla_N*	*sfla_q*	*sfla_Pper*	*SGS_Pper*
Parameter Value	100	10	10	2	0.1	0.7

Table 6
The impact of different operators of the SFLA.

Algorithm	Local search type	Maximum number of generated schedules					
		5000	10,000	20,000	30,000	40,000	50,000
PSO ($m = 1$)	None	2.05%	2.04%	2.04%	2.04%	2.04%	2.04%
	MPBLS	1.60%	1.41%	1.26%	1.21%	1.17%	1.05%
	MFBI	1.67%	1.62%	1.59%	1.57%	1.57%	1.57%
	MPBLS + MFBI	**1.33%**	**1.13%**	0.99%	0.91%	0.82%	0.79%
SFLA ($m = 10$)	None	2.33%	2.16%	2.09%	2.03%	2.01%	1.99%
	MPBLS	1.55%	1.36%	1.22%	1.15%	1.11%	1.02%
	MFBI	2.05%	1.92%	1.81%	1.71%	1.69%	1.67%
	MPBLS + MFBI	1.40%	**1.13%**	**0.91%**	**0.80%**	**0.74%**	**0.68%**

Note: The bold values denote the best results among the results obtained by all the algorithms.

the evolution (i.e. only after thousands of schedules) the PSO leads to better results than the SFLA. Then, the SFLA becomes superior. Hence, the PSO algorithm can be chosen when the number of generated schedules is relatively small, whereas the SFLA should be chosen when the maximum number of generated schedules is relatively large.

It can be seen from Table 6 that both the PSO and the SFLA with the MPBLS + MFBI outperform the one with only the MPBLS or the MFBI; both the PSO and the SFLA with only the MPBLS or the MFBI outperform the one without local search. As a result, it is concluded that both the MPBLS and the MFBI are effective in improving the performance of the SFLA. It also can be seen that the PSO outperforms the SFLA when only the MFBI is adopted as the local search. Contrastively, the SFLA outperforms the PSO when only the MPBLS is adopted. So, the SFLA stresses more on exploration and it needs the local search to stress more on exploitation, whereas the PSO stresses more on exploitation and it needs local search to stress more on exploration. To get satisfactory performances, the exploration and exploitation should be well balanced.

5.4. Comparisons with existing algorithms

In the literature of the MRCPSP, the maximum number of generated schedules and the maximum CPU time consumed are frequently used as the stopping conditions for comparison. According to the literature, we consider the following three kinds of stopping conditions: (1) 5000/10000/20000/30000/40000/50000 generated schedules; (2) 1 s CPU time; (3) 0.15 s CPU time per activity. We run the SFLA 20 times independently for each instance. The statistical results are listed in Tables 7 and 8, respectively. Besides, we use Av.dev., Min.dev., Max.dev., and Av.CPU for comparison, which denote the average, minimum, and maximum deviations from the lower bounds and the average CPU time, respectively. As for the lower bounds, the theoretically optimal values are used for set J10–20 and the critical-path based lower bounds are employed for set J30 since the theoretically optimal values are not available.

From Table 7, it can be seen that the deviation decreases constantly as the number of generated schedules increases for all the problem sets. It shows that our SFLA can keep finding better solutions as the population evolves. Moreover, the variances for all the problem sets are very small, which shows that the SFLA is very robust. From Table 8, similar conclusions can be drawn.

Next, we compare the SFLA with some existing algorithms based on the PSPLIB to further show the effectiveness of the SFLA. The compared algorithms include the simulated annealing developed by Józefowska et al. [12] denoted as JSA, the genetic algorithm developed by Alcaraz et al. [1] denoted as AGA, the hybrid scatter search developed by Ranjbar et al. [25] denoted as RSS, and the hybrid rank-based evolutionary algorithm developed by Elloumi and Fortemps [7] denoted as EFEA, the SA algorithm developed by Bouleimen and Lecocq [2] denoted as BLSA, the genetic algorithm developed by Hartmann [10] denoted as HGA, the truncated branch and bound developed by Sprecher and Drexl [26] denoted as SDBB, the PSO algorithm developed by Jarboui et al. [11] denoted as JPSO, and the DE algorithm developed by Damak et al. [4] denoted as DDE. The Pseudo-code and the parameters of these algorithms can be found from the references. Note that all the compared algorithms adopt activity list and mode list as their encoding type except SDBB and JPSO. The SDBB belongs to the category of branch and bound algorithms, and it does not need encoding; the encoding type of JPSO can be considered as a combination of discrete priority list and mode list.

The one-tailed *t*-test is also adopted to explain whether the SFLA is significantly better than the compared algorithms. We denote Av.dev. of the SFLA as μ and Av.dev. of the best compared algorithm as μ_0. Suppose the null hypothesis (H_0) is $\mu \geqslant \mu_0$, and the alternative hypothesis (H_1) is $\mu < \mu_0$. The t statistic can be calculated as follows:

Table 7
The statistic results of deviation from the lower bound for the SFLA.

Problem set		Stopping condition							
		5000	10,000	20,000	30,000	40,000	50,000	1 s	0.15 s/act.
J10	Av.dev. (%)	0.103	0.070	0.055	0.047	0.041	0.039	0.035	0.029
	Min.dev. (%)	0.058	0.029	0.018	0.018	0.006	0.006	0.006	0.006
	Max.dev. (%)	0.155	0.105	0.081	0.069	0.059	0.055	0.052	0.046
	Var.dev.	5.29E−08	3.67E−08	1.87E−08	1.63E−08	1.62E−08	1.50E−08	1.32E−08	1.04E−08
	Av.CPU (s)	0.069	0.139	0.280	0.420	0.561	0.701	1.000	1.800
J12	Av.dev. (%)	0.210	0.139	0.089	0.069	0.059	0.045	0.039	0.021
	Min.dev. (%)	0.147	0.070	0.035	0.024	0.016	0.016	0.016	0.007
	Max.dev. (%)	0.259	0.192	0.124	0.112	0.112	0.076	0.060	0.031
	Var.dev.	7.66E−08	8.18E−08	5.17E−08	3.62E−08	3.63E−08	1.94E−08	1.24E−08	6.23E−09
	Av.CPU (s)	0.094	0.188	0.375	0.563	0.750	0.938	1.000	2.100
J14	Av.dev. (%)	0.462	0.329	0.215	0.176	0.148	0.118	0.132	0.075
	Min.dev. (%)	0.351	0.238	0.141	0.125	0.111	0.073	0.107	0.054
	Max.dev. (%)	0.606	0.501	0.351	0.333	0.269	0.175	0.158	0.104
	Var.dev.	5.58E−07	6.38E−07	3.53E−07	3.04E−07	1.25E−07	6.15E−08	2.43E−08	2.01E−08
	Av.CPU (s)	0.125	0.248	0.495	0.741	0.988	1.235	1.000	2.410
J16	Av.dev. (%)	0.578	0.407	0.284	0.228	0.194	0.177	0.216	0.126
	Min.dev. (%)	0.480	0.325	0.227	0.185	0.137	0.127	0.168	0.080
	Max.dev. (%)	0.648	0.461	0.330	0.266	0.228	0.214	0.248	0.166
	Var.dev.	1.97E−07	1.19E−07	7.24E−08	4.25E−08	6.01E−08	5.11E−08	4.67E−08	5.39E−08
	Av.CPU (s)	0.152	0.305	0.610	0.914	1.219	1.524	1.000	2.710
J18	Av.dev. (%)	0.944	0.707	0.524	0.457	0.403	0.368	0.456	0.281
	Min.dev. (%)	0.827	0.634	0.458	0.388	0.340	0.301	0.379	0.229
	Max.dev. (%)	1.116	0.820	0.609	0.528	0.493	0.453	0.558	0.406
	Var.dev.	6.83E−07	2.42E−07	1.59E−07	1.73E−07	1.63E−07	1.55E−07	1.84E−07	1.56E−07
	Av.CPU (s)	0.193	0.385	0.769	1.154	1.539	1.923	1.000	3.000
J20	Av.dev. (%)	1.399	1.129	0.905	0.804	0.742	0.679	0.914	0.565
	Min.dev. (%)	1.180	0.905	0.708	0.603	0.530	0.466	0.724	0.422
	Max.dev. (%)	1.540	1.286	1.089	0.982	0.903	0.874	1.105	0.762
	Var.dev.	1.56E−06	1.71E−06	1.82E−06	1.93E−06	2.01E−06	2.17E−06	1.86E−06	1.56E−06
	Av.CPU (s)	0.268	0.535	1.069	1.603	2.137	2.671	1.000	3.300
J30	Av.dev. (%)	13.461	13.096	12.785	12.628	12.521	12.443	13.168	12.536
	Min.dev. (%)	13.293	12.982	12.684	12.537	12.427	12.349	13.059	12.418
	Max.dev. (%)	13.581	13.219	12.909	12.708	12.626	12.550	13.31	12.669
	Var.dev.	5.51E−07	3.44E−07	2.79E−07	2.56E−07	2.57E−07	2.09E−07	3.12E−07	5.43E−07
	Av.CPU (s)	0.506	1.013	2.026	3.039	4.052	5.065	1.000	4.810

Table 8
The statistic results of optimal rate for the SFLA.

Problem set		Stopping condition							
		5000	10,000	20,000	30,000	40,000	50,000	1 s	0.15 s/act.
J10	Ave. (%)	97.929	98.498	98.787	98.937	99.086	99.142	99.226	99.375
	Min. (%)	97.202	97.761	98.134	98.321	98.508	98.694	98.694	98.881
	Max. (%)	98.694	99.440	99.627	99.627	99.813	99.813	99.813	99.813
	Var.	1.04E−05	7.50E−06	4.94E−06	7.06E−06	3.70E−06	3.88E−06	3.10E−06	3.76E−06
J12	Ave. (%)	95.978	97.249	98.236	98.611	98.803	99.040	99.122	99.488
	Min. (%)	95.430	96.527	97.623	97.623	97.623	98.355	98.538	99.269
	Max. (%)	97.075	98.355	99.269	99.452	99.634	99.634	99.634	99.817
	Var.	1.37E−05	2.13E−05	1.94E−05	1.67E−05	1.49E−05	7.34E−06	6.54E−06	3.38E−06
J14	Ave. (%)	90.862	93.439	95.445	96.162	96.615	97.087	96.642	97.995
	Min. (%)	90.018	92.378	94.192	95.463	95.826	96.370	96.007	97.278
	Max. (%)	91.833	94.555	96.370	96.915	97.278	98.004	97.278	98.367
	Var.	3.23E−05	2.78E−05	3.64E−05	1.88E−05	1.74E−05	1.82E−05	1.20E−05	9.87E−06
J16	Ave. (%)	86.491	90.100	92.900	94.227	95.036	95.491	94.445	96.736
	Min. (%)	85.273	88.909	92.182	93.455	94.000	94.364	93.455	95.636
	Max. (%)	88.000	91.455	94.000	95.091	96.364	96.727	95.455	97.636
	Var.	5.92E−05	3.84E−05	2.10E−05	2.08E−05	3.62E−05	3.05E−05	2.70E−05	3.29E−05
J18	Ave. (%)	79.438	83.732	87.455	89.013	90.181	91.042	88.460	92.500
	Min. (%)	77.536	81.884	86.232	87.862	88.587	89.855	87.138	90.942
	Max. (%)	80.797	84.783	88.406	90.217	91.304	92.029	89.855	93.659
	Var.	9.90E−05	5.34E−05	4.49E−05	4.71E−05	4.68E−05	4.96E−05	5.50E−05	4.82E−05
J20	Ave. (%)	72.84	76.97	80.83	82.97	84.31	85.41	80.42	86.55
	Min. (%)	71.661	75.271	79.242	81.047	81.950	82.672	78.881	84.477
	Max. (%)	74.188	78.159	82.311	84.838	86.101	86.823	81.769	88.087
	Var.	6.16E−05	6.70E−05	8.22E−05	9.17E−05	9.62E−05	9.68E−05	9.17E−05	7.01E−05
J30	Ave. (%)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	Min. (%)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	Max. (%)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	Ave. (%)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a

Note: The optimal values of J30 are not available.

Table 9
Comparison with EFEA, RSS, AGA and JSA (5000 generated schedules as stopping condition).

Av. dev (%)	J10	J12	J14	J16	J18	J20
SFLA	**0.10** ($p < 0.0005$)	**0.21** ($p < 0.0005$)	**0.46** ($p < 0.0005$)	**0.58** ($p < 0.0005$)	**0.94** ($p < 0.0005$)	**1.40** ($p < 0.0005$)
EFEA	0.14	0.24	0.77	0.91	1.30	1.62
RSS	0.18	0.65	0.89	0.95	1.21	1.64
AGA	0.24	0.73	1.00	1.12	1.43	1.91
JSA	1.16	1.73	2.6	4.07	5.52	6.74

Note: The bold values denote the best results among the results obtained by all the algorithms.

Table 10
Comparison with EFEA and BLSA (1 s CPU time as stopping condition).

J	Av. dev. (%)			Optimal rate (%)		
	SFLA[a]	EFEA[b]	BLSA[c]	SFLA[a]	EFEA[b]	BLSA[c]
10	**0.035** ($p < 0.0005$)	0.08	0.21	**99.2**	98.5	96.3
12	**0.039** ($p < 0.0005$)	0.12	0.19	**99.1**	97.5	91.2
14	**0.13** ($p < 0.0005$)	0.42	0.92	**96.6**	90.6	82.6
16	**0.21** ($p < 0.0005$)	0.60	1.43	**94.4**	85.9	72.8
18	**0.46** ($p < 0.0005$)	0.71	1.85	**88.5**	83.1	69.4
20	**0.91** ($p < 0.0005$)	1.02	2.10	**80.4**	76.7	66.9

Note: The bold values denote the best results among the results obtained by all the algorithms.
[a] T7500 2.2 GHz, time limit 1 s.
[b] Pentium 3.00 GHz, time limit 1 s.
[c] Pentium 100 MHz, time limit five times the instance size (in seconds).

Table 11
Comparison with HGA and SDBB (1s CPU time as stopping condition).

J	Av. dev.			Feasible rate			Optimal rate		
	SFLA	HGA	SDBB	SFLA	HGA	SDBB	SFLA	HGA	SDBB
10	0.035	0.06	**0.00**	100	100	100	**99.2**	98.7	100
12	**0.039** ($p < 0.0005$)	0.14	0.12	100	100	100	**99.1**	97.3	98.2
14	**0.13** ($p < 0.0005$)	0.44	1.46	100	100	99.6	**96.6**	89.8	85.7
16	**0.21** ($p < 0.0005$)	0.59	3.81	100	100	99.5	**94.4**	87.8	69.5
18	**0.46** ($p < 0.0005$)	0.99	7.48	100	100	98.0	**88.5**	78.3	57.4
20	**0.91** ($p < 0.0005$)	1.21	11.51	100	100	96.4	**80.4**	73.3	47.3
30	**13.2** ($p < 0.0005$)	16.93	57.22	86.1	**86.3**	55.8	n/a	n/a	n/a

Note: The optimal value of J30 is non-available. The bold values denote the best results among the results obtained by all the algorithms.

Table 12
Comparison with EFEA, JPSO and DDE (0.15 s/activity as stopping condition).

J	Av. dev.				Optimal rate			
	SFLA	EFEA	JPSO	DDE	SFLA	EFEA	JPSO	DDE
10	**0.03** ($p < 0.4$)	**0.03**	**0.03**	0.09	**99.38**	99.25	99.25	99.3
12	**0.02** ($p < 0.0005$)	0.05	0.09	0.11	**99.49**	98.72	98.47	99.3
14	**0.08** ($p < 0.0005$)	0.26	0.36	0.34	**98.00**	93.93	91.11	97.6
16	**0.13** ($p < 0.0005$)	0.25	0.44	0.42	**96.74**	93.09	85.91	96.38
18	**0.28** ($p < 0.0005$)	0.45	0.89	0.59	92.50	88.22	79.89	**94.43**
20	**0.57** ($p < 0.4$)	0.58	1.10	0.70	86.55	84.81	74.19	**91.75**

Note: The bold values denote the best results among the results obtained by all the algorithms.

$$t = \frac{\mu - \mu_0}{s/\sqrt{n}} \tag{14}$$

where n is the number of samples that is the number of independent running tries of the SFLA, and $s = \sqrt{Var.dev.}$ is the standard deviation of samples that can be calculated according to the Var.dev. in Table 7. According to the value obtained by the above equation, we can further get the p-value from the corresponding lookup table that is the probability to make a mistake if we accept H_1 as true.

In Table 9, we compare the SFLA with EFEA, RSS, AGA, and JSA. The stopping condition for this comparison is 5000 generated schedules. It can be seen from the results that our SFLA outperforms the four existing algorithms for all the sets.

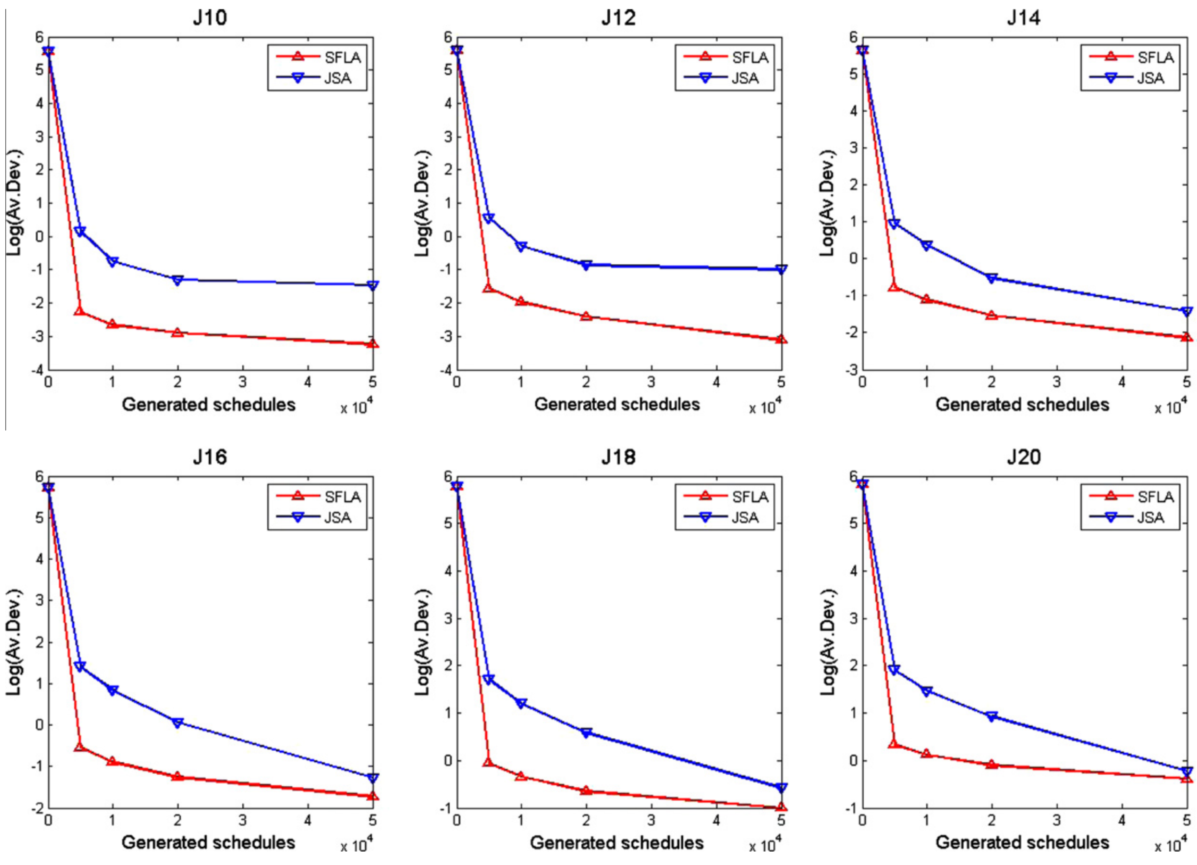


Fig. 12. Comparison between SFLA and JSA with different number of generated schedules.

In Table 10, we compare the SFLA with EFEA and BLSA. The stopping condition for this comparison is 1s CPU time in each run. It can be seen from Table 10 that our SFLA has not only higher optimal rate than EFEA and BLSA, but also lower average deviation with less computation effort.

In Table 11, we compare the SFLA with HGA and SDBB. The stopping condition for this comparison is also 1s CPU time. As a branch and bound algorithm, SDBB can solve J10 optimally. However, as the problem scale increases, the performance of SDBB deteriorates rapidly. Even for J12 which only has 2 activities more than J10, SFLA performs much better than SDBB. It can be seen from Table 11 that our SFLA outperforms HGA and SDBB for all the sets except J10, while only the feasible rate of our SFLA for J30 is slightly lower than that of HGA.

In Table 12, we compare the SFLA with EFEA, JPSO and DDE. The stopping condition for this comparison is 0.15 s CPU time per activity. It can be seen from Table 12 that our SFLA not only has lower average deviation but also has better optimal rate than EFEA, JPSO and DDE for set J12–16. For J10, SFLA, EFEA and JPSO have a similar performance in average deviation, but SFLA performs better in optimal rate. For J18 and 20, DDE performs better than our SFLA in optimal rate but worse in average deviation. However, there is no algorithm can perform better than our SFLA in both terms.

In Fig. 12, we compare SFLA with JSA with different number of generated schedules. Due to the large range of Av.Dev., we adopt Log (Av.Dev.) as the base for comparison. It can be seen that SFLA outperforms JSA in all the cases. Shortly after the beginning of the evolution, SFLA converges faster than JSA. It shows the initial method adopted in SFLA is effective. For the set J10, J12, it seems that JSA is trapped in a local minimum, since its convergence speed decreases evidently after generating 20000 schedules. Contrastively, SFLA not only has better performance but also converges faster.

In addition, according to the *p*-values obtained by the *t*-test given in Tables 9–12, it can be seen that SFLA outperforms other algorithms significantly in most of the cases ($p < 0.0005$) with only 3 exceptions. As a branch and bound algorithm, SDBB can solve J10 optimally in 1 s CPU time. However, as the scale of problem increases, the performance of SDBB drops rapidly. Even for J12 which only has 2 activities more than J10, SFLA performs much better than SDBB. Our SFLA cannot outperform EFEA and JPSO overwhelmingly for the set J10 and J20 with 0.15s/activity as stopping condition in which the largest computational effort is allowed. The reason is that it is very hard to find a better solution when you have found a good enough solution.

According to the above comparisons between the SFLA and the nine existing algorithms, it can be concluded that our proposed SFLA is effective in solving the MRCPSP.

6. Conclusion

This was the first reported work to develop a shuffled frog-leaping algorithm for solving the multi-mode resource-constrained project scheduling problem. By using an encoding scheme based on the extended multi-mode active list and a decoding scheme based on the multi-mode serial SGS, we generalized the SFLA to solve the MRCPSP. Moreover, by applying the regret based sampling method and LFT priority rule, the population was well initialized; by adopting simplified two-point crossover and exchanging information during shuffling and partitioning process, the population evolved with certain diversity in an interactive way; by utilizing the combined local search with multi-mode permutation based local search and multi-mode forward–backward improvement, the exploitation was enhanced as well. We determined a set of suitable parameter values according to the DOE test, and then compared the SFLA with some existing algorithms based on the well-known PSPLIB. Computational results and comparisons demonstrated the effectiveness of the proposed SFLA. The further work is to develop more efficient algorithm by exploring structure information of the MRCPSP, such as network complexity, resource factor, and resource strength, and to study multi-objective algorithms for the problems with multiple objectives.

Acknowledgments

The authors sincerely thank the Editor-in-Chief and the reviewers for providing the constructive and helpful comments to improve the quality of this paper. The authors also would like to thank Prof. Rainer Kolisch of the Technical University of Munich for the helpful discussion. This research is partially supported by National Science Foundation of China (70871065 and 60834004) and Program for New Century Excellent Talents in University (NCET-10-0505) as well as Doctoral Program Foundation of Institutions of Higher Education of China (20100002110014).

References

[1] J. Alcaraz, C. Maroto, R. Ruiz, Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms, Journal of Operation Research Society 54 (2003) 614–626.

[2] K. Bouleimen, H. Lecocq, A new efficient simulated annealing algorithm for resource-constrained project scheduling problem and its multiple mode version, European Journal of Operational Research 149 (2003) 268–281.

[3] G. Chung, K.E. Lansey, Application of the shuffled frog leaping algorithm for the optimization of a general large-scale water supply system, Water Resource Management 23 (2009) 797–823.

[4] N. Damak, B. Jarboui, P. Siarry, T. Loukil, Differential evolution for solving multi-mode resource-constrained project scheduling problems, Computers & Operations Research 36 (2009) 2653–2659.

[5] E.W. Davis, J.H. Patterson, A comparison of heuristic and optimum solutions in resource-constrained project scheduling, Management Science 21 (1975) 944–955.

[6] E. Elbeltagia, T. Hegazyb, D. Griersonb, Comparison among five evolutionary-based optimization algorithms, Advanced Engineering Informatics 19 (2005) 43–53.

[7] S. Elloumi, P. Fortemps, A hybrid rank-based evolutionary algorithm applied to multi-mode resource-constrained project scheduling problem, European Journal of Operational Research 205 (2010) 31–41.

[8] M. Eusuff, K. Lansey, F. Pasha, Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization, Engineering Optimization 38 (2006) 129–154.

[9] M. Eusuff, K. Lansey, Optimization of water distribution network design using the shuffled frog leaping algorithm, Journal of Water Resources Planning and Management 129 (2003) 210–225.

[10] S. Hartmann, Project scheduling with multiple modes: a genetic algorithm, Annals of Operations Research 102 (2001) 111–135.

[11] B. Jarboui, N. Damak, P. Siarry, A. Rebai, A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems, Applied Mathematics and Computation 195 (2008) 299–308.

[12] J. Józefowska, M. Mika, R. Różycki, G. Waligóra, J. Węglarz, Simulated annealing for multi-mode resource-constrained project scheduling, Annals of Operations Research 102 (2001) 137–155.

[13] R. Kolisch, A. Drexl, Local search for nonpreemptive multi-mode resource-constrained project scheduling, IIE Transactions 29 (1997) 987–999.

[14] R. Kolisch, S. Hartmann, Heuristic algorithms for solving the resource-constrained project scheduling problem: classification, computational, analysis, in: J. Węglarz (Ed.), Project Scheduling: Recent Models, Algorithms and Applications, Kluwer Press, Berlin, 1999, pp. 147–178.

[15] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: theory and computation, European Journal of Operational Research 90 (1996) 320–333.

[16] R. Kolisch, Project Scheduling Under Resource Constraints: Efficient Heuristics for Several Problem Classes, Springer Press, Heidelberg, 1995.

[17] R. Kolisch, A. Sprecher, PSPLIB-a project scheduling problem library, European Journal of Operational Research 96 (1997) 205–216.

[18] V.J. Leon, B. Ramamoorthy, Strength and adaptability of problem space based neighborhoods for resource-constrained scheduling, OR Spectrum 17 (1995) 173–182.

[19] K.Y. Li, R.J. Willis, An iterative scheduling technique for resource constrained project scheduling, European Journal of Operational Research 56 (1992) 370–379.

[20] C.C. Palmer, A. Kershenbaum, Representing trees in genetic algorithms, in: Proceedings of the first IEEE International conference on Evolutionary Computation, 1994, pp. 376–384.

[21] V. Van Peteghem, M. Vanhoucke, An artificial immune system for the multi-mode resource-constrained project scheduling problem, in: EvoCOP 2009, Springer, Berlin, Tübingen, Germany, 2009, pp. 85–96.

[22] A. Rahimi-Vahed, M. Dangchi, H. Rafiei, E. Salimi, A novel hybrid multi-objective shuffled frog-leaping algorithm for a bi-criteria permutation flow shop scheduling problem, International Journal of Advanced Manufacturing Technology 41 (2009) 1227–1239.

[23] A. Rahimi-Vahed, A.H. Mirzaei, A hybrid multi-objective shuffled frog-leaping algorithm for a mixed-model assembly line sequencing problem, Computers & Industrial Engineering 53 (2007) 642–666.

[24] A. Rahimi-Vahed, A.H. Mirzaei, Solving a bi-criteria permutation flow-shop problem using shuffled frog-leaping algorithm, Soft Computing 12 (2008) 435–452.

[25] M. Ranjbar, B. De Reyck, F. Kianfar, A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling, European Journal of Operational Research 193 (2009) 35–48.

[26] A. Sprecher, A. Drexl, Solving multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm, European Journal of Operational Research 107 (1998) 431–450.
[27] A. Sprecher, S. Hartmann, A. Drexl, An exact algorithm for project scheduling with multiple modes, OR Spektrum 19 (1997) 195–203.
[28] T. Wauters, K. Verbeeck, G.V. Berghe, P. De Causmaecker, A multi-agent learning approach for the multi-mode resource-constrained project scheduling problem, in: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, 2009.
[29] H. Zhang, C.M. Tam, H. Li, Mulitmode project scheduling based on particle swarm optimization, Computer-Aided Civil and Infrastructure Engineering 21 (2006) 93–103.