

Research Article

Shuffled Frog Leaping Algorithm for Preemptive Project Scheduling Problems with Resource Vacations Based on Patterson Set

Yi Han,^{1,2,3} Ikou Kaku,⁴ Jianhu Cai,² Yanlai Li,⁵ Chao Yang,¹ and Lili Deng²

¹ School of Management, Huazhong University of Science and Technology, Wuhan 430074, China

² College of Economics and Management, Zhejiang University of Technology, Hangzhou 310023, China

³ Technological Innovation and Enterprise Internationalization Research Center, Zhejiang Provincial Key Research Institute of Philosophy & Social Sciences, Hangzhou 310023, China

⁴ Department of Environmental and Information Studies, Tokyo City University, Yokohama 224-0015, Japan

⁵ School of Logistics, Southwest Jiaotong University, Chengdu 610031, China

Correspondence should be addressed to Jianhu Cai; hzcjh@yahoo.com

Received 24 July 2013; Revised 29 August 2013; Accepted 30 August 2013

Academic Editor: Sabri Arik

Copyright © 2013 Yi Han et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper presents a shuffled frog leaping algorithm (SFLA) for the single-mode resource-constrained project scheduling problem where activities can be divided into equant units and interrupted during processing. Each activity consumes 0–3 types of resources which are renewable and temporarily not available due to resource vacations in each period. The presence of scarce resources and precedence relations between activities makes project scheduling a difficult and important task in project management. A recent popular metaheuristic shuffled frog leaping algorithm, which is enlightened by the predatory habit of frog group in a small pond, is adopted to investigate the project makespan improvement on Patterson benchmark sets which is composed of different small and medium size projects. Computational results demonstrate the effectiveness and efficiency of SFLA in reducing project makespan and minimizing activity splitting number within an average CPU runtime, 0.521 second. This paper exposes all the scheduling sequences for each project and shows that of the 23 best known solutions have been improved.

1. Introduction

Project scheduling problem (PSP), arising from a construction project of a bridge in 1950s, is a classical optimization problem. It is one of the most challenging problems in the field of scheduling and combinatorial optimization. Many real-world scheduling problems such as course scheduling, sports timetabling, railway and airline scheduling, software development, construction engineering problem, and maritime disaster rescue problem can be categorized as project scheduling problem [1–4]. As a generalization of job shop scheduling problem, PSP, especially with constrained renewable and nonrenewable resources, is known to be strongly NP-hard [5, 6]. PSP's objective, which is single or multiple, can be characterized by makespan minimization, deadline satisfaction, total cost minimization, and so forth. Activities

can be preemptive or not. Resources can be unconstrained, renewable, and nonrenewable. The project can be single or multiple with one or more executive modes.

For traditional PSP, critical path method (CPM) and program evaluation and review technique (PERT) were developed by assuming unlimited resources. However, in real-life circumstances, only a fixed amount of resources is available and the cost of acquiring additional resources is very high [7, 8]. Therefore the resource-constrained project scheduling problem (RCPSP), which is NP-hard, began to receive more and more attention afterwards. RCPSP belongs to the scheduling of activities of a project to minimize the total duration under the precedence relations and renewable resources (materials, machines, or manpower). Nowadays, even if there are many extensions and variations of RCPSP such as multiproject problem, multimode problem, and

tardiness punishment problem, they can be easily handled by adding dummy activities (starting node and ending node) for multiproject problem or by slightly modifying those solving methods for traditional RCPSP [9].

In the early age, most researchers and practitioners used to utilize exact methods for solving the RCPSP. These methods include zero-one programming, dynamic programming, implicit enumeration, and cutting plane method. Since the RCPSP is NP-hard, most of the proposed exact methods can only handle problems with at most 60 activities and slight resource constraints [5].

Since the exact methods had some limitations, people invented heuristics to solve optimization problems for acceptable solutions [10–12]. Yan et al. [13] proposed a heuristic algorithm and tested it with 21–50 activities problem. Golenko-Ginzburg and Gonik [14] adopted a heuristic for network project scheduling with random activity durations. Chtourou and Haouari [15] hired a two-stage-priority-rule-based algorithm for RCPSP. Vanhoucke [16] proposed a fast tracking method for RCPSP with preemptive activities. Xu et al. [17] developed an algorithm based on resource push-pull technology for multiproject RCPSP. Krüger and Scholl [18] programmed a heuristic for RCPSP with multiple projects. Kolisch [19] used efficient priority rules for RCPSP. Valls et al. [20] presented a critical activity reordering heuristic for RCPSP.

Compared with heuristics, metaheuristic could reach better solutions within acceptable runtime. Gonçalves et al. [9] proposed a genetic algorithm (GA) for scheduling problem with multiple projects. Mendes et al. [21] adopted a random key-based GA for the RCPSP. Zhang et al. [7] solved a 25-activity project with particle swarm optimization (PSO) and compared PSO with GA to prove its superiority without presenting the runtime. Pan et al. [22] studied Patterson sets-based RCPSP via a Tabu search (TS) algorithm. Yamashita et al. [23] adopted scatter search (SS) for project scheduling with resource availability cost and proved the effect of SS by 20- and 30-activity project. Mobini et al. [5] used an artificial immune algorithm for RCPSP. Agarwal et al. [6] proposed a neurogenetic algorithm for RCPSP. Lova et al. [24] solved multimode RCPSP with hybrid genetic algorithm. Kim et al. [25] employed a hybrid genetic algorithm for multiproject RCPSP. Chen et al. [26] used hybrid ant colony optimization (ACO) and SS for RCPSP. Ranjbar [27] solved the RCPSP with filter-and-fan approach. Deng et al. [28] presented a hybrid ACO for RCPSP considering preemptive assumption but they did not show the runtime for each case. Through mapping the solution space to quantum space, Huang [29] solved RCPSP without presenting the runtime. Peteghem and Vanhoucke [30] considered the preemptive multimode RCPSP with a genetic algorithm. Pan and Jiao [8] presented a multiagent social evolutionary algorithm for RCPSP. Ying et al. [31] designed a hybrid genetic algorithm for RCPSP with multiple projects. Montoya-Torres et al. [32] and Valls et al. [33] implemented different genetic algorithms for RCPSP. Bouleimen and Lecocq [34] and Elloumi and Fortemps [35] invented a simulated annealing and a hybrid rank-based evolutionary algorithm, respectively, for RCPSP with multiple modes. Al-Fawzan and

Haouari [36] resorted to a tabu search for biobjective RCPSP.

Since it is of no significance to comment on all the papers related to the RCPSP, readers can refer to some survey papers for more details. Hartmann and Briskorn [37] fully summed up variants and extensions of the RCPSP. Fang and Wang [38] showed a survey paper to conclude the RCPSP in Chinese. Kolisch and Padman [39] finished a survey of deterministic project scheduling problems in 2001. Ballestín and Blanco [40] presented theoretical and practical fundamentals for RCPSP with multiple objectives. Herroelen and Leus [41] pointed out the research potentials for PSP under uncertainty. Wglarz et al. [42] surveyed PSP with finite or infinite number of activity processing modes.

This paper addresses the single project RCPSP with preemptive activities (RCPSP_PA). This problem is firstly proposed by Buddhakulsomsiri and Kim [43] in 2006 and thereafter Buddhakulsomsiri and Kim [44] designed a heuristic for it. Afterwards, GAs, ACO, and particle swarm optimization (PSO) were adopted as solving tools [28, 30, 45, 46]. To date, only several papers focused on preemptive-activity-based RCPSP. Shuffled frog leaping algorithm (SFLA), proposed by Eusuff and Lansey [47], is a population-based novel and effective metaheuristics computing method, which received increasing focuses from academic and engineering optimization fields in recent years. Since SFLA is a combination of memetic algorithm (MA) with strong local search (LS) ability and particle swarm optimization (PSO) with good global search (GS) capability, it is of strong optimum-searching power and easy to be implemented [48–53]. We adopted SFLA as a solving tool and proved SFLA's efficiency and effectiveness by 110-case Patterson set.

This paper is organized as follows. Section 2 is presenting the problem description. Section 3 shows some basic knowledge on SFLA. Section 4 is dedicated to SFLA for RCPSP. Section 5 presents the experimental computation, and Section 6 sums up the paper.

2. Problem Description

The RCPSP can be presented by an activity-on-node network. A project consists of N activities numbered with $i = 1, 2, \dots, N$ and M renewable resources numbered with $k = 1, 2, \dots, M$. A constant amount of R_k units of resource k is available in each time unit. The duration of an activity i is denoted by d_i (without losing generality, it is an integer in this paper). We set $d_0 = d_{N+1} = 0$, since in common sense two dummy activity nodes 0 and $N + 1$, which consume no resources, are added as start and end. The immediate precedence set of an activity i is denoted by P_i indicating that activity i cannot be processed before all activities in P_i are completed. S_i is the immediate successor set of activity i . F is the set of those processed activities. D indicates the set of those activities without precedence. f_i is the finishing time of activity i . r_{ik} is the consumption amount of activity i on each resource. $C(t)$ is the set of on-processing activities in a certain time unit t .

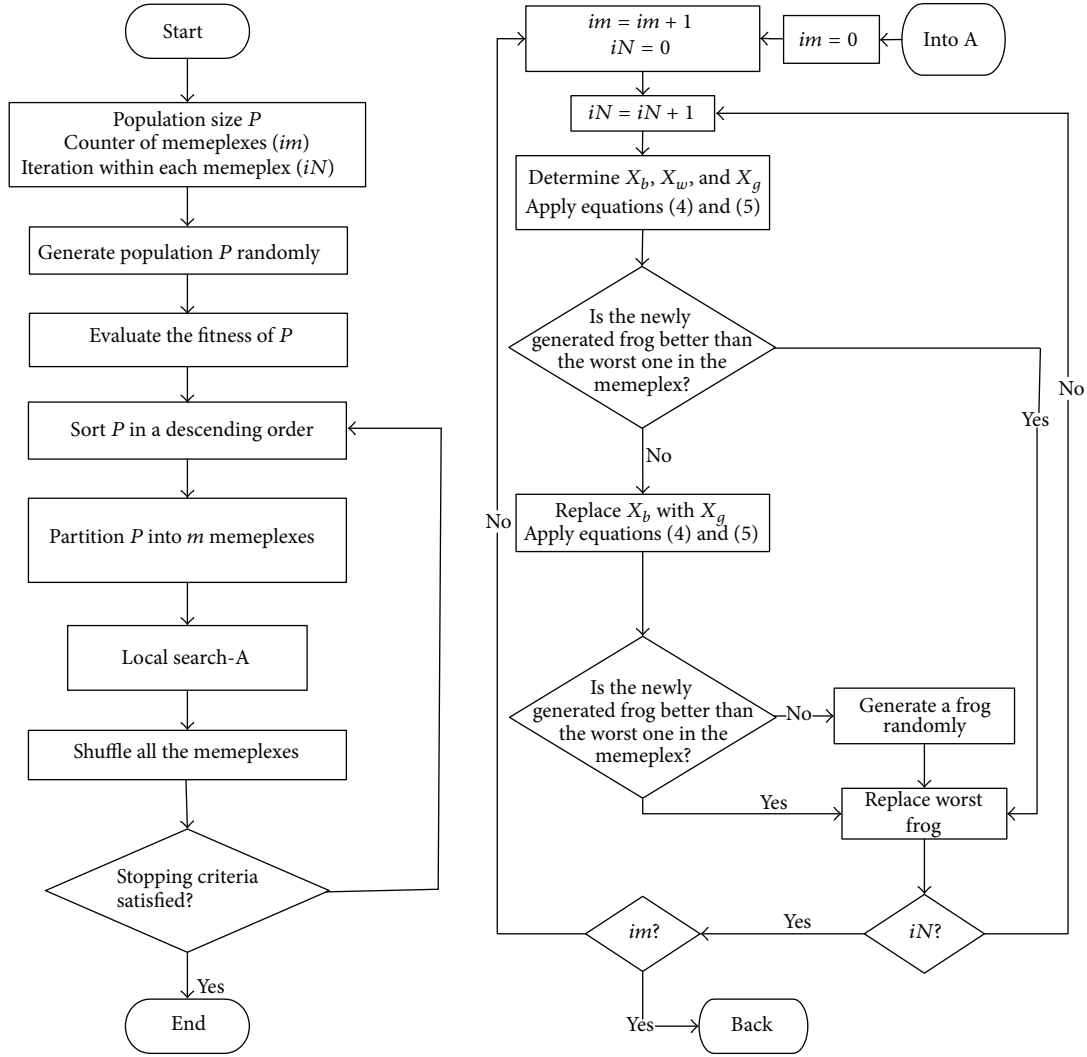


FIGURE 1: Flowchart of SFLA.

A schedule is a feasible arrangement of activities (such as $\langle 1, 2, 4, 3, 5, 7, 6, 8 \rangle$ for a 8-activity project) under the resource and precedence constraints. The objective of the RCPSP is to find a feasible schedule that minimizes the project makespan. Mathematical model can be formulated as follows [19, 28]:

$$\text{Minimize } f_{N+1} \quad (1)$$

$$\text{s.t. } f_i \leq f_j - d_j \quad \forall i \in P_j, j = 1, \dots, N+1 \quad (2)$$

$$\sum_{i \in C(t)r_{ik}} \leq R_k \quad i = 0, 1, \dots, N+1; k = 1, \dots, M. \quad (3)$$

The object function minimizes the makespan of the project. The precedence relations of two activities (i, j) (i is an immediate predecessor of j) can not be destroyed as shown in constraint (2). Constraint (3) limits the resource consumption amount within each time unit.

3. Shuffled Frog Leaping Algorithm (SFLA)

SFLA is a recently popular metaheuristic based on the memetic evolution of a group of frogs when seeking for the location that has the maximum amount of available food. Proposed in 2003, SFLA is the combination of the merits of memetic algorithm (MA) and PSO. In SFLA, the population consists of a group of frogs (solutions) that is partitioned into subsets (memplexes). Different memplexes, each performs a local search, are considered as different cultures (memes) of frogs. Examples of memes are songs, ideas, catch phrases, clothes fashions, and ways of making pots or of building arches. Frogs in each memplex experience a memetic evolution; that is, they conduct local exploration of solution space according to specific strategies which allow the transference of meme among local individuals. After a predefined number of memetic evolution steps, information is passed between memplexes in a shuffling process. The local exploration and the shuffling process are carried out alternatively until the defined convergence criterion is satisfied [54].

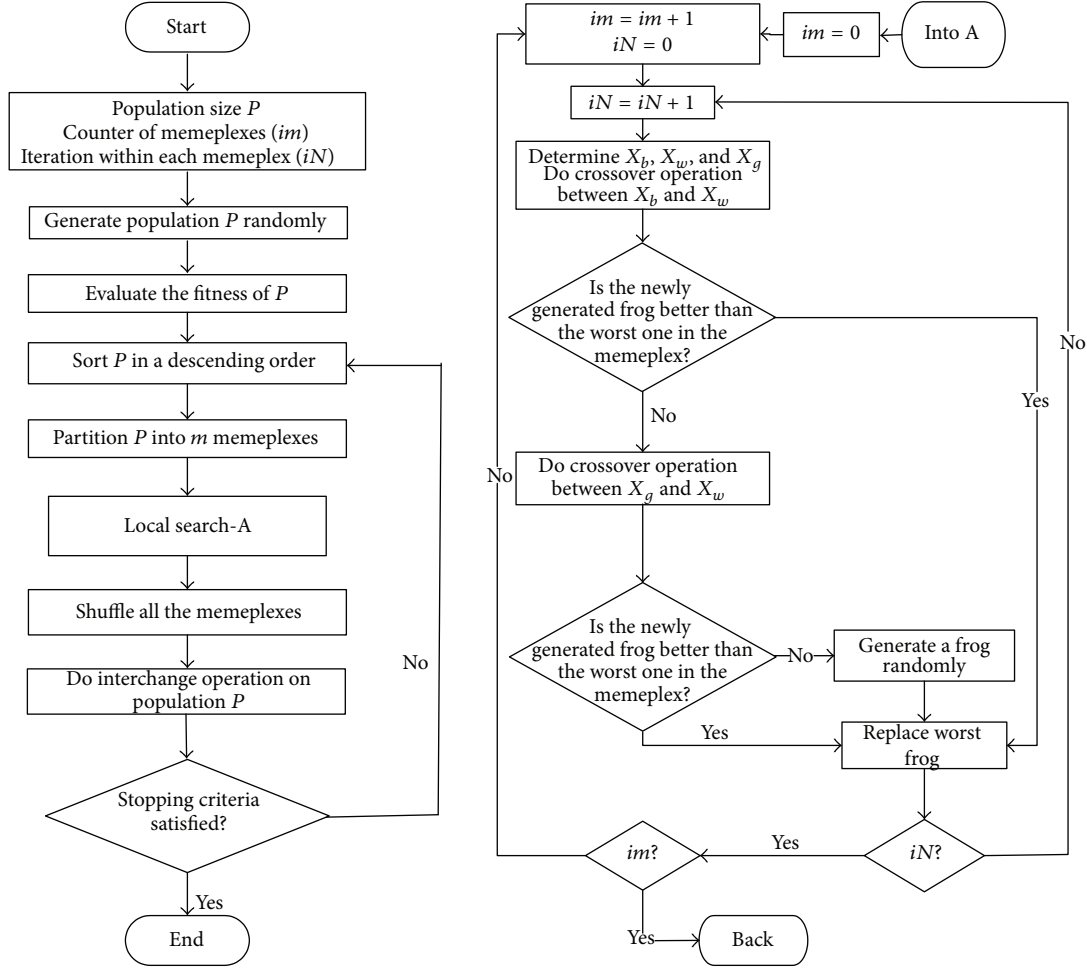


FIGURE 2: Flowchart of RCPSPPA.SFLA.

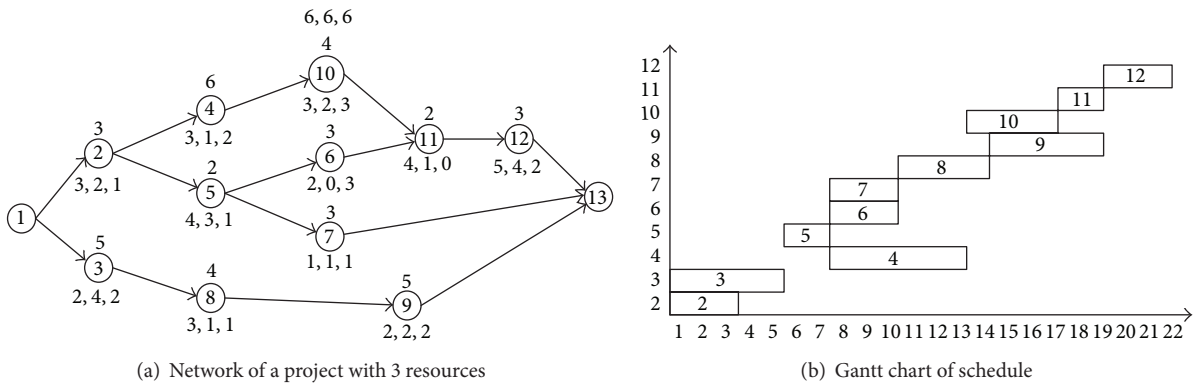


FIGURE 3: Project with 13 activities and its corresponding Gantt chart.

The SFLA is described as follows: assume that the initial population is formed by P randomly generated frogs. For L -dimensional problems (L variables), a frog i is represented as $X_i = (x_{i1}, x_{i2}, \dots, x_{iL})$. Afterwards, the frogs are sorted in a descending order according to their fitness. Then, the entire population is divided into m memplexes, each containing n

frogs (i.e., $P = m \times n$). In this process, the first frog goes to the first memplex, the second frog goes to the second memplex, frog m goes to the m th memplex, frog $m + 1$ goes back to the first memplex, and so forth [53].

Within each memplex, the frogs with the best and the worst fitness are denoted by X_b and X_w . The frog with the

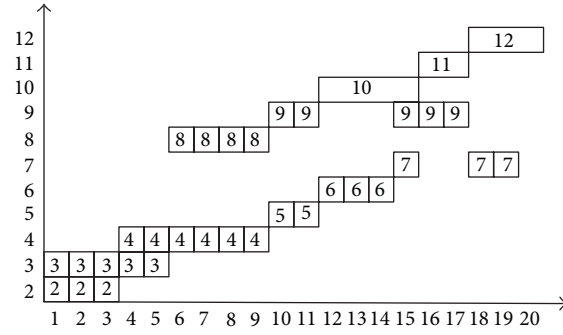


FIGURE 4: Optimal Gantt chart of RCPSP_PA.

Step 1. Initialize, set $t(i, h) = 0$ ($i = 1, 2, \dots, N$ (including two dummy nodes), $h = 1, 2, \dots, d_i$), $F = \{1\}$, $j = 1$;
 Step 2. If $|F| \leq N$, $j = j + 1$ and go to Step 3; else **Stop**;
 Step 3. Choose the j th element $q(j)$ in q ;
 Step 4. Set $ES(q(j)) = \max\{t(b, d_b) \mid b \in P_{q(j)}\} + 1$, if $|P_{q(j)}| \neq 0$; else $ES(q(j)) = 0$;
 Step 5. Set $h = 1$ and $t(q(j), h) = \min\{t \mid ES(q(j)) \leq t, r_{q(j)k} \leq R_k - \sum_{m \in C(t)} r_{mk}, k = 1, 2, \dots, M\}$;
 Step 6. Set $h = h + 1$;
 Step 7. If $h \leq d_{q(j)}$, $t(q(j), h) = \min\{t \mid t(q(j), h - 1) \leq t, r_{q(j)k} \leq R_k - \sum_{m \in C(t)} r_{mk}, k = 1, 2, \dots, M\}$ then go to Step 6; else go to Step 8;
 Step 8. $F = F \cup q(j)$, go to Step 2.

ALGORITHM 1: Decoding procedure for a feasible schedule q .

Step 1. Initialize, $S = \{1\}$, clear D ;
 Step 2. If $|F| \leq N$, go to Step 3, else **Stop** and **output** S ;
 Step 3. Construct D , $D = \{i \mid i \notin F, P_i \in F, i = 2, 3, \dots, N\}$;
 Step 4. Choose randomly an activity j from D , $S = S \cup j$; go to Step 2.

ALGORITHM 2: Feasible schedule generation procedure.

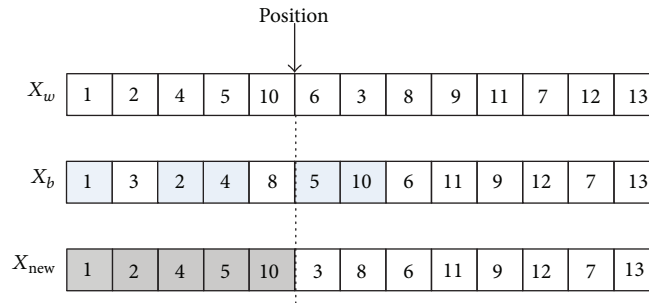


FIGURE 5: New schedule generation with crossover operation when Pos = 5.

Step 1. Initialize, $F = \Phi$, $D = \Phi$;
 Step 2. From $j = \text{position 1 to } P_1$, update F and D ;
 Step 3. $d = P_1$;
 Step 4. From d to P_2 , if $S(d)$ is not in D , search from $u = d + 1$ to P_2 until $S(u) \in D$;
 Step 5. put $S(u)$ in d and from $y = u - 1$ to d put $S(y + 1) = S(y)$;
 Step 6. add $S(d)$ to F and update D ;
 Step 7. $d = d + 1$, if $d > P_2$, **Stop**; else go to Step 4.

ALGORITHM 3: Repair strategy for interchange operation.

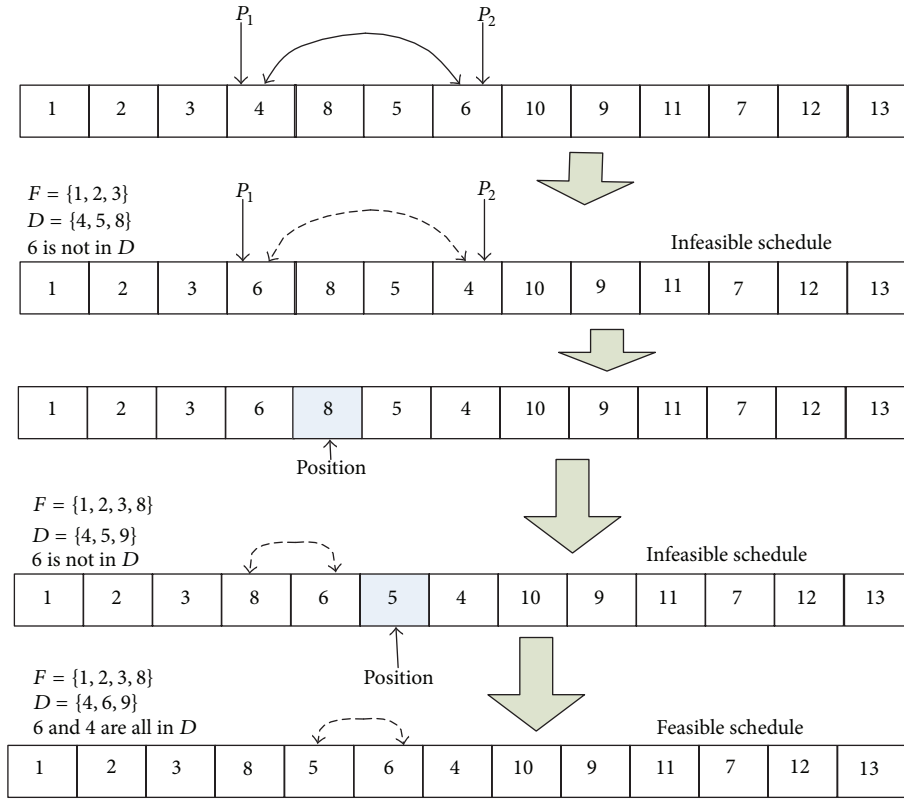


FIGURE 6: Detailed exhibition of repair procedure.

Step 1. Generate randomly two position P_1 and $P_2 \in [1, N]$;
 Step 2. **If:** swapping two activities $S(P_1)$ and $S(P_2)$ without producing infeasible schedule, go to Step 3;
Else: execute the **Repair strategy**.
 Step 3. interchange two activities' positions.

ALGORITHM 4: Procedure of interchange operation.

global best fitness is identified as X_g . Here, a process similar to PSO is applied to improve X_w (not all frogs) in each small group. The adjusting measure is as follows:

$$S = \text{rand}() \times (X_b - X_w) \quad (4)$$

$$X_{\text{new}} = X_w + S, \quad -S_{\max} \leq S \leq S_{\max}, \quad (5)$$

where $\text{rand}()$ is a random number between 0 and 1. $S_{\max}$ is the maximum change in a frog's position. If a better solution X_{new} is produced, it replaces the worst frog X_w ; otherwise, the calculations in (4) and (5) are repeated by replacing X_b with X_g in (4). If no improvement is achieved under this situation, then a new solution is randomly generated to replace X_w . Then the memplex is reordered and renewed within local search times and thereafter all the memplexes are shuffled together to exchange information and reallocated for next search process. The flowchart of SFLA can be given in Figure 1 [51].

4. SFLA for RCPSP with Preemptive Activities (RCPSP_PA_SFLA)

It is easy to tell that SFLA is an algorithm especially for continuous optimization problem. Since RCPSP is a discrete optimization problem, SFLA is adjusted by integrating crossover and interchange operations analogous to GA. The flowchart of RCPSP_PA_SFLA is shown in Figure 2.

4.1. Problem Encoding. Based on the precedence relationship, the problem is encoded as a scheduling sequence of activities, that is, $\langle 1, 2, 3, 5, 4, 6, 7, 8, 10, 9, 11, 12, 13 \rangle$ is a feasible schedule for a 13-activity project with nonpreemptive activities with 3 resources (each is 6 units) as shown in Figure 3. In Figure 3(a), the numbers below an activity are resource consuming volumes and the number above an activity is the processing time.

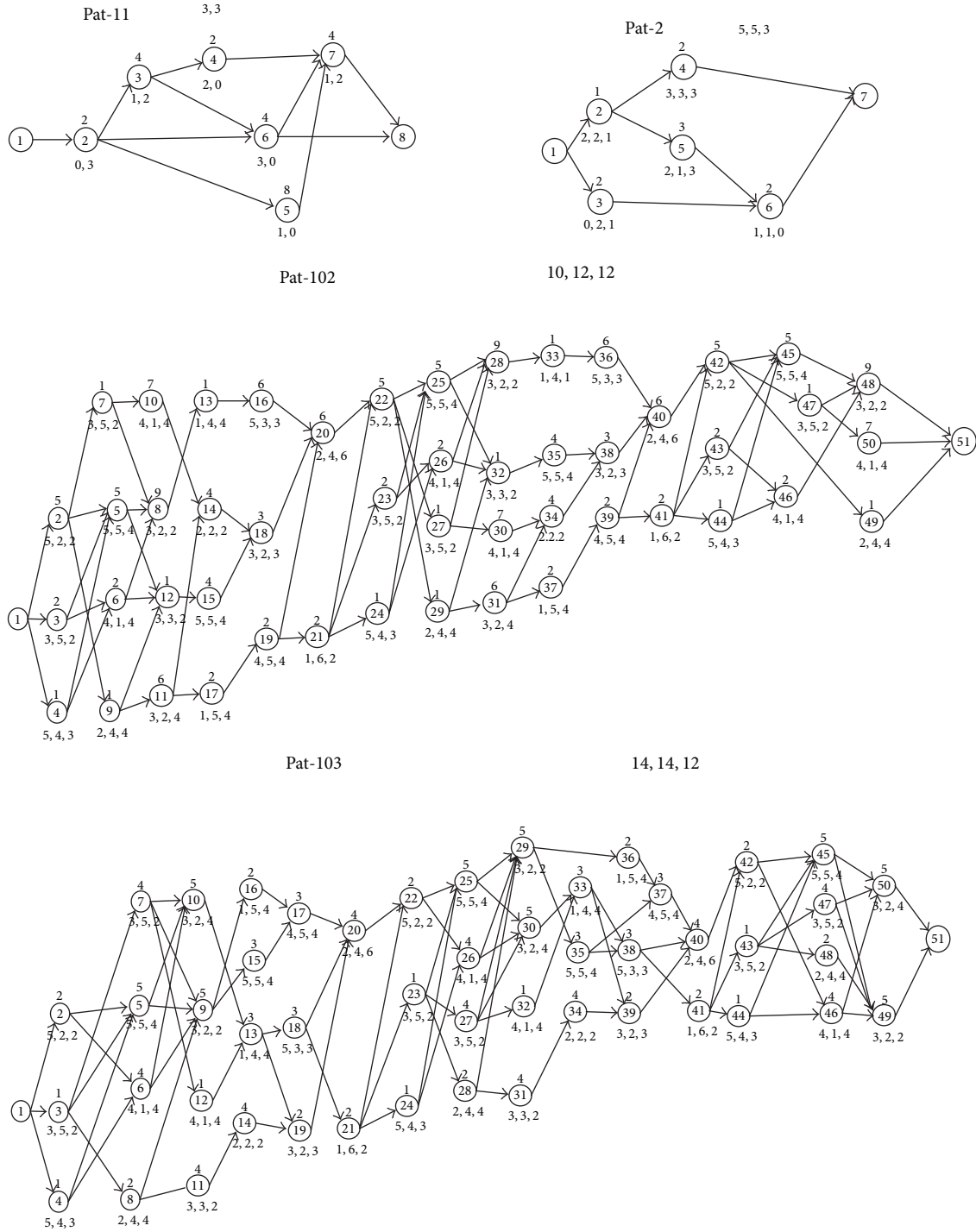


FIGURE 7: Four networks from Patterson set.

However, for RCPSP with preemptive activities, activities are torn up into small pieces and arranged as early as possible to shorten the makespan and minimize the splitting times. For a network in Figure 3(a), an optimal schedule is $\langle 1, 2, 3, 4, 8, 5, 6, 10, 9, 11, 7, 12, 13 \rangle$ with the corresponding Gantt chart shown in Figure 4. Comparing Figures 3(b) and 4, the makespan improvement is 2 time units.

4.2. Problem Decoding. With a feasible schedule, a Gantt chart can be constructed to calculate the makespan (objective function value). Activity i , which is allowed to be splitted into at most d_i parts, can be preempted at any integer time unit and restarted later on at no additional cost [45].

Suppose $t(i, h)$ is the starting time of the h th part of activity i , and $ES(i)$ and $LF(i)$ are the earliest starting time and

TABLE 1: Computational results of RCPSP_PA_SFLA for Patterson set (1/5).

Patterson set					RCPSP_PA_SFLA		
No.	Opt.	Best	Average	Worst	P.R.B.S (%)	A.Runtime (s)	B.A.Ps.
1	19	19	19	19	100	0.275	1—3—5—2—10—6—7—4—11—12—8—9—13—14
2	7	7	7	7	100	0.125	1—2—3—5—6—4—7
3	20	20	20	20	100	0.265	1—3—2—5—4—6—10—11—12—8—9—7—13
4	6	6	6	6	100	0.359	1—3—4—7—2—8—6—11—5—9—10—12—14—16—13—19— 15—18—20—17—21—22
5	7	7	7	7	100	0.35	1—3—4—2—5—9—6—14—7—8—11—10—12—16—19—13— 17—15—18—20—21—22
6	8	8	8	8	100	0.35	1—3—4—8—7—11—2—5—9—6—10—14—12—13—16—15—17— —19—21—18—20—22
7	8	8	8	8	100	0.148	1—3—6—2—7—8—4—5—9
8	11	11	11	11	100	0.142	1—8—3—4—6—2—5—7—9
9	19	19	19	19	100	0.35	1—4—5—3—8—2—14—7—6—13—11—9—10—12—15—16—17— 18
10	14	14	14	14	100	0.16	1—2—5—3—4—6—7—8
11	18	18	18	18	100	0.156	1—2—5—3—4—6—7—8
12	13	12	12	12	100	0.437	1—6—14—2—13—4—11—5—12—8—3—10—17—7—9—16— 18—21—22—19—20—15—23
13	20	20	20	20	100	0.453	1—18—2—5—4—11—13—17—3—10—6—8—9—7—15—16— 20—14—21—12—19—22
14	43	42	42	42	100	0.8	1—4—2—8—3—6—7—9—5—12—14—15—17—11—16—19— 10—13—18—22—23—20—24—21—26— 25—28—27—32—31—30—34—29—33—35
15	43	43	43	43	100	0.781	1—3—6—4—10—2—8—16—5—9—15—7—12—14—11—13— 18—19—17—22—25—20—24—23—21—26—27—28—32—31— 30—29—33—34—35
16	32	32	32	32	100	0.453	1—2—3—4—7—8—6—5—12—16—10—15—11—9—13— 14—18—17—19—20—21—22
17	29	29	29	29	100	0.453	1—2—8—3—4—7—5—9—11—14—6—12—15—10— 13—17—16—18—19—21—20—22
18	41	40	40	40	100	0.453	1—3—2—7—4—6—5—9—12—10—15—13—19—16—8—11— 14—17—18—21—20—22
19	31	31	31	31	100	0.453	1—2—8—3—4—6—7—10—5—12—15—9—16—11—13— 14—18—17—19—20—21—22
20	37	37	37	37	100	0.437	1—2—4—6—12—7—16—3—11—15—18—5—8—9—13—17— —21—10—14—19—20—22
21	48	48	48	48	100	0.453	1—4—2—7—6—11—3—12—8—15—5—10—14—16— —19—9—18—20—13—17—21—22
22	36	36	36.2	37	80	0.453	1—3—2—4—7—8—6—5—10—12—9—16—11—15— 13—14—18—17—19—20—21—22

the latest finishing time. F and D are presenting the scheduled activity set and the no-immediate-predecessor activity set. Then the decoding procedure for a feasible schedule q can be summarized in Algorithm 1.

4.3. Feasible Schedule Generation. A feasible solution is generated mainly depending on the precedence relationship. The generation method is given out in Algorithm 2.

4.4. Crossover Operation. The crossover operation is executed by combining the former part of a schedule S_1 with the latter part of another schedule S_2 to produce a new schedule

S_{new} according to a *randomly selected position* (Pos). To keep the newly generated schedule S_{new} as a feasible one, those corresponding activities in S_2 to the former part of S_1 are eliminated and the rest of the activities, maintaining the original order, are stuck together as the latter part of S_{new} . Figure 5 shows how a new schedule is generated by crossover X_w with X_b within a memplex.

4.5. Repair Strategy. When an infeasible schedule is produced especially by interchange operation, a repair procedure is adopted to keep feasibility. The interchange operation is executed by choosing randomly two interchange positions

TABLE 2: Computational results of RCPSP_PA_SFLA for Patterson set (2/5).

Patterson set							SFLA
No.	Opt.	Best	Average	Worst	P.R.B.S (%)	A.Runtime (s)	B.A.Ps.
23	32	31	31	31	100	0.453	1—2—3—4—7—8—5—10—9—16—6—11—13—12—18— 14—15—17—20—19—21—22
24	40	40	40	40	100	0.468	1—3—2—4—8—10—6—7—5—11—9—12—15—13— 16—14—18—19—17—21—20—22
25	33	33	33	33	100	0.468	1—3—2—6—4—7—5—8—10—9—14—11—12— —13—16—19—17—15—18—20—21—22
26	43	43	43	43	100	0.453	1—3—4—2—6—5—7—11—9—8—12—15—10—13— —14—18—17—16—19—20—21—22
27	36	35	35	35	100	0.48	1—4—3—8—5—2—7—11—6—10—12—13—16—9—15—19— —14—17—21—18—20—22
28	43	42	42	42	100	0.483	1—4—8—12—13—3—5—2—6—7—10—16—9—11—14— 18—15—19—17—21—20—22
29	29	29	29	29	100	0.437	1—4—3—2—8—5—7—6—10—11—14—16—19—12—15— —17—21—9—13—18—20—22
30	32	31	31	31	100	0.45	1—3—4—8—5—11—15—6—12—2—7—10—13—9—14—18— —16—19—21—17—20—22
31	35	35	35	35	100	0.453	1—3—4—2—8—7—6—10—12—5—9—11—14—13—18—16— —15—19—21—17—20—22
32	22	22	22	22	100	0.43	1—3—4—2—6—7—8—10—11—5—9—14—15—12—18— 19—16—13—21—17—20—22
33	31	30	30.1	31	90	0.43	1—4—3—2—6—7—5—8—11—12—16—10—15—9—14—19— —13—18—17—21—20—22
34	30	30	30	30	100	0.43	1—4—3—2—6—11—8—5—15—7—18—21—12—9—10— 16—13—17—14—19—20—22
35	31	31	31	31	100	0.45	1—4—3—5—2—7—6—10—8—11—9—14—15—19—12—16— —17—21—13—18—20—22
36	33	32	32	32	100	0.453	1—4—3—8—2—5—6—7—12—9—13—19—10—14—11— 15—18—16—17—21—20—22
37	28	27	27	27	100	0.484	1—2—7—4—12—3—8—6—5—10—9—11—14—13—15— 16—17—19—21—18—20—22
38	30	30	30	30	100	0.437	1—3—4—5—2—7—12—6—10—13—8—9—15—11—16— 17—19—21—14—18—20—22
39	31	31	31	31	100	0.46	1—4—8—3—2—7—5—6—12—11—9—15—16—18—10— 13—14—17—19—21—20—22
40	31	30	30	30	100	0.453	1—4—3—2—5—7—6—9—12—8—11—15—10—16—13— 14—19—17—20—18—21—22
41	36	36	36	36	100	0.48	1—3—2—5—4—7—8—9—6—10—12—15—11—13—16— 18—17—21—14—19—20—22
42	28	28	28	28	100	0.438	1—3—2—4—5—6—7—8—11—16—10—12—9—13—14— 17—21—15—18—19—20—22
43	41	41	41	41	100	0.453	1—4—2—3—8—7—11—16—6—12—5—10—15—9—14— 13—17—19—18—21—20—22
44	31	31	31	31	100	0.437	1—2—4—3—6—8—10—14—7—5—12—9—13— 11—19—21—15—16—17—18—20—22

(P_1 and P_2) and then interchanging two activities' position. After this execution, a repair mechanism is adopted as shown in Algorithm 3. Figure 6 presents a case-based description.

4.6. Interchange Operation. As stated in §4.5, interchange operation is a swap procedure of two activities in different positions of a schedule. The interchange operation can be depicted in Algorithm 4.

4.7. Stopping Criteria. The maximum iteration times are taken as the stopping rule in this paper.

5. Experimental Results

In this section, the performance of RCPSP_PA_SFLA described in the previous section is evaluated on a standard benchmark set—Patterson set. The set contains 110 cases with

TABLE 3: Computational results of RCPSP_PA_SFLA for Patterson set (3/5).

Patterson set		SFLA					
No.	Opt.	Best	Average	Worst	P.R.B.S (%)	A.Runtime (s)	B.A.Ps.
45	39	39	39	39	100	0.437	1—4—2—3—7—8—6—10—9—5—11—15—12— 14—13—16—19—17—21—18—20—22
46	33	32	32	32	100	0.437	1—4—2—8—3—7—5—11—16—15—6—9—12—13 —10—17—18—21—14—19—20—22
47	35	34	34	34	100	0.437	1—3—4—2—8—7—5—6—12—11—9—10—14— 16—15—13—18—17—21—19—20—22
48	23	23	23	23	100	0.421	1—3—2—6—4—8—10—14—5—7—11—15—12— 16—18—9—13—17—19—20—21—22
49	18	18	18	18	100	0.421	1—3—7—4—8—11—2—5—6—10—9—14—12— 16—19—15—18—21—13—17—20—22
50	25	25	25	25	100	0.406	1—2—4—3—7—5—11—8—15—6—9—10—12 —16—13—19—17—20—14—18—21—22
51	25	24	24	24	100	0.437	1—2—4—8—3—6—5—9—7—11—12—16— 10—13—14—15—17—18—19—20—21—22
52	27	26	26	26	100	0.437	1—4—3—2—8—5—6—9—10—7—11—12— 13—16—14—15—18—17—19—20—21—22
53	28	27	27	27	100	0.453	1—4—3—2—8—5—6—7—9—11—12—10— 13—15—14—16—19—18—17—20—21—22
54	50	50	50	50	100	0.46	1—4—3—2—8—6—7—5—10—9—11—12—13— 14—15—16—19—17—18—21—20—22
55	29	28	28	28	100	0.46	1—2—4—3—5—7—6—8—12—10—9—11—14— 13—15—16—18—19—17—20—21—22
56	27	27	27	27	100	0.453	1—3—4—2—7—6—8—5—11—9—12—10—14— 13—15—16—17—19—18—20—21—22
57	21	21	21	21	100	0.43	1—4—2—3—8—12—5—7—6—10—14—16—9— 19—21—11—13—17—15—20—18—22
58	35	35	35	35	100	0.55	1—2—4—3—8—5—7—12—6—9—15—11—10—19 —16—13—14—17—20—18—22—23—21—24—26—25—27
59	31	31	31	31	100	0.52	1—4—3—8—2—7—5—6—12—9—11—15—20—23 —13—10—17—24—16—14—18—19—21—26—22—25—27
60	39	39	39	39	100	0.58	1—4—2—3—8—7—6—5—9—10—12—11—13—14 —18—22—15—24—16—17—19—20—23—21—26—25—27
61	36	36	36	36	100	0.55	1—3—4—2—8—5—10—7—12—6—9—16—14—11 —13—19—15—17—20—18—21—24—23—26—22—25—27
62	37	37	37	37	100	0.53	1—3—4—2—8—7—5—6—9—10—12—16—13—18 —20—11—15—23—24—14—17—21—19—26—22—25—27
63	40	40	40	40	100	0.55	1—4—3—2—6—5—7—11—16—12—19—8—10— 9—13—14—15—17—18—20—22—23—25—21—24—26—27
64	37	37	37	37	100	0.55	1—3—4—2—7—12—5—6—10—8—11—9—15—14— 13—18—19—16—20—17—21—22—23—24—25—26—27
65	40	40	40	40	100	0.55	1—3—4—2—6—11—7—8—5—9—10—13—16—12— 15—14—17—24—20—23—19—21—25—18—22—26—27
66	38	38	38	38	100	0.53	1—3—2—4—8—6—5—7—9—10—12—11—14—15—13 —16—19—17—24—18—20—22—23—21—26—25—27

5–49 nondummy activities and 1–3 renewable resources. All the optimal solutions (under nonpreemptive scenario) of Patterson set are known. Figure 7 shows 4 networks from Patterson set.

After a pretest, the population size is 100; the size of a memplex is 10; the iteration times within a memplex are 5; the maximum iteration times are 2000. For each case, 50 executions are performed to appraise the results.

The simulation work is conducted on a laptop with a dual 2.4 GHz CPU, 1 G RAM, and Window XP operating system. The algorithm is coded in C program language and compiled in MS.Visual C++ 6.0. The computational results are listed in Tables 1–5. Here, No. is the number of case and Opt. is the known optimum for nonpreemptive RCPSP. Best, average, and worst represent the best achieved solution, the average value of all 50 executions, and the worst achieved

TABLE 4: Computational results of RCPSP_PA_SFLA for Patterson set (4/5).

Patterson set					SFLA		
No.	Opt.	Best	Average	Worst	P.R.B.S (%)	A.Runtime (s)	B.A.Ps.
67	27	27	27	27	100	0.515	1-4-2-7-3-5-6-8-11-9-12-14-10-15-13 -18-16-20-19-23-17-24-21-26-22-25-27
68	41	41	41	41	100	0.515	1-4-3-2-5-7-6-9-8-11-10-15-12-13-14- 19-16-20-23-24-17-18-22-21-26-25-27
69	30	29	29	29	100	0.484	1-4-2-3-5-8-7-11-6-12-10-16-14-9- 13-15-20-17-19-18-24-21-26-22-23-25-27
70	31	31	31	31	100	0.5	1-2-3-5-4-7-6-11-8-12-9-16-15-13- 18-19-23-10-20-14-17-22-26-21-24-25-27
71	32	32	32	32	100	0.515	1-4-3-6-2-5-8-11-12-13-7-20-9-16-10- 14-15-18-17-24-19-21-26-23-22-25-27
72	41	41	41	41	100	0.515	1-2-3-8-5-4-6-7-10-14-11-12-16-9-15- 13-20-18-23-19-26-24-21-17-22-25-27
73	36	36	36.2	37	80	0.515	1-2-4-8-3-7-9-5-16-12-13-20-6-23-22 -11-10-15-18-19-14-21-26-17-24-25-27
74	30	30	30	30	100	0.5	1-4-3-2-7-5-9-12-6-14-8-11-10-13-19 -24-16-23-20-26-15-17-22-18-21-25-27
75	34	33	33	33	100	0.515	1-4-3-8-6-2-7-5-12-11-10-13-16-15-17 -9-19-14-20-23-18-24-21-22-26-25-27
76	43	43	43	43	100	0.5	1-4-3-8-2-7-9-6-14-10-5-11-12-13-16 -19-15-23-18-20-21-17-24-22-26-25-27
77	64	64	64	64	100	0.531	1-2-4-7-6-3-5-12-11-16-20-8-9-15-10 -17-22-13-24-14-18-19-21-23-26-25-27
78	53	52	52	52	100	0.54	1-3-2-4-5-6-7-11-16-8-20-12-9-14-18 -15-17-10-13-22-24-19-23-21-25-26-27
79	45	45	45	45	100	0.546	1-3-2-5-4-6-7-8-11-16-12-9-15-20-10 -17-13-24-19-14-18-21-22-23-26-25-27
80	38	38	38	38	100	0.531	1-4-3-2-5-7-6-12-14-18-11-16-8-9-15 -10-17-24-20-13-19-22-21-23-26-25-27
81	36	36	36	36	100	0.52	1-2-4-6-3-8-7-5-12-9-11-15-14-16-17 -18-24-20-10-22-13-19-23-21-26-25-27
82	34	34	34	34	100	0.531	1-4-3-2-6-7-5-8-11-16-10-20-9-15-17 -13-19-23-26-24-22-12-14-18-21-25-27
83	34	34	34	34	100	0.531	1-2-4-3-5-12-14-18-8-6-7-10-9-11-13 -15-19-16-21-20-17-24-22-23-26-25-27
84	33	33	33	33	100	0.52	1-2-4-6-7-3-5-8-10-12-14-11-18-9-16 -15-17-20-22-24-13-19-21-23-26-25-27
85	31	31	31	31	100	0.515	1-3-2-5-4-7-8-9-6-11-10-13-16-15-20 -17-22-19-24-23-12-14-26-18-21-25-27
86	31	31	31	31	100	0.515	1-3-4-2-7-8-6-10-5-9-15-13-11-19-16 -17-20-23-22-24-26-12-14-18-21-25-27
87	29	29	29	29	100	0.515	1-2-4-3-7-6-9-8-10-15-5-11-12-19-23 -16-13-17-14-24-21-20-18-22-25-26-27
88	40	40	40	40	100	0.531	1-3-4-7-2-6-5-11-8-9-12-10-16-15-19 -14-18-20-23-24-13-17-22-21-26-25-27

solution, respectively. P.R.B.S(%), A.Runtime(s), and B.A.Ps show percentage of optima reached by RCPSP_PA_SFLA, the average runtime for each case, and the best arrangement of activities differently. Those bolded numbers are solutions better than the optimum.

From the simulation work, we can conclude that SFLA, which achieved 23 improved solutions within an average

runtime of 0.521 second, no less than 60 percent success rate and low deviation for 110 cases, is an efficient and effective choice for RCPSP with preemptive activities. The preemptive scenario can help shorten the total makespan of a project. Since no any special improving mechanisms are integrated into SFLA, the improvement space of this algorithm is still very big.

TABLE 5: Computational results of RCPSP_PA_SFLA for Patterson set (5/5).

Patterson sets					SFLA		
No.	Opt.	Best	Average	Worst	P.R.B.S (%)	A.Runtime (s)	B.A.Ps.
89	31	31	31	31	100	0.515	1-4-3-2-6-5-8-11-12-7-14-9-13-10-16 -17-24-23-18-15-19-20-21-22-25-26-27
90	39	39	39	39	100	0.52	1-3-6-4-2-5-8-12-7-9-15-19-23-25-10 -16-17-11-13-20-14-18-24-21-22-26-27
91	35	35	35	35	100	0.531	1-3-7-4-2-5-12-8-16-18-24-15-6-10-11 -14-19-20-9-22-13-17-23-26-21-25-27
92	28	28	28	28	100	0.5	1-3-6-2-4-7-9-8-11-14-18-5-12-20-23 -10-15-13-19-16-17-21-24-26-22-25-27
93	26	26	26	26	100	0.515	1-2-3-4-6-7-5-8-10-11-12-15-14-16-20 -9-18-13-19-22-24-21-25-17-23-26-27
94	36	36	36	36	100	0.531	1-2-4-3-6-8-7-5-11-12-14-18-16-9-19 -10-24-13-20-15-17-21-23-26-22-25-27
95	33	33	33	33	100	0.515	1-4-6-2-3-8-11-7-5-9-12-15-10-13-16 -18-14-17-19-21-20-22-25-24-23-26-27
96	26	26	26	26	100	0.484	1-2-6-4-3-7-8-11-15-10-12-5-18-9-24 -13-16-14-17-22-19-20-21-23-25-26-27
97	30	30	30	30	100	0.484	1-3-4-8-10-5-2-6-7-11-15-12-17-19-9 -22-16-20-13-14-24-18-23-21-25-26-27
98	41	40	40.2	41	80	0.55	1-4-3-2-7-5-10-15-6-19-11-9-14-16-8-18 -20-22-17-24-23-12-21-13-25-26-27
99	37	37	37	37	100	0.61	1-4-3-8-2-5-6-11-14-7-10-9-12-13-15-16 -19-17-20-24-18-21-23-22-25-26-27
100	33	32	32	32	100	0.5	1-3-4-6-8-2-7-5-11-10-16-12-9-13-15-14 -20-19-17-18-22-23-21-24-26-25-27
101	75	75	75.4	76	60	1.2	1-2-4-5-3-6-7-8-11-10-13-9-12-16-14-15 -18-17-19-21-24-20-22-23-25-28-27-26- 31-33-30-29-34-32-37-39-36-35-38-40- 41-44-42-45-43-47-46-48-49-50-51
102	83	83	83	83	100	1.11	1-3-2-4-7-6-5-8-9-10-11-13-16-14-12 -15-18-17-19-21-24-20-23-26-22-27- 25-29-30-31-28-32-35-34-33-36-38-37- 39-40-41-42-43-49-44-46-45-47-50-48-51
103	56	56	56	56	100	1.07	1-3-8-4-2-5-7-6-10-9-15-11-12-14-16 -13-19-17-18-21-23-27-20-22-24-32-26 -25-30-28-29-31-34-36-35-37-33-39- 38-41-43-40-44-48-47-42-46-45-49-50-51
104	79	79	79	79	100	1.156	1-2-3-9-6-4-5-7-10-8-11-14-13-17-12 -15-19-16-20-18-21-22-24-25-23-28-34 -27-29-26-33-30-31-32-35-36-38-40-37 -39-41-43-42-44-47-49-46-45-48-50-51
105	76	76	76.2	77	80	1.17	1-2-4-3-5-8-10-6-13-16-7-11-15-19- 9-12-14-17-21-18-20-23-22-24-26-25-28 -30-27-29-33-32-31-35-36-34-38-37-41- 39-43-40-42-44-45-50-46-47-49-48-51
106	60	60	60	60	100	1	1-2-3-4-6-8-7-5-14-15-16-18-19-10-9- 11-12-13-20-23-22-17-25-21-24-26-27-28 -31-30-29-32-33-35-39-40-34-36-38-48-41 -37-43-44-45-42-47-50-46-49-51
107	78	78	78	78	100	1.04	1-3-2-6-10-12-4-7-11-5-13-8-9-16-15-18 -14-17-19-20-25-21-22-24-23-26-29-27-28 -31-36-35-37-32-38-41-45-49-30-43-33-40 -34-39-42-44-50-46-47-48-51
108	61	61	61	61	100	1	1-5-2-3-4-9-6-7-15-18-8-14-17-10-12-19 -20-22-11-13-16-23-21-24-25-26-30-27-28 -29-31-34-32-35-40-43-33-39-37-44-36-38 -42-41-48-45-46-47-50-49-51

TABLE 5: Continued.

Patterson sets							SFLA
No.	Opt.	Best	Average	Worst	P.R.B.S (%)	A.Runtime (s)	B.A.Ps.
109	60	58	58	58	100	1.031	1—2—3—6—9—5—4—11—8—7—13—15—16—18—23—10 —12—14—19—17—22—20—25—21—24—26—27—28—29 —31—30—33—36—32—34—37—35—42—45—38—39—41 —40—47—44—46—49—43—48—50—51
110	50	50	50	50	100	0.953	1—2—3—4—5—12—7—8—6—10—9—19—16—15—11—17 —14—13—25—18—22—20—23—26—29—21—30—31— 24—37—27—28—33—35—32—40—36—34—44—38— 39—42—41—47—50—43—45—46—48—49—51
Average runtime						0.521	

6. Conclusion

In this paper, we presented a shuffled frog leaping algorithm for preemptive project scheduling problems. Through an extensive test on the proposed algorithm with the well-known Patterson benchmark set, we can conclude that SFLA is among the best algorithms for project scheduling problems. With the preemptive assumption, the traditional project scheduling problems could be further improved. The future research could focus on taking into account the changeable resources and the integration with other optimization problems such as vehicle routing problem or supplier selection problem.

Acknowledgments

This research is supported by Natural Science Foundation of China (nos. 71301147, 71202140, 71302051, and 71301148), Humanities and Social Sciences Project of Ministry of Education of China (nos. 12YJCZH065 and 10YJC630009), Postdoctoral Foundation of China (nos. 2012M511620, 2011M501210, and 2012T50651), and the Science and Technology Project of Zhejiang Province (no. 2012C25078).

References

- [1] X. Wang and W. Huang, "Fuzzy resource-constrained project scheduling problem for software development," *Wuhan University Journal of Natural Sciences*, vol. 15, no. 1, pp. 25–30, 2010.
- [2] L. Yan, B. Jinsong, H. Xiaofeng, and J. Ye, "A heuristic project scheduling approach for quick response to maritime disaster rescue," *International Journal of Project Management*, vol. 27, no. 6, pp. 620–628, 2009.
- [3] C. Bai, M. Gershon, and X. Wei, "Fuzzy critical chain method based on genetic algorithm for project scheduling," *Information B*, vol. 13, no. 3, pp. 877–892, 2010.
- [4] C. Bai, M. Gershon, and X. Wei, "Comparison of fuzzy buffer size algorithms in critical chain scheduling," *Information B*, vol. 13, no. 3, pp. 893–903, 2010.
- [5] M. Mobini, Z. Mobini, and M. Rabbani, "An Artificial Immune Algorithm for the project scheduling problem under resource constraints," *Applied Soft Computing Journal*, vol. 11, no. 2, pp. 1975–1982, 2011.
- [6] A. Agarwal, S. Colak, and S. Erenguc, "A Neurogenetic approach for the resource-constrained project scheduling problem," *Computers and Operations Research*, vol. 38, no. 1, pp. 44–50, 2011.
- [7] H. Zhang, H. Li, and C. M. Tam, "Particle swarm optimization for resource-constrained project scheduling," *International Journal of Project Management*, vol. 24, no. 1, pp. 83–92, 2006.
- [8] X. Pan and L. Jiao, "Multi-agent social evolutionary algorithm for project optimization scheduling," *Journal of Computer Research and Development*, vol. 45, no. 6, pp. 998–1003, 2008.
- [9] J. Gonçalves, J. Mendes, and M. Resende, "A genetic algorithm for the resource constrained multi-project scheduling problem," *European Journal of Operational Research*, vol. 189, no. 3, pp. 1171–1190, 2008.
- [10] Y. Xu, L. Wang, and Y. Yang, "Dynamic vehicle routing using an improved variable neighborhood search algorithm," *Journal of Applied Mathematics*, vol. 2013, Article ID 672078, 12 pages, 2013.
- [11] S. L. Tilahun and H. C. Ong, "Modified firefly algorithm," *Journal of Applied Mathematics*, vol. 2012, Article ID 467631, 12 pages, 2012.
- [12] L. Zhang, Y. Xu, and Y. Liu, "An elite decision making harmony search algorithm for optimization problem," *Journal of Applied Mathematics*, vol. 2012, Article ID 860681, 15 pages, 2012.
- [13] L. Yan, B. Jinsong, H. Xiaofeng, and J. Ye, "A heuristic project scheduling approach for quick response to maritime disaster rescue," *International Journal of Project Management*, vol. 27, no. 6, pp. 620–628, 2009.
- [14] D. Golenko-Ginzburg and A. Gonik, "A heuristic for network project scheduling with random activity durations depending on the resource allocation," *International Journal of Production Economics*, vol. 55, no. 2, pp. 149–162, 1998.
- [15] H. Chtourou and M. Haouari, "A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling," *Computers and Industrial Engineering*, vol. 55, no. 1, pp. 183–194, 2008.
- [16] M. Vanhoucke, "Setup times and fast tracking in resource-constrained project scheduling," *Computers and Industrial Engineering*, vol. 54, no. 4, pp. 1062–1070, 2008.
- [17] C. Xu, A. Li, and X. Li, "Multi-project scheduling algorithm based on resource push-pull technology," *Computer Integrated Manufacturing Systems*, vol. 16, no. 6, pp. 1246–1254, 2010.
- [18] D. Krüger and A. Scholl, "A heuristic solution framework for the resource constrained (multi-)project scheduling problem with sequence-dependent transfer times," *European Journal of Operational Research*, vol. 197, no. 2, pp. 492–508, 2009.
- [19] R. Kolisch, "Efficient priority rules for the resource-constrained project scheduling problem," *Journal of Operations Management*, vol. 14, no. 3, pp. 179–192, 1996.

- [20] V. Valls, S. Quintanilla, and F. Ballestín, "Resource-constrained project scheduling: a critical activity reordering heuristic," *European Journal of Operational Research*, vol. 149, no. 2, pp. 282–301, 2003.
- [21] J. Mendes, J. Gonçalves, and M. Resende, "A random key based genetic algorithm for the resource constrained project scheduling problem," *Computers and Operations Research*, vol. 36, no. 1, pp. 92–109, 2009.
- [22] N. Pan, P. Hsiao, and K. Chen, "A study of project scheduling optimization using Tabu Search algorithm," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 7, pp. 1101–1112, 2008.
- [23] D. S. Yamashita, V. A. Armentano, and M. Laguna, "Scatter search for project scheduling with resource availability cost," *European Journal of Operational Research*, vol. 169, no. 2, pp. 623–637, 2006.
- [24] A. Lova, P. Tormos, M. Cervantes, and F. Barber, "An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes," *International Journal of Production Economics*, vol. 117, no. 2, pp. 302–316, 2009.
- [25] K. W. Kim, Y. S. Yun, J. M. Yoon, M. Gen, and G. Yamazaki, "Hybrid genetic algorithm with adaptive abilities for resource-constrained multiple project scheduling," *Computers in Industry*, vol. 56, no. 2, pp. 143–160, 2005.
- [26] W. Chen, Y. Shi, H. Teng, X. Lan, and L. Hu, "An efficient hybrid algorithm for resource-constrained project scheduling," *Information Sciences*, vol. 180, no. 6, pp. 1031–1039, 2010.
- [27] M. Ranjbar, "Solving the resource-constrained project scheduling problem using filter-and-fan approach," *Applied Mathematics and Computation*, vol. 201, no. 1-2, pp. 313–318, 2008.
- [28] L. Deng, Y. Lin, and M. Chen, "Hybrid ant colony optimization for the resource-constrained project scheduling problem," *Journal of Systems Engineering and Electronics*, vol. 21, no. 1, pp. 67–71, 2010.
- [29] Z. Huang, "Quantum-inspired evolutionary algorithm for resources constrained project scheduling problem," *Computer Integrated Manufacturing Systems*, vol. 15, no. 9, pp. 1779–1822, 2009.
- [30] V. V. Peteghem and M. Vanhoucke, "A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 201, no. 2, pp. 409–418, 2010.
- [31] Y. Ying, Y. Shou, and M. Li, "Hybrid genetic algorithm for resource constrained multi-project scheduling problem," *Journal of Zhejiang University*, vol. 43, no. 1, pp. 23–27, 2009.
- [32] J. Montoya-Torres, E. Gutierrez-Franco, and C. Pirachicán-Mayorga, "Project scheduling with limited resources using a genetic algorithm," *International Journal of Project Management*, vol. 28, no. 6, pp. 619–628, 2010.
- [33] V. Valls, F. Ballestín, and S. Quintanilla, "A hybrid genetic algorithm for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 185, no. 2, pp. 495–508, 2008.
- [34] K. Bouleimen and H. Lecocq, "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version," *European Journal of Operational Research*, vol. 149, no. 2, pp. 268–281, 2003.
- [35] S. Elloumi and P. Fortemps, "A hybrid rank-based evolutionary algorithm applied to multi-mode resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 205, no. 1, pp. 31–41, 2010.
- [36] M. Al-Fawzan and M. Haouari, "A bi-objective model for robust resource-constrained project scheduling," *International Journal of Production Economics*, vol. 96, no. 2, pp. 175–187, 2005.
- [37] S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 207, no. 1, pp. 1–14, 2010.
- [38] C. Fang and L. Wang, "Survey on resource-constrained project scheduling," *Control and Decision*, vol. 25, no. 5, pp. 641–656, 2010.
- [39] R. Kolisch and R. Padman, "An integrated survey of deterministic project scheduling," *Omega*, vol. 29, no. 3, pp. 249–272, 2001.
- [40] F. Ballestín and R. Blanco, "Theoretical and practical fundamentals for multi-objective optimisation in resource-constrained project scheduling problems," *Computers and Operations Research*, vol. 38, no. 1, pp. 51–62, 2011.
- [41] W. Herroelen and R. Leus, "Project scheduling under uncertainty: survey and research potentials," *European Journal of Operational Research*, vol. 165, no. 2, pp. 289–306, 2005.
- [42] J. Wglarz, J. Józefowska, M. Mika, and G. Waligóra, "Project scheduling with finite or infinite number of activity processing modes—a survey," *European Journal of Operational Research*, vol. 208, no. 3, pp. 177–205, 2011.
- [43] J. Buddhakulsomsiri and D. S. Kim, "Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting," *European Journal of Operational Research*, vol. 175, no. 1, pp. 279–295, 2006.
- [44] J. Buddhakulsomsiri and D. S. Kim, "Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting," *European Journal of Operational Research*, vol. 178, no. 2, pp. 374–390, 2007.
- [45] L. Deng and Y. Lin, "Particle swarm optimization for resource-constrained project scheduling problems with activity splitting," *Control and Decision*, vol. 23, no. 6, pp. 681–688, 2008.
- [46] X. Luo, D. Wang, and J. Tang, "Project scheduling problem involving time-splittable tasks," *Journal of Northeastern University*, vol. 27, no. 9, pp. 961–964, 2006.
- [47] M. Eusuff and K. Lansey, "Optimization of water distribution network design using the shuffled frog leaping algorithm," *Journal of Water Resources Planning and Management*, vol. 129, no. 3, pp. 210–225, 2003.
- [48] M. Eusuff, K. Lansey, and F. Pasha, "Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization," *Engineering Optimization*, vol. 38, no. 2, pp. 129–154, 2006.
- [49] A. Rahimi-Vahed, M. Dangchi, H. Rafiei, and 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*, vol. 41, no. 11-12, pp. 1227–1239, 2009.
- [50] A. Rahimi-Vahed and A. H. Mirzaei, "A hybrid multi-objective shuffled frog-leaping algorithm for a mixed-model assembly line sequencing problem," *Computers and Industrial Engineering*, vol. 53, no. 4, pp. 642–666, 2007.
- [51] B. Amiri, M. Fathian, and A. Maroosi, "Application of shuffled frog-leaping algorithm on clustering," *International Journal of Advanced Manufacturing Technology*, vol. 45, no. 1-2, pp. 199–209, 2009.
- [52] H. Elbehairy, E. Elbeltagi, T. Hegazy, and K. Soudki, "Comparison of two evolutionary algorithms for optimization of bridge deck repairs," *Computer-Aided Civil and Infrastructure Engineering*, vol. 21, no. 8, pp. 561–572, 2006.

- [53] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms," *Advanced Engineering Informatics*, vol. 19, no. 1, pp. 43–53, 2005.
- [54] N. Wang, X. Li, and X. Chen, "Fast three-dimensional Otsu thresholding with shuffled frog-leaping algorithm," *Pattern Recognition Letters*, vol. 31, no. 13, pp. 1809–1815, 2010.